ABSTRACT SYNTAX AND INTERPRETATION OF PL/I

K. WALK
K. ALBER
M. FLECK
H. GOLDMANN
P. LAUER
E. MOSER
P. OLIVA
H. STIGLEITNER
G. ZEISEL

IBM LABORATORY VIENNA
NOTE

This document is not an official PL/I Language Specification. For information concerning the official interpretation the reader is referred to the PL/I Language Specifications, Form No. Y31-6603-1.
ABSTRACT SYNTAX AND INTERPRETATION OF PL/I

by

K. WALK
K. ALBIER
M. FLECK
H. GOLLNANN
P. LAUER
E. MOSER
P. OLIVA
H. STIGLITZER
G. ZEISEL

ABSTRACT

This document provides the formal definition of the abstract syntax and interpretation of PL/I. The method used is based on the definition of an abstract machine which is characterized by the set of its states and its state transitions. A PL/I program specifies an initial state of the machine and the subsequent behaviour of the machine is said to define the interpretation of the PL/I program. PL/I programs are specified in an abstract form according to the abstract syntax of PL/I. The translation of the concrete representation of PL/I programs into the abstract form is defined in "Translation of PL/I into Abstract Syntax" (TR 25.097).
NOTE

This document is not an official PL/I Language Specification. For information concerning the official interpretation the reader is referred to the PL/I Language Specifications, Form No. Y33-6063-1.
This document is an updated version of:


It is part of a series of documents (ULD Version III) presenting the formal definition of syntax and semantics of PL/I:


The method and notation used in these documents are essentially taken over from the first version of a formal definition of PL/I issued by the Vienna Laboratory:


An outline of the method is given in:


This document also contains the appropriate references to the relevant literature. The basic ideas and their application to PL/I have been made available through several workshops on the formal definition of PL/I, and presentations and publications inside and outside IBM. The method is demonstrated by application to a suitably tailored subset of PL/I in:


The language defined in the present version is PL/I as specified in the PL/I Language Specifications, Form No. Y33-5003-1, with the addition of attention handling, input stream and string scanning, and file variables.

The present document will be made subject to validation by the PL/I Language Department, Harwell.
CONTRIBUTION TO THE DOCUMENT

Project manager: K. WALK

Authors: Authors of the present version are listed in the following with their main contributions indicated by chapters. It should be stated that credit must be given also to those authors of previous versions who have not been engaged in producing the present document. The contributions of these authors are listed in the respective documents of the previous versions.

K. ALBER 2., 3.1, 3.2, 3.3, 4., 5., 6., 12.4
M. TLECK 11.1, 11.2, 11.5
H. GOLDMANN 12. (except 12.4)
P. LAUER 9.
E. MOSER 1.
P. OLIVA 3.6, 11.3, 11.4, 11.6, 11.7, 11.8
H. STIGLEITNER 13.
K. WALK 3.4, 3.7, 7., 8.
G. ZEISEL 3.5, 10.

PRODUCTION

This document was prepared by means of automated text-processing systems. TEXT 360 was used for processing the prose parts. The formatting, indexing, cross-referencing, and updating of formula texts was handled by means of FORMULA 360.

FORMULA 360 is a syntax-controlled formula processing system which was developed in the Vienna Laboratory especially to facilitate the production and maintenance of PL/I Formal Definition documents. The achievements of K.F. KOCH in the overall design and implementation of FORMULA 360 are acknowledged in particular. Essential components of the system are due to G. URSCHLER (syntactical decomposition of formulas) and E. MOSER (formula input checker). H. Hoja and G. Zeisel contributed to the clarification and formulation of the required formatting processes.

Coordination: J. Schwarzenberger, M. Stadler
Technical control: K.F. Koch, E. Moser, M. Stadler
Data transcription: Miss W. Schatzl and sub-contractors
System support: H. Chladak, G. Lehmayer
# ABSTRACT SYNTAX AND INTERPRETATION OF PL/I

## CONTENTS

1. NOTATION AND CONVENTIONS ................................................................. 1
   1.1 Notation of Meta-Expressions ....................................................... 1
   1.1.1 Conditional expressions ....................................................... 1
   1.1.2 Functional composition ....................................................... 7
   1.1.3 The General Class of Objects .................................................. 2
   1.1.3.1 Informal Introduction .................................................... 2
   1.1.3.2 Formal treatment .......................................................... 3
   1.1.3.3 The μ-function ............................................................ 4
   1.1.3.4 The class of objects used for the formal definition of PL/I ....... 5
   1.1.4 Lists .................................................................................. 8
   1.1.5 Equality .............................................................................. 10
   1.1.6 Truth values, logical operators and quantifiers ......................... 10
   1.1.6.1 Truth values .................................................................... 10
   1.1.6.2 Logical operators ............................................................ 10
   1.1.6.3 Quantifiers ....................................................................... 11
   1.1.6.4 Jota-operator .................................................................... 11
   1.1.7 Arithmetic operators and relations ........................................... 11
   1.1.7.1 Operators .......................................................................... 11
   1.1.7.2 Relations .......................................................................... 12
   1.1.7.3 Arithmetic functions .......................................................... 13
   1.1.8 Set operators, relations and notation for sets ............................. 16
   1.1.8.1 Set operators ..................................................................... 16
   1.1.8.2 Relations .......................................................................... 16
   1.1.8.3 Notation for sets ................................................................ 16
   1.1.9 Rules of precedence ................................................................. 16
   1.1.10 Naming ................................................................................. 18
   1.2 Definitions .................................................................................... 18
   1.2.1 Metavariables .......................................................................... 18
   1.2.2 Abbreviations ......................................................................... 18
   1.2.3 Formulas ................................................................................ 19
   1.2.3.1 Function definitions ........................................................... 19
   1.2.3.2 Instruction definitions ......................................................... 19
   1.2.3.3 Predicate definitions .......................................................... 19
   1.2.3.4 Logical statements .............................................................. 21
   1.2.5 References .............................................................................. 21
   1.3 The Control of the PL/I Machine ..................................................... 22
   1.3.1 Informal introduction ............................................................... 22
   1.3.2 The abstract syntax of control-trees and the control part ............ 22
   1.3.3 The functions term-node, compute and instruction definitions ..... 24
   1.3.4 A special notation for meta-expressions denoting control-trees ... 27
   1.3.4.1 Informal introduction ......................................................... 27
   1.3.4.2 Format and meaning of meta-expressions denoting control-trees 28
   1.3.4.3 Formalization of the meaning of control-tree expressions ....... 29
   1.4 The Treatment of Program Errors .................................................... 30
   1.5 Generally Used Simple Instructions ............................................... 31

2. ABSTRACT SYNTAX OF PROGRAM ......................................................... 1
   2.1 Program, Procedure Body ............................................................... 2
   2.2 Declarations ................................................................................ 2
   2.2.1 General ................................................................................ 2
   2.2.2 Aggregate attributes ................................................................ 4
   2.2.2.1 Aggregates in variable declarations .................................... 4
   2.2.2.2 Aggregates in parameter descriptors .................................... 6
   2.2.2.3 Aggregates in generic parameter descriptors ....................... 7
   2.2.2.4 Aggregates in allocate statements ....................................... 8
   2.2.3 Pictures ................................................................................. 8
   2.2.4 Formats ................................................................................. 10
   2.3 Statements .................................................................................... 11
   2.3.1 Block, group ......................................................................... 11
   2.3.2 Flow of control statements ....................................................... 12
   2.3.3 Storage manipulating statements .............................................. 13
   2.3.4 Condition and attention handling statements ......................... 14
   2.3.5 Input and output statements ..................................................... 15
ABSTRACT SYNTAX AND INTERPRETATION OF PL/I

2.4 Expressions .......................................................... 18
4.1 The Initial State ..................................................... 2
4.2 Initiating and Terminating Actions of Program Interpretation .... 2
4.3 The Prepass ............................................................ 5
5. TASKS ........................................................................ 1
5.1 Attaching of Tasks ...................................................... 1
5.2 Termination of Tasks ................................................... 5
5.3 The Wait Statement ...................................................... 9
5.3.1 The wait mechanism .................................................. 9
5.3.2 Completion of input and output events ......................... 13
5.4 The Delay Statement .................................................... 15
6. FLOW OF CONTROL WITHIN A SINGLE TASK .................. 1
6.1 Block Activation ......................................................... 1
6.1.1 Unique qualification of names .................................... 1
6.1.2 Block prologue ....................................................... 3
6.1.2.1 Order of evaluating declarations .............................. 3
6.1.2.2 Denotations of the single declarations ....................... 6
6.1.2.3 Evaluation of aggregate attributes ............................ 8
6.1.3 Block epilogue ........................................................ 11
6.2 Flow of Control .......................................................... 6
6.2.1 The dump D .......................................................... 6
6.2.2 The block activation name BA ...................................... 7
6.2.3 The epilogue information EI ....................................... 8
6.2.4 The control information CI ........................................ 8
6.3 State Components and Computation of the PL/I Machine ......... 1
6.3.1 Parallel Actions ..................................................... 3
6.3.1.1 The parallel action part PA .................................... 3
6.3.1.2 The task-event specification TE ................................. 4
6.3.1.3 The time and date part TD ...................................... 4
6.3.1.4 The event trace ET .............................................. 5
6.3.1.5 The event trace ET .............................................. 6
6.3.2 The current task-event name TN .................................. 7
6.3.3 The task-event specification TE ................................... 7
6.3.4 The aggregatE directory AG ....................................... 7
6.3.5 The epilogue information EI ....................................... 7
6.3.6 The control information CI ........................................ 7
6.4 Input and Output ....................................................... 23
6.4.1 The file directory ID ................................................ 23
6.4.2 The file uninc directory FU ........................................ 24
6.4.3 The external storage ES ........................................... 24
6.4.4 The message part M ................................................ 27
6.5 State Components for Attentions and Conditions ................. 18
6.5.1 Attention directory AN ............................................. 18
6.5.2 Attention enabling state EN ....................................... 19
6.5.3 Attention environment directory EN .............................. 19
6.5.4 Condition selectors .................................................. 19
6.5.5 The condition state CS ............................................ 21
6.6 The Computation of the PL/I Machine .............................. 29
6.6.1 The computation step ............................................... 30
6.6.2 The computation step ............................................... 31
6.6.3 The interrupt step .................................................. 32
6.6.4 Optimization ........................................................ 33
4. FLOW OF CONTROL WITHIN A SINGLE TASK .................. 1
4.1 The Initial State ..................................................... 1
4.2 Initiating and Terminating Actions of Program Interpretation .... 2
4.3 The Prepass ............................................................ 5

30 April 1969
9. DATA, OPERATIONS AND CONVERSIONS
9.1 Values, Value Representations, Operands, and Operators
9.1.2 Values
9.1.2.1 Numeric values
9.1.2.2 Character values
9.1.2.3 Bit values
9.1.2.4 Event values
9.1.2.5 Matching values and attributes
9.1.3 Representing and retrieving scalar values
9.1.4 Operands
9.1.5 Operators
9.2 Auxiliary Definitions on Data Attributes
9.2.1 Special data attributes and classes of data attributes
9.2.2 Implementation-defined limits associated with arithmetic data attributes
9.3 Evaluation of Infix Expressions
9.3.1 Value of the result
9.3.1.1 Arithmetic operations
9.3.1.2 Comparison operations
9.3.1.3 String comparison
9.3.1.4 Pointer comparison
9.3.1.5 Bit string operations
9.3.2 Data attributes of the result
9.3.2.1 Target attributes for conversion of operands
9.3.2.2 Higher arithmetic characteristics
9.3.2.3 Result attributes
9.4.2 The special case of integer constant exponentiation
9.4 Evaluation of Prefix Expressions
9.4.2 Value of the result
9.4.2.1 Arithmetic prefix operations
9.5 Conversion
9.5.1 Value conversion
9.5.1.1 Numeric to character conversion
9.5.1.2 Numeric to bit conversion
9.5.1.3 Character to numeric conversion
9.5.1.4 Character to bit conversion
9.5.1.5 Bit to numeric conversion
9.5.1.6 Bit to character conversion
9.5.2 Area conversion
9.5.3 Completion of target attributes
9.5.3.1 Completion of arithmetic attributes
9.5.3.2 Completion of string attributes
9.5.3.3 Completion of area attributes
9.5.3.4 Transformation into an arithmetic attribute
9.5.3.5 Transformation into a string attribute
9.6 Pictures
9.6.1 Auxiliary definitions
9.6.1.1 Explicit picture attributes
9.6.1.2 Transformation of an explicit picture into a picture
9.6.1.3 Transformation of a picture into an explicit picture
9.6.1.4 Classification of pictures and picture specifications
9.6.1.5 Predicates classifying pictures
9.6.1.6 Predicates classifying picture specifications
9.6.1.7 Functions computing and counting digit positions
9.6.2 Checking picture attributes
9.6.2.1 Three classes of binary pictures
9.6.2.2 Proper subfields
9.6.2.3 Additional restrictions for sterling subfields
9.6.2.4 Correct drifting
ABSTRACT SYNTAX AND INTERPRETATION OF PL/I

9.6.3 Representation in pictured form
9.6.3.1 Representation of numeric pictures
9.6.3.2 Representation of subfields
9.6.3.2.1 Binary subfields
9.6.3.2.2 Decimal and sterling subfields
9.6.3.2.2.1 Decomposition into a number list
9.6.3.2.2.2 Representation of number lists
9.6.3.2.2.3 Representation of the sign
9.6.3.2.2.4 Zero suppression and drifting
9.6.3.2.2.5 Representation of character pictures
9.6.3.3 Composition of subfields and linearization
9.6.3.4 Representation of character pictures
9.6.3.5 Conversion from numeric picture to character string
9.6.4 Value of a picture
9.6.4.1 Value of a numeric picture
9.6.4.2 Testing of string values against picture attributes

10. ATTENTIONS AND CONDITIONS
10.1 Enabling and Disabling
10.1.1 Enabling and disabling of conditions
10.1.1.1 Block and procedure prefixes
10.1.1.2 Statement prefixes
10.1.2 Enabling and disabling of attentions
10.1.2.1 Enable statement
10.1.2.2 Disable statement
10.2 Condition Action
10.2.1 Standard system action for conditions
10.2.2 Interpretation of on statement
10.2.3 Interpretation of revert statement
10.3 Attention Activation
10.3.1 Asynchronous interrupt
10.4 Condition Activation
10.4.1 Interpretation of the condition call
10.4.2 Interpretation of condition action
10.4.3 Signal statement
10.4.4 Special condition activations
10.4.4.1 Activation of check condition
10.4.4.2 Activation of conversion condition
10.4.4.3 Activation of i/o conditions
10.5 Condition Builtin Function Status

11. INPUT AND OUTPUT
11.1 Data Set Mapping
11.1.1 Structure of inner data sets
11.1.2 Details of mapping
11.1.3 Data set activity
11.1.4 Correspondence between backwards and forwards access
11.1.5 Mapping properties related with the file attributes INPUT, OUTPUT, and UPDATE
11.1.6 Properties necessary for data transmission
11.1.6.1 Positioning, reading, and deleting
11.1.6.2 Rewriting
11.1.6.3 Writing
11.2 Basic Access to Data
11.3 Initiation and Termination of a File
11.3.1 Open and close statement
11.3.2 Implicit opening
11.3.2.1 Opening with a file option
11.3.2.2 Opening of a standard system print file
11.3.2.3 Opening...11.3.2.4 Closing
11.3.5 Data set label processing
11.4 Initiation of I/O Statements
11.4.1 Interpretation of statement options
11.4.2 Display and record handling statements
11.4.2.1 Locking and unlocking by exclusive files
11.4.3 Get and put statements

11.5 Functions on picture attributes
10. ATTENTIONS AND CONDITIONS
10.1 Enabling and Disabling
10.1.1 Enabling and disabling of conditions
10.1.1.1 Block and procedure prefixes
10.1.1.2 Statement prefixes
10.1.2 Enabling and disabling of attentions
10.1.2.1 Enable statement
10.1.2.2 Disable statement
10.2 Condition Action
10.2.1 Standard system action for conditions
10.2.2 Interpretation of on statement
10.2.3 Interpretation of revert statement
10.3 Attention Activation
10.3.1 Asynchronous interrupt
10.4 Condition Activation
10.4.1 Interpretation of the condition call
10.4.2 Interpretation of condition action
10.4.3 Signal statement
10.4.4 Special condition activations
10.4.4.1 Activation of check condition
10.4.4.2 Activation of conversion condition
10.4.4.3 Activation of i/o conditions
10.5 Condition Builtin Function Status

11. INPUT AND OUTPUT
11.1 Data Set Mapping
11.1.1 Structure of inner data sets
11.1.2 Details of mapping
11.1.3 Data set activity
11.1.4 Correspondence between backwards and forwards access
11.1.5 Mapping properties related with the file attributes INPUT, OUTPUT, and UPDATE
11.1.6 Properties necessary for data transmission
11.1.6.1 Positioning, reading, and deleting
11.1.6.2 Rewriting
11.1.6.3 Writing
11.2 Basic Access to Data
11.3 Initiation and Termination of a File
11.3.1 Open and close statement
11.3.2 Implicit opening
11.3.2.1 Opening with a file option
11.3.2.2 Opening of a standard system print file
11.3.2.3 Opening...11.3.2.4 Closing
11.3.5 Data set label processing
11.4 Initiation of I/O Statements
11.4.1 Interpretation of statement options
11.4.2 Display and record handling statements
11.4.2.1 Locking and unlocking by exclusive files
11.4.3 Get and put statements

ix
11.5 Record Transmission
11.5.1 Write and locate
11.5.1.1 Buffer transmission
11.5.1.2 Write transmission
11.5.1.3 Buffer allocation
11.5.2 Rewrite transmission
11.5.3.2 Read with area allocation
11.5.4 Ignore read
11.5.5 Delete

11.6 Stream Transmission
11.6.1 Data fields
11.6.1.1 Syntax of constants
11.6.1.2 Parsing and syntactical correctness
11.6.1.4 Arithmetic data fields
11.6.2 Data specifications
11.6.3 Stream output
11.6.3.2 Data-directed output
11.6.3.3 Elementary and edit-directed output
11.6.3.3.2 Control formats
11.6.3.3.3 Data formats
11.6.4 Stream input
11.6.4.2 Data-directed input
11.6.4.3 Elementary and edit-directed input
11.6.4.3.2 Control formats
11.6.4.3.3 Data formats

11.7 Special Cases of Stream Transmission
11.7.1 Copy option
11.7.2 Positioning to tabulators
11.7.3 Check standard system action
11.7.4 Endpage standard system action

11.8 Display Transmission

12. BUILT-IN FUNCTIONS AND PSEUDO-VARIABLES
12.1 Syntactic Expansion of Built-in Function References
12.2 Evaluation of Built-in Function References
12.3 The Aggregate Attributes of Built-in Function References
12.4 Built-in Functions as Values of Entry Variables
12.5 Table of Built-in Functions
12.5.1 Explanation
12.5.1.1 The argument columns id and i
12.5.1.2 Number of arguments of built-in functions
12.5.1.3 The classes of arguments of built-in functions
12.5.1.4 The types of arguments of built-in functions
12.5.1.5 Conversion of arguments of built-in functions
12.5.1.6 Testing of arguments of built-in functions
12.5.1.7 Data attributes of built-in functions
12.5.1.8 Evaluation of the individual built-in functions
12.5.2 The table
12.5.3 The table of the Individual Built-in Functions
12.6.1 General auxiliary definitions
12.6.2 Evaluation of instructions
12.6.2.1 String handling built-in functions
12.6.2.2 Arithmetic generic built-in functions
12.6.2.3 Mathematical generic built-in functions
12.6.2.4 Generic built-in functions for array manipulation
12.6.2.5 Condition built-in functions
12.6.2.6 Logical built-in functions
12.6.2.7 Multitasking built-in functions
12.6.2.8 Miscellaneous built-in functions
12.7 Pseudo-Variables .................................................. 66
12.7.1 Assignment to pseudo-variables .................................. 66
12.7.2 Evaluation of pseudo-generations ............................... 74

13. OPTIMIZATION ......................................................... 1
13.1 Rules for Commoning of Expressions; the NOLOCAL Attribute .......... 1
13.1.1 Commoning step .................................................. 2
13.1.2 Common expressions ............................................ 4
13.1.3 Common references ............................................. 6
13.2 Definition of the NOORDER Attribute and the RECURSIVE Attribute .... 13
13.2.1 The NOORDER attribute ....................................... 14
13.2.2 The RECURSIVE attribute .................................... 18

APPENDIX: CROSS-REFERENCE INDEX
This chapter presents the notation and conventions adopted for the formal definition of PL/I. First, the notation used for forming expressions is introduced. Expressions written in this notation will be called meta-expressions, to distinguish them from expressions of the object language (PL/I). Second, the general class of objects, the definition of the \( \mu \)-function and predicate definitions are described. Next the control of the PL/I machine and the notation and meaning of definitions of instructions are described. The conventions around treatment of program errors are discussed next, in particular the way in which the detection of errors is expressed in the definition. Finally, the definition is given for a couple of simple instructions which are used without reference throughout the document.

1.1. NOTATION AND CONVENTIONS

The syntax and the notational elements used for forming expressions in the meta-language are described in this chapter. A major portion of the notation is adopted from LISP (conditional expressions), from predicate calculus, from arithmetic expressions and relations and from set theory. It is amended by the notation used for the manipulation of objects (the \( \mu \)-function). The definition of meaning is not given below in those cases where it is entirely conformable with the conventional meaning (e.g. for arithmetic operations).

1.1.1 CONDITIONAL EXPRESSIONS

\[(p_1 \rightarrow e_1, p_2 \rightarrow e_2, \ldots, p_n \rightarrow e_n)\]

- \(p_i\) expression denoting a truth value
- \(e_i\) expression denoting some object (the value of \( e_i \))

An alternative form may be used omitting the parentheses and commas:

- \(p_1 \rightarrow e_1\)
- \(p_2 \rightarrow e_2\)
- \(\ldots\)
- \(p_n \rightarrow e_n\)

A conditional expression denotes the value of \( e_i \) where \( i \) is the integer \( 1 \leq i \leq n \), for which \( p_i \) is true and all preceding \( p_j \), \( 1 \leq j < i \), are false. If there is no such integer, the expression has no value.

It is important to note that the left to right order in which the individual conditions \( p_i \) are inspected is relevant. If \( p_i \) is true then a consequence of the above definition is that the value of the successors of \( p_i \), say \( p_k \), \( i < k \leq n \), are irrelevant for the evaluation of the conditional expression and may even be undefined.

1.1.2 FUNCTIONAL COMPOSITION

- functional composition operator

The operator is defined by:
\[(f \ast g) (x-1, \ldots, x-n) = f(g(x-1, \ldots, x-n))\]

\(f\) and \(g\) may be either simple function names or expressions denoting functions. In the latter case, the expression must be parenthesized.

The following rules for omission of parentheses hold:

1) The functional composition operator \(\ast\) binds stronger than functional application, e.g.:

\[f \ast g(x) = (f \ast g)(x)\]

2) \[f-1 \ast f-2 \ast \ldots \ast f-n(x-1, \ldots, x-n) = f-1(f-2(\ldots(f-n(x-1, \ldots, x-n)\ldots))}\]

3) \[f(x-1)(x-2) \ldots (x-n) = (\ldots((f(x-1))(x-2))\ldots)(x-n)\]

1.1.3 THE GENERAL CLASS OF OBJECTS

1.1.3.1 Informal Introduction

Objects are constructs composed of a finite number of immediate components which are themselves objects of similar nature. The names used to name immediate components of objects are called simple selectors.

For selecting immediate components of given objects an operation is introduced which for a given simple selector \(s\) and a given object \(ob\) yields that immediate component of \(ob\) whose name in the formation of \(ob\) is \(s\) if such a component exists at all. (If such a component does not exist the operation yields the null-object which is composed of zero components).

By analogy with functional application the operation is represented by:

\[s(ob)\]

and reads "\(s\) applied to \(ob\)". The application of \(s\) to \(ob\) is said to yield the \(s\)-component of \(ob\).

The successive application of selectors to a given object will, after a finite number of steps, result in an object which does not have any further components. Objects of this kind are called elementary objects. A set of elementary objects \(EO = \{e_0, \ldots, e_n\}\) is presupposed. The application of any selector \(s\) to an elementary object results in the null-object.

Objects which are not elementary are called composite objects.

By analogy with the operation "functional composition", an operation for selectors is introduced by:

\[s-1 \ast s-2 \ast \ldots \ast s-n(ob) = s-1(s-2(\ldots(s-n(ob))\ldots))\]

where \(s-1, s-2, \ldots, s-n\) are simple selectors.

\(s-1 \ast s-2 \ast \ldots \ast s-n\) is called a composite selector.

The case \(n = 0\) is included and the corresponding composite selector is called identity selector \(I\):

\[I(ob) = ob\]

\(s(ob)\) will be called a component of \(ob\) or the \(s\)-component of \(ob\), where \(s\) is a composite selector.

\[\text{---}\]

1) One may consistently think of an object as being composed of an infinite number of components, where only a finite number of these components are not null.

2.1 NOTATION AND CONVENTIONS
Objects may be graphically represented as finite trees with named branches (circle selectors) and elementary objects associated with the terminal nodes (Fig. 1.1).

Fig. 1.1

The object represented by this tree is constructed from the simple selectors \( s_1, s_2, s_3, s_4 \) and elementary objects \( e_1, e_2, e_3 \).

1.1.3.2 Formal treatment

It is assumed that there is a countable set of elementary objects \( S_0 \) and a countable set of simple selectors \( S_0 \). No further assumptions about the sets are made. The symbols \( e_0, e_0-1, e_0-2, \ldots \) and \( s_0, s_0-1, s_0-2, \ldots \) are variables whose range of values are \( S_0 \) and \( S_0 \), respectively. A closed associative binary operation "\( * \)" is introduced in the set \( S_0 \).

The set \( S \) is the set of all possible combinations \( s-1\ast s-2\ast \ldots \ast s-n \) where \( s-1 \in S_0 \) for \( n \geq 1 \) (for \( n=0 \) the identity element \( I \). Distinct combinations of selectors are distinct elements of \( S \). The symbols \( \sigma \) and \( \pi \) with or without indices will be used to denote arbitrary elements of \( S \).

An object is uniquely described by a finite set of pairs \( <\sigma, e_0> \), the characteristic set of the object. (Intuitively the characteristic set of an object is the set of all pairs \( <\sigma, e_0> \) such that the application of \( \sigma \) to the object yields the elementary object \( e_0 \).

Thus the characteristic set of the object shown in Fig. 1.1 is:

\[
<\sigma-1; \ e_0-1>, <\sigma-3\ast \sigma-2; \ e_0-2>, <\sigma-3; \ e_0-3>
\]

Not every such set describes an object. The condition which must hold for a characteristic set \( C \) is:

\[
<\sigma-1; e_0-1>, <\sigma-2; e_0-2> \in C \quad \land \quad <\sigma-1; e_0-1> \neq <\sigma-2; e_0-2> \implies \text{dep}(\sigma-1, \sigma-2)
\]

where:

\[
\text{dep}(\sigma-1, \sigma-2) = \{\sigma \mid (\sigma-1 = \sigma \ast \sigma-2 \lor \sigma-2 = \sigma \ast \sigma-1)\}
\]

This restriction means intuitively that the components of an object must be uniquely named.

The characteristic set of an object \( x \) is denoted by \( C(x) \). The characteristic set of an elementary object \( <\sigma; e_0> \) (consistently with the intuitive interpretation of the characteristic set) and of \( I_0 \), the null-object, \( C(I_0) = \{} \). Note that the identity selector \( I \) can only occur in characteristic sets of elementary objects, since it depends on all selectors, i.e., \( (\forall \sigma) (\text{dep}(I, \sigma)) \), since \( \sigma = \ast I \).

The result of the application of a composite selector \( \sigma \) to an object \( x \) may be specified by its characteristic set \( C(\sigma(x)) \):

\[
C(\sigma(x)) = \{<\sigma; e_0> \mid <\sigma; e_0> \in C(x)\}
\]

1) \( S \) is the free semi-group with identity generated from \( S_0 \) and the operation "\( * \)."
It can be shown that the application of any selector to any object yields an object. It can also be shown that successive application of selectors is the same as the application of the composition of these selectors:

\[ \sigma^{-1} \circ \sigma^{-2}(x) = \sigma^{-1}(\sigma^{-2}(x)) \]

1.1.3.3 The \( \mu \)-function

A rather powerful two place operation is introduced whose operands are an object \( x \) and a pair \( \langle s;y \rangle \) where \( s \) is a composite selector and \( y \) is an object, and whose values are objects.

The application of this operation is represented by the form:

\[ \mu(x;\langle s;y \rangle) \]

The result of the operation is the object \( x \) with the \( s \)-component being replaced by \( y \).

Let \( C(x) \), \( C(y) \) be the characteristic sets of the objects \( x \) and \( y \), respectively. The characteristic set \( C(\mu(x;\langle s;y \rangle)) \) of the result will be specified in terms of \( C(x), C(y) \) and \( s \) as the union of two sets:

\[ C(\mu(x;\langle s;y \rangle)) = \{ w;eo \mid w;eo \in C(x) \land w;eo \notin \text{dep}(s,y) \} \cup \{ w;eo \mid w;eo \in C(y) \} \]

The first set of the right hand side is the characteristic set of \( x \) whose \( s \)-component has been deleted, the second set is the characteristic set of an object which has \( y \) as the \( s \)-component and no other components.

The operation yields an object for any two objects \( x, y \) and any selector \( s \).

The \( \mu \) operation can be used to replace, add, or delete components in a given object.

1.1.3.3.1 Extension of the meaning of the \( \mu \)-function

The purpose of the extension of the \( \mu \)-function is to facilitate the replacement of several components of an object in either specified or unspecified order. One form of the extension will also allow the specification of the set of components to be replaced implicitly.

The following extensions are defined:

1) \( \mu(x;\langle s^{-1};y^{-1} \rangle;\langle s^{-2};y^{-2} \rangle;\ldots;\langle s^{-n};y^{-n} \rangle) \)

The above form is defined iteratively by the equation:

\[ \mu(x;\langle s^{-1};y^{-1} \rangle;\langle s^{-2};y^{-2} \rangle;\ldots;\langle s^{-n};y^{-n} \rangle) = \mu(\mu(x;\langle s^{-1};y^{-1} \rangle;\langle s^{-2};y^{-2} \rangle;\ldots;\langle s^{-n};y^{-n} \rangle)) \]

for the case \( n=0 \) the form is defined by:

\[ \mu(x) = x. \]

2) \( \mu(x;\{ \langle s;y \rangle \mid p(s,y) \}) \)

The second argument of the above form defines a set of pairs \( \langle s;y \rangle \), namely the set of pairs for which a certain proposition \( p(s,y) \) holds. This form is reducible to the form 1 in the following way: if the elements of the set of pairs are written in any linear order and used in the form 1), then the result is the result of the present form provided that the ordering of pairs is not significant. If the order is significant, the result is undefined. The form yields for the empty set.
\( \mu(x; \{\}) = x \)

3) \( \mu_0(<a-1:x-1>, <a-2:x-2>, \ldots, <a-n:x-n>) \)

The meaning of \( \mu_0 \) is defined by the following equation:

\[
\mu_0(<a-1:x-1>, <a-2:x-2>, \ldots, <a-n:x-n>) = \mu(\emptyset; <a-1:x-1>, <a-2:x-2>, \ldots, <a-n:x-n>)
\]

4) \( \mu_0(\{<a:x> | p(a,x)\}) \)

The above form is analogous to 2), it is defined by:

\[
\mu_0(\{<a:x> | p(a,x)\}) = \mu(\emptyset, \{<a:x> | p(a,x)\})
\]

5) \( \delta(x; a-1, a-2, \ldots, a-n) \)

This function deletes the \( a-i \) components from \( x \), it is defined by the following equation:

\[
\delta(x; a-1, a-2, \ldots, a-n) = \mu(x; <a-1:x>, <a-2:x>, \ldots, <a-n:x>)
\]

6) \( \delta(x; \{a \mid p(a)\}) \)

The above form is analogous to 2), it is defined by:

\[
\delta(x; \{a \mid p(a)\}) = \mu(x; \{a \mid p(a)\})
\]

The operators \( \mu_0, \mu, \delta \) and the application of selectors to objects can be used to form expressions denoting objects. One may write in the previous forms expressions denoting objects in places where an object is required, and expressions denoting selectors in places where a selector is required. Functions may be defined whose domains and/or ranges are objects and predicates whose domains are objects.

1.1.3.4 The class of objects used for the formal definition of PL/I

The class of objects used in the present document is characterized in the following by the class of simple selectors and the class of elementary objects, used for the construction of objects.

The following simple selectors are used:

1) selectors denoted by strings of small letters, digits, hyphens, prefixed by \( s- \)

2) the range of the following functions

\[
\text{sel}(\text{idl}) \quad \text{for is-id-list}(\text{idl})
\]

\[
\text{s}(i) \quad \text{for is-intg-val}(i)
\]

\[
\text{mk-id}(\text{cl}) \quad \text{for is-char-val-list}(\text{cl})
\]

\[
\text{elem}(i) \quad \text{for is-intg-val}(i)
\]

The ranges of these one to one functions have no elements in common and do not contain selectors mentioned under 1) and 3).

3) data set names, i.e. the set of elements satisfying the predicate is-ds-n

From these selectors the semigroup of all composite selectors is formed, including the identity function \( I \), which is the unity with respect to functional composition.
Composite selectors satisfy the predicate:

(1) \text{is-selector} = \\
\text{The following elementary objects are used:}
1) objects denoted by strings of capital letters, hyphens and digits without definition place and the special objects *, <>, {}
2) numbers, satisfying the predicate \text{is-num-val}, represented as decimal numbers
3) character values, satisfying the predicate \text{is-char-val}, but not mentioned under 1) (extralingual characters)
4) composite selectors
5) identifiers, satisfying the predicate \text{is-id}
6) unique names, satisfying the predicate \text{is-n}
7) value representations, satisfying the predicate \text{is-vr}
8) data sets, satisfying the predicate \text{is-ds}
9) sizes of storage, satisfying the predicate \text{is-size}
10) pointer values, satisfying the predicate \text{is-ptr-val}
11) finite sets of objects

All objects satisfy the predicate:

(2) \text{is-ob}(x) = \\
T \\
The predicate \text{is-\emptyset} is only satisfied by the null object

(3) \text{is-\emptyset}(x) = \\
(x = \emptyset) \\
The following special predicates and functions are used:

(4) \text{is-id} = \\
Note: This predicate characterizes an infinite class of elementary objects which are different from all constant elementary objects (denoted by names written in capital letters), <>, *, {}, all values, value representations and unique names.

(5) \text{is-n} = \\

6 1. \text{NOTATION AND CONVENTIONS}
Note: This predicate characterizes the enumerably infinite set of unique names \( n_0, n_1, n_2, \ldots \) which are all elementary objects.

(6) \( \text{is-ds-n} = \)

Note: This predicate characterizes the enumerably infinite set of data set names.

(7) \( \text{is-set} = \)

\( \text{is-ob-set} \)

Note: The meaning of this predicate is explicated in 1.2.3.3.

The above mentioned special functions are now formalized:

(8) \( \text{sel(idl)} = \)

(9) \( \text{is-id-list(idl-1)} \& \text{is-id-list(idl-2)} \& \text{idl-1} \neq \text{idl-2} \Rightarrow \text{sel(idl-1)} \neq \text{sel(idl-2)} \)

(10) \( s(i) = \)

(11) \( \text{is-intg-val}(i) \& \text{is-intg-val}(j) \& i \neq j \Rightarrow s(i) \neq s(j) \)

Ref.: \( \text{is-intg-val 9-3(5)} \)

(12) \( \text{elem}(i) = \)

Note: This function maps integer values into unique names and is defined by:

(13) \( \text{is-intg-val}(i) \Rightarrow \text{is-n\text{*elem}(i)} \)

Ref.: \( \text{is-intg-val 9-3(5)} \)

(14) \( \text{is-intg-val}(i) \& \text{is-intg-val}(j) \& i \neq j \Rightarrow \text{elem}(i) \neq \text{elem}(j) \)

Ref.: \( \text{is-intg-val 9-3(5)} \)

(15) \( \text{mk-id(cl)} = \)

Note: This function maps lists of character values into identifiers is-id and is defined by:

(16) \( \text{is-char-val-list(cl)} \Rightarrow \text{is-id\text{*mk-id(cl)}} \)

Ref.: \( \text{is-char-val 9-3(6)} \)
(17) \textit{is-char-val-list}(cl-1) \, \& \, \textit{is-char-val-list}(cl-2) \, \& \, cl-1 \neq cl-2 \Rightarrow
mk-id(cl-1) \neq mk-id(cl-2)

Ref.: \textit{is-char-val} 9-3(6)

In some applications of the function \textit{mk-id} the following shorthand notation is used for the arguments, e.g.
\textit{mk-id} (ABC) stands for \textit{mk-id}(\langle A-CHAR, B-CHAR, C-CHAR\rangle)

1.1.4 LISTS

Lists form a special class of objects. A number of functions, operations, and abbreviations is introduced for the manipulation of lists, which correspond closely to the conventional means for that purpose.

- The predicate \textit{is-list} holds for any list, it is defined as

\begin{align*}
(18) \textit{is-list(list) =} \\
= \, \textit{is-<}(list) \, \lor \, (\exists x-1, \ldots, x-n) \{ \text{list} = \mu(\langle \text{elem}(1) : x-1 \rangle, \langle \text{elem}(2) : x-2 \rangle, \ldots, \langle \text{elem}(n) : x-n \rangle) \, \& \, \neg \textit{is-O}(x-1) \, \& \, \ldots \, \& \, \neg \textit{is-O}(x-n) \}
\end{align*}

\begin{align*}
(19) \textit{is-<}(list) = \\
= \textit{is-O} \bullet \text{elem}(i, \text{list})
\end{align*}

An abbreviation for denoting elements of a list is introduced.
\textit{elem}(i, \text{list}) = \textit{elem}(i) (\text{list}) \, \text{for} \, \textit{is-list(list)}.

\begin{align*}
(20) \textit{is-list-1(list) =} \\
= \textit{is-list(list)} \, \& \, \neg \textit{is-<}(list).
\end{align*}

\textbf{Note:} This predicate excludes especially for \( n=0 \) \( \textit{is-<}(list) \)

The length of a list is defined as the largest index of an element which is not the null object.

\begin{align*}
(21) \textit{length(list) =} \\
= \textit{is-<}(list) \, \rightarrow \, 0 \\
\rightarrow \textit{is-list(list)} \, \rightarrow \, (\forall i) (\neg \textit{is-O} \bullet \text{elem}(i, \text{list}) \, \& \, \textit{is-O} \bullet \text{elem}(i + 1, \text{list})) \\
T \rightarrow \text{error}
\end{align*}

The following functions yield, when applied to a list, the head (which is the first element of a list if it exists), the last element of a list (if it exists), the tail of a list (which is the original list except the first element) and the first of a list (which is the original list except the last element).
(22) \texttt{head}(\texttt{list}) =
\begin{align*}
is-\text{list}(\texttt{list}) & \land \neg \text{-<>(\texttt{list})} \rightarrow \text{elem}(1,\texttt{list}) \\
\top & \rightarrow \text{error}
\end{align*}

(23) \texttt{tail}(\texttt{list}) =
\begin{align*}
is-\text{list}(\texttt{list}) & \land \neg \text{-<>(\texttt{list})} \rightarrow \\
(\text{length}(\texttt{list}) = 1 & \rightarrow \langle \rangle, \\
\text{length}(\texttt{list}) > 1 & \rightarrow \\
\mu_0(\langle \text{elem}(i) : \text{elem}(i + 1, \texttt{list}) \mid 1 \leq i \leq \text{length}(\texttt{list}) - 1 \rangle)
\end{align*}
\top \rightarrow \text{error}

(24) \texttt{tail-1}(\texttt{list}) =
\begin{align*}
is-\text{list}(\texttt{list}) & \land \neg \text{-<>(\texttt{list})} \rightarrow \text{tail}(\texttt{list}) \\
is-\langle\rangle(\texttt{list}) & \rightarrow \langle \rangle \\
\top & \rightarrow \text{error}
\end{align*}

(25) \texttt{first}(\texttt{list}) =
\begin{align*}
\neg \text{is-\text{list}(\texttt{list})} & \lor \text{-<>(\texttt{list})} \rightarrow \text{error} \\
\text{length}(\texttt{list}) = 1 & \rightarrow \langle \rangle, \\
\text{length}(\texttt{list}) > 1 & \rightarrow \mu_0(\langle \text{elem}(i) : \text{elem}(i, \texttt{list}) \mid 1 < i < \text{length}(\texttt{list}) - 1 \rangle)
\end{align*}

(26) \texttt{last}(\texttt{list}) =
\begin{align*}
is-\text{list}(\texttt{list}) & \land \neg \text{-<>(\texttt{list})} \rightarrow \text{elem}(\text{length}(\texttt{list}), \texttt{list}) \\
\top & \rightarrow \text{error}
\end{align*}

The concatenation of two lists is defined by:
\begin{align*}
\texttt{list-1\texttt{-list-2}} & \equiv \text{is-\text{list}(\texttt{list-1})} \land \text{is-\text{list}(\texttt{list-2})} \rightarrow \\
\mu(\texttt{list-1}; \langle \text{elem}(\text{length}(\texttt{list-1} + i) : \text{elem}(i, \texttt{list-2}) \mid 1 \leq i \leq \text{length}(\texttt{list-2}) \rangle)
\end{align*}

Note that \texttt{list-<> = <>-list = list} for is-\text{list}(\texttt{list}).

Multiple concatenation is defined by:
\begin{align*}
\texttt{CONC list-i \equiv ( n < 1 \rightarrow \langle \rangle, n = 1 \rightarrow \texttt{list-1}, \texttt{CONC list-i-list-2 \cdots list-n} \mid i = 1) \rightarrow \text{list-1-list-2 - \ldots - list-n} \\
\top & \rightarrow \text{is-}\text{list}\text{-}\text{list-1-list-2 - \ldots - list-n}
\end{align*}

As a convenient form to denote lists one may enumerate the elements within pointed brackets. The form is defined by:

1. NOTATION AND CONVENTIONS 9
<a-1,a-2, \ldots, a-n> = \mu_0(<\text{elem}(1):a-1>, <\text{elem}(2):a-2>, \ldots, <\text{elem}(n):a-n>)

for: n \geq 1, a-i \neq 0 (1 \leq i \leq n)

An alternative form is:

\[ \text{LIST}_{i=1}^n \text{ } a-i = (n<1 \rightarrow \text{ }, T \rightarrow <a-1,a-2, \ldots,a-n>) \]

1.1.5 EQUATION

= equal
\# not equal

The equality and not equality relations are used with no specific restriction to the range of arguments.

1.1.6 TRUTH VALUES, LOGICAL OPERATORS AND QUANTIFIERS

1.1.6.1 Truth values

T true
F false

1.1.6.2 Logical operators

\begin{align*}
\rightarrow & \text{ not} \quad \equiv \text{ equivalence} \\
\& & \text{ logical and} \quad \neq \text{ non equivalence} \\
\lor & \text{ logical or (vel)} \quad \Rightarrow \text{ implication}
\end{align*}

The operators have the conventional meaning except for two-place operators in cases where one of the operands is undefined. The meaning adopted is best described using conditional expressions:

\[ (P_1 \& P_2) = (\neg P_1 \rightarrow T, T \rightarrow P_2) \]
\[ (P_1 \lor P_2) = (P_1 \rightarrow T, T \rightarrow P_2) \]
\[ (P_1 \equiv P_2) = (P_1 \& P_2) \lor (\neg P_1 \& \neg P_2) \]
\[ (P_1 \neq P_2) = (P_1 \lor P_2) \]
\[ (P_1 \Rightarrow P_2) = (\neg P_1 \rightarrow T, T \rightarrow P_2) \]

The following rule for omission of parentheses holds for expressions built from the implication operator:

\[ p-1 \Rightarrow p-2 \Rightarrow \ldots \Rightarrow p-(n-1) \Rightarrow p-n = (p-1 \Rightarrow (p-2 \Rightarrow \ldots \Rightarrow (p-(n-1) \Rightarrow p-n)) \]

To ease printing the symbols "\texttt{Et}" and "\texttt{Vel}" are used for multiple conjunction and disjunction:

\[ \text{Et}_{i=1}^n \text{ } p-i = (n<1 \rightarrow \text{ }, T \rightarrow p-1 \& p-2 \& \ldots \& p-n) \]
\[ \forall x \in X \quad \exists \mathbf{p}(x) = (\forall y (\exists z \in z) (x) = F_{\mathbf{p}(x)}(z)) \]

Note, that \( \forall x \in X \quad \exists \mathbf{p}(x) \) is defined only if \( \forall m \in M \quad \mathbf{p}(x_m) \) is independent of the ordering of \( x_1, \ldots, x_n \).

\[ \forall_{i=1}^{n} \quad p \in i = (n \leq 1 \rightarrow T, T \rightarrow n-1 \lor p-2 \lor \ldots \lor p-n) \]

### 1.1.5.3 Quantifiers

- \( \exists \) existential quantifier
- \( \forall \) universal quantifier

The above symbols are used in expressions of the following form:

\( (\exists x_1, x_2, \ldots, x_n) (\mathbf{p}(x_1, x_2, \ldots, x_n)) \)

The variables \( x_1, x_2, \ldots, x_n \) are called the bound variables of the expression. The expression is true if there exists at least one \( n \)-tuple \( x_1, x_2, \ldots, x_n \) such that \( \mathbf{p}(x_1, x_2, \ldots, x_n) \) is true, otherwise the expression is false.

\( (\forall x_1, x_2, \ldots, x_n) (\mathbf{p}(x_1, x_2, \ldots, x_n)) \)

The variables \( x_1, x_2, \ldots, x_n \) are called the bound variables of the expression. The expression is true if for all possible \( n \)-tuples \( x_1, x_2, \ldots, x_n \) (in the range of the variables) \( \mathbf{p}(x_1, x_2, \ldots, x_n) \) is true, otherwise the expression is false.

It is important that the range of the bound variables in expressions of the above form always be defined. This will either be done explicitly by the expression or implicitly by using a convention that associates a range with a specific class of variable names.

As a notational convention the outermost universal quantifier can be omitted in cases where no confusion can arise.

For convenience, composite constructs containing bound variables as components may be written in place of the bound variable part of the expression, e.g.

\( (\exists x, y) (p(x, y < x, y)) \).

### 1.1.6.4 Jota-operator

The symbol is used in expressions of the following form:

\( (\exists x) (\mathbf{p}(x)) \)

The \( x \) is called the bound variable of the expression. The expression denotes the value (out of the range of \( x \)) for which \( \mathbf{p}(x) \) is true. The expression has no value if no or more than one value in the range of \( x \) has the property \( p \).

### 1.1.7 ARITHMETIC OPERATORS AND RELATIONS

Arithmetic operators and relations are defined for entities satisfying the predicate is-num-val, see chapter 9.

### 1.1.7.1 Operators

---

1. NOTATION AND CONVENTIONS 11
* prefix plus, infix plus       / division
- prefix minus, infix minus    * exponentiation
  multiplication

\[ \sum_{i=1}^{n} x^{-i} = \left( \begin{array}{c} n < 1 \rightarrow 0, \ T \rightarrow x^{-1} + x^{-2} + \ldots + x^{-n} \end{array} \right) \]

\[ \prod_{i=1}^{n} x^{-i} = \left( \begin{array}{c} n < 1 \rightarrow 1, \ T \rightarrow x^{-1} \cdot x^{-2} \cdot \ldots \cdot x^{-n} \end{array} \right) \]

1.1.7.2 Relations

<  less
\leq  less or equal
\leq  equal
\neq  not equal
\geq  greater or equal
>  greater

The relational operators are occasionally used in meta-expressions of the form:

\[ e^{-1} \ R^{-1} \ e^{-2} \ R^{-2} \ e^{-3} \]

where \( e^{-i} \) is an arithmetic expression and \( R^{-i} \) is one of the above relational operators. This form has the meaning:

\( (e^{-1} \ R^{-1} \ e^{-2}) \ & (e^{-2} \ R^{-2} \ e^{-3}) \)

A meta-expression of the form

\[ e^{-1} \ R^{-1} \ e^{-2}, e^{-2} \ R^{-2} \ e^{-3} \]

has the meaning

\( (e^{-1} \ R^{-1} \ e^{-2} \ R^{-2} \ e^{-3}) \ & (e^{-1} \ R^{-1} \ e^{-2} \ R^{-2} \ e^{-3}) \)
1.1.7.3 Arithmetic functions

In this section satisfy:

\[
x, y \quad \text{is-\text{real-val}}
\]
\[
z \quad \text{is-\text{num-val}}
\]
\[
n, m \quad \text{is-\text{intg-val}}
\]

(27) \( \text{abs}(x) = \)

\[
\begin{align*}
(x \geq 0 & \rightarrow x, \\
T & \rightarrow -x)
\end{align*}
\]

(28) \( \text{sign}(x) = \)

\[
\begin{align*}
(x \geq 0 & \rightarrow 1, \\
T & \rightarrow -1)
\end{align*}
\]

(29) \( \text{modulo}(x, y) = \)

\[
(\exists m)(\text{is-\text{intg-val}}(m) \land x = n \cdot j + r \land 0 \leq r < \text{abs}(y))
\]

for: \( y > 0 \)

Ref.: is-\text{intg-val} 9-3(5)

Note: The function gives the remainder if \( x \) is divided by \( y \).

(30) \( \text{ceil}(x) = \)

\[
(\exists n)(\text{is-\text{intg-val}}(n) \land n - 1 < x \leq n)
\]

Ref.: is-\text{intg-val} 9-3(5)

Note: The function gives the smallest integer not exceeded by \( x \).

(31) \( \text{floor}(x) = \)

\[
(\exists n)(\text{is-\text{intg-val}}(n) \land n \leq x < n + 1)
\]

Ref.: is-\text{intg-val} 9-3(5)

Note: The function gives the greatest integer not exceeding \( x \).
(32) \[ \text{trunc}(x) = \]
\[
\begin{cases} 
(x \geq 0 \rightarrow \text{floor}(x), \\
T \rightarrow \text{ceil}(x) 
\end{cases}
\]

(33) \[ \text{max}(x,y) = \]
\[
\begin{cases} 
(x \geq y \rightarrow x, \\
T \rightarrow y 
\end{cases}
\]

(34) \[ \text{min}(x,y) = \]
\[
\begin{cases} 
(x \leq y \rightarrow x, \\
T \rightarrow y 
\end{cases}
\]

(35) \[ \text{cplx}(x,y) = \]
\[ \text{real}(x) + \text{imag}(y) \]

(36) \[ \text{conjg}(z) = \]
\[ \text{real}(z) - \text{imag}(z) \]

(37) \[ \text{real}(z) = \]
\[ \text{Note: This function gives the real part of the complex number } z. \]

(38) \[ \text{imag}(z) = \]
\[ \text{Note: This function gives the imaginary part of the complex number } z. \]

(39) \[ \text{card}(\text{set}) = \]
\[ \text{for:is-set}(\text{set}) \]
\[ \text{Note: This function yields the cardinal number of the set.} \]

(40) \[ \text{max-set}(\text{n-set}) = \]
\[ (\exists x)(x \in \text{n-set} \& (\forall y)(y \in \text{n-set} \rightarrow x \geq y)) \]
\[ \text{for:is-real-val-set}(\text{n-set}) \& \neg \text{is-}\emptyset(\text{n-set}) \]
\[ \text{Ref.: is-real-val 9-3(8)} \]
30 April 1969

ABSTRACT SYNTAX AND INTERPRETATION OF PL/I

(41) \text{min-set}(n\text{-set}) =
\begin{align*}
&\{x \in n\text{-set} \land (\forall y)(y \in n\text{-set} \Rightarrow x \leq y)\} \\
\text{for:is-real-val-set}(n\text{-set}) & \land \text{not-}\{\text{n-set}\}
\end{align*}

Ref.: is-real-val 9-3(4)
1.1.8 Set Operators, Relations and Notation for Sets

1.1.8.1 Set Operators

\[ u \quad \text{union} \]
\[ n \quad \text{intersection} \]
\[- \quad \text{difference} \]
\[ n \]
\[ \bigcup_{i=1}^{n} \text{set}_i = \{ a \in i \rightarrow \{ \}, i \rightarrow \text{set}_1 \cup \text{set}_2 \cup \ldots \cup \text{set}_n \} \]

1.1.8.2 Relations

\[ \in \quad \text{is element of} \]
\[ \notin \quad \text{is not element of} \]
\[ < \quad \text{is proper subset} \]
\[ \subseteq \quad \text{is subset or equal} \]

1.1.8.3 Notation for Sets

\[ \{a, b, c, \ldots\} \quad \text{the elements } a, b, c, \ldots \text{ are the elements of the set} \]
\[ \{\} \quad \text{the empty set} \]
\[ \{e(x_1, \ldots, x_n) \mid p(x_1, \ldots, x_n)\} \quad \text{implicit definition of a set} \]

where \( e(x_1, \ldots, x_n) \) is an expression, depending on the n-tuple \( x_1, \ldots, x_n \). This notation defines the set of the values of this expression for all n-tuples \( x_1, \ldots, x_n \) for which the predicate \( p(x_1, \ldots, x_n) \) is true. If the \( x_i \) are defined outside the set notation, they retain their meaning.

1.1.9 Rules of Precedence

Parentheses may be omitted according to the following rules of precedence:
1. NOTATION AND CONVENTIONS
1.1.10 NAMING

The following conventions have been adopted for the naming of entities:

1) Names of functions are strings of small letters, digits, and hyphens.
2) Metavariables are strings of small letters, digits, and hyphens.
3) Constant objects are strings of capital letters, digits, and hyphens (with the exception of the special objects U,*,<,{}, and numerical values).
4) Names of predicates are strings of letters, digits, and hyphens, prefixed by is-.

1.2 DEFINITIONS

This chapter describes the structure and organization of the definitions presented in chapters 2 through 13 of the document. There are exceptions to the general rules described in the following, which, however, are stated at the appropriate places.

The document is divided in chapters. Each chapter or sub-chapter consists of the following:

1) a chapter number and a title,
2) optionally a summary of the contents of the chapter,
3) optionally a list of metavariables,
4) optionally a list of abbreviations,
5) a list of formulas, or a list of sub-chapters, or a list of formulas followed by a list of sub-chapters.

1.2.1 METAVARIABLES

Metavariables used in a definition are either listed under the heading "for:" following the formula, or at the beginning of the chapter containing the formula, or in one of the chapters containing this chapter, under the heading "Metavariables". The range of the metavariables is characterized in these listings by a predicate. If the range is specified for a metavariable, say \( x \), then this specification is also valid for all metavariables of the form \( x-i \), where \( i \) is a decimal digit.

The range of a metavariable is of significance in all cases where it is bound by a logical quantifier or by the implicit set notation. In all other cases the indication of the range has the character of a comment.

1.2.2 ABBREVIATIONS

Abbreviations used in a definition are defined either under the heading "where:" following the definition, or at the beginning of the chapter containing the definition, or at the beginning of one of the chapters containing this chapter, under the heading "Abbreviations".

Names used as abbreviations are syntactically characterized by subscripts, and subscripted names are only used for abbreviations.
1.2.3 FORMULAS

A formula consists of

1) A formula number (formulas are numbered sequentially within main chapters).
2) A function definition, or an instruction definition, or a predicate definition, or a logical statement.
3) Optionally a list of abbreviations under the heading "where:"
4) Optionally a list of statements on the range of metavariables used in the definition under the heading "for:"
5) Optionally a list of references under the heading "Ref:"
6) optionally a comment under the heading "Note:"

1.2.3.1 Function definitions

A function definition has the form:

\[ f(x-1,x-2,\ldots,x-n) = \]
\[ \text{cond-expr}(x-1,x-2,\ldots,x-n) \]

where \( f \) is a function name, \( x-i \) are metavariables, \( \text{cond-expr} \) is a conditional expression in the metavariables \( x-i \).

The special case

\[ f(x-1,x-2,\ldots,x-n) = \]
\[ T \rightarrow e(x-1,x-2,\ldots,x-n) \]

may be written as

\[ f(x-1,x-2,\ldots,x-n) = \]
\[ e(x-1,x-2,\ldots,x-n) \]

1.2.3.2 Instruction definitions

Instruction definitions are described in detail in section 1.3.3.

1.2.3.3 Predicate definitions

Special classes of objects will be defined by predicates applicable to objects. The members of a class of objects defined by a predicate are precisely those objects which satisfy the predicate.

For the predicate definitions the following notations are used. Thereby the righthand terms of these equations are not only used to define the lefthand predicates; they are themselves predicates and are used in contexts of predicates. In these notation definitions \( n \) may be any non-negative integer (generally including \( 0 \)), \( x-i \), \( y-i \) and \( z-i \) are arbitrary objects.

As a notational convention all predicates have a prefix "is-" and all explicitly mentioned selectors have a prefix "s-".

1. NOTATION AND CONVENTIONS 19
Abstract Syntax and Interpretation of PL/I

30 April 1969

a) is-pred = is-pred-1 v ... v is-pred-n

means that is-pred is defined by:

is-pred(x) =

is-pred-1(x) v ... v is-pred-n(x)

b) is-pred = (<s-sel-1:is-pred-1>,...,<s-sel-n:is-pred-n>)

means that is-pred is defined by:

is-pred(x) =

(\exists x_1, ..., x_n)( \bigwedge_{i=1}^{n} is-pred_i(x_i) & x = <x_1, ..., x_n>)

Note: This includes especially (for n = 0) is-pred (<>).

c) is-pred = is-OBJ

where OBJ is the name of a constant elementary object or Φ, means that is-pred is defined by:

is-pred(x) =

x = OBJ

d) is-pred = is-pred_k-list

means that is-pred is defined by:

is-pred(x) =

(\exists x_1, ..., x_n)( \bigwedge_{i=1}^{n} is-pred_k(x_i) & x = <x_1, ..., x_n>)

Note: This includes especially (for n = 0) is-pred (<>).

e) is-pred = is-pred_k-set

means that is-pred is defined by:

is-pred(x) =

(\exists x_1, ..., x_n)( \bigwedge_{i=1}^{n} is-pred_k(x_i) & x = {x_1, ..., x_n})

Note: This includes especially (for n = 0) is-pred ({}).

f) is-pred = (\langle f(y):is-pred-2 > | | is-pred-1(y)\rangle)

means that is-pred is defined by:

is-pred(x) =

(\exists y, z_1, ..., y-n, z-n)( \bigwedge_{i=1}^{n} (is-pred-1(y_i) & is-pred-2(z_i)) & x = \mu_0(\langle f(y-1):z-1>,...,f(y-n):z-n\rangle)

Note: This includes especially (for n = 0) is-pred (Ω).

g) is-pred = (<sel-fct(1):is-pred_0>,...)

20  1. Notation and Conventions
where sel-fct is one of the selector functions elem, and mapping integer values into selectors, means that the predicate is-pred is defined by:

\[
\text{is-pred}(x) = (\exists m, y-1, \ldots, y-m \text{ s.t. } \bigwedge_{i=1}^{m} \text{is-pred}_i(y-i) \text{ a}
\]

\[
x = \nu_0(<\text{sel-fct}(1):y-1>, \ldots, <\text{sel-fct}(m):y-m>)
\]

**General notes:**

1) The definition schemes a), b), c) are the main tools for the structural description of object classes, especially the class of programs itself: An object class may be defined by case distinction into different object classes (scheme a); each of these classes may be defined by giving the selectors of its objects and the object classes of the corresponding components (scheme b); these two kinds of definitions are iterated until one finally comes down to classes consisting of only one elementary object (scheme c).

The additional definition schemes d), f), and g) are used in those cases where a class of objects is to be defined, which have an indefinite number of components of equal structure.

The additional definition scheme e) is used in those cases where a class of objects is to be defined, which are unstructured sets of objects of equal structure.

2) The notation d) is "nearly" a special case of g):

\[
\text{is-pred-list}(x) = \text{is-<>}(x) \lor <\text{elem}(1):\text{is-pred}>, \ldots
\]

3) The most frequent use of notation f) is that the function f is the identity and is-pred-1 is a predicate characterizing a class of objects which in the same time are used as selectors: is-id or (not in the abstract syntax of text) is-n.

The following additional convention has been used. Composite predicates formed from other predicates by using the operators \& and \lor may be used directly as parts of expressions with the meaning

\[
(p-1 \& p-2)(x) = p-1(x) \& p-2(x)
\]

\[
(p-1 \lor p-2)(x) = p-1(x) \lor p-2(x)
\]

The composite predicates have to be enclosed in parentheses, as has been done in the above two equations.

1.2.3.4 Logical statements

Logical statements presented as formulas are formed according to the rules for meta-expressions. Such statements are used to express asserted properties of partially undefined functions, i.e., to state constraints for implementation defined functions.

1.2.3.5 References

All function, instruction and object names used in a formula are referenced under the heading "Ref.:", where a reference specifies the page number and the formula number under which the referenced function or instruction is defined.

There are the following exceptions to this rule:

1) All names defined in chapter 1 are not referenced.
2) All names defined in chapter 2 (Abstract Syntax) are not referenced.

3) Names of elementary objects are not referenced.

4) All names defined in a sub-chapter are not referenced throughout this sub-chapter.

1.3 THE CONTROL OF THE PL/I MACHINE

1.3.1 INFORMAL INTRODUCTION

The control part can be visualized as a finite tree where each node of the tree is associated with an instruction. Such a tree is called a control tree. The sequence in which the instructions are to be executed is partially given by the convention that only the instructions associated with the terminal nodes of the control tree are candidates for being executed next. The successor state of a given state of the PL/I machine is determined by choosing one of the terminal nodes of the control tree and executing the associated instruction.

An instruction is composed of an instruction name and optionally a list of arguments. The notation used for representing instructions is:

```
instr (arg-1, arg-2, ..., arg-n)
```

Instruction names are underlined words and the arguments are objects.

The execution of an instruction may produce an intermediate result to be used subsequently by other instructions. The PL/I machine uses a mechanism which inserts intermediate results directly into the appropriate argument-positions of those instructions in the control tree which are to make use of them.

For this purpose a so-called return information is associated with each node of a control tree which specifies the argument position into which an intermediate result produced by the instruction is to be inserted.

There is no logical restriction as to the types of state transitions which can be defined for an instruction. In particular, the execution of an instruction may modify the control part.

The definition of PL/I uses two types of effects which the execution of an instruction might have. The first type of effect, called basic transformation, is to delete the instruction being executed from the control tree, to substitute a value specified by the definition of the instruction into argument places of instructions in predecessor nodes of the control tree, and to make some changes in the state, though, in general, no further changes in the control part. The second type of effect, called self-replacing, is such that the instruction being executed replaces itself in the control tree by another control tree, a process which is very similar to a macro expansion and serves the same purpose. Whether the effect of an instruction is a basic transformation or self-replacing, in general, depends on the state in which the instruction is executed.

1.3.2 THE ABSTRACT SYNTAX OF CONTROL-TREES AND THE CONTROL PART

A specific class of objects has been chosen to represent control-trees. This specific choice is convenient but is by no means a necessary one.

In particular, control-trees are defined in the sequel more precisely as a class of objects satisfying the predicate is-ct. First, some new elementary objects and selectors are introduced:

1) is-instr-name defines an infinite class of instruction names.

2) succn is an infinite class of simple selectors which are used to select the immediate sub-trees of a control-tree.
3) The special selectors s-in, s-al, s-ri, s-comp, s-ap are used to select the various components of the instruction and the return information associated with each node of a control-tree. It is assumed that the above selectors are not contained in the set SUCCo.

Key to abbreviations:
in ...... instruction name  an ...... argument position
al ...... argument list  np ...... natural number pair
ri ...... return information
comp ... composite selector

The class of control-trees is defined by:

\[(\text{is-ct} = (\text{succ:is-ct} \cup \text{SUCCo}), \text{is-in:is-instr-name}, \text{is-al:is-ob} \cup \text{is-intg-val}(i)), \text{is-ri:is-selector}, \text{is-ap:is-np-set}))\]

Ref.: is-intg-val 9-3(5)

\[(\text{is-np} = (\text{elem}(1):is-intg-val), \text{elem}(2):is-intg-val))\]

Ref.: is-intg-val 9-3(5)

The control-part of the state of the PL/I machine satisfies the predicate is-c.

\[(\text{is-c} = \text{is-ct} \cup \text{is-np}))\]

The nodes of a control-part c are defined as the set of composite selectors:

\[(\text{node}(c) = \{s \mid s \in \text{SUCC} \& s(c) \neq 0\})\]

Note: SUCC is the set of all composite selectors which can be constructed from the set SUCCo and the identity selector I.

Some node s⁻¹ of a given control-tree, s⁻¹ \in \text{node}(ct), is a predecessor node with respect to some other node, s⁻² \in \text{node}(ct), if for some n > 1 it is the case that s⁻¹ = pred(n, s⁻²)

The predecessor function is defined by the equations:

1. NOTATION AND CONVENTIONS 23
(46) \( \text{pred}\_1(\sigma) = (i\sigma-1) \quad \exists (s \cdot \sigma-1 = s) \)

(47) \( \text{pred}(i,\sigma) = (i = 0 \rightarrow \sigma, T \rightarrow \text{pred}(i - 1, \text{pred}\_1(\sigma))) \)

where \( s \) is a simple selector and \( \sigma, \sigma-1 \) are composite selectors.

The instruction associated with a node \( \sigma \) is the instruction whose name is \( \text{s-in} \cdot \sigma(\text{ct}) \) and whose argument list is \( \text{s-al} \cdot \sigma(\text{ct}) \). The associated return-information is \( \text{s-ri} \cdot \sigma(\text{ct}) \).

The return-information has two components selected by \( \text{s-ap} \) and \( \text{s-comp} \), which are a set of pairs of natural numbers, and a composite selector, respectively.

Each pair of integers \( <i,j> \in \text{s-ap} \cdot \text{s-ri} \cdot \sigma(\text{ct}) \) is interpreted to define a certain argument position, more specifically the \( j \)-th argument position of the instruction attached to the \( i \)-th predecessor of \( \sigma \), i.e. the argument position \( \text{elem}(j) \cdot \text{s-al} \cdot \text{pred}(i, \sigma) \). The second component of the return-information is a composite selector, \( \sigma-1 \) say. If the instruction associated with \( \sigma \) is executed and produces an intermediate result, then this result is inserted as the \( \sigma-1 \) component of all argument positions specified by the \( \text{s-ap} \) part of the return-information as described above.

### 1.3.3 The Functions Term-node, Compute and Instruction Definitions

The last section has defined control-trees as a class of objects, \( \text{is-ct} \).

For each control-part, say \( c \), a set of composite selectors \( \text{node}(c) \) was identified with the nodes of the control-tree. The terminal nodes, \( \text{term-node}(c) \), of a control-part \( c \) are all its nodes which have no successor nodes.

(48) \( \text{term-node}(c) = \{ \sigma \mid \sigma \in \text{node}(c) \land \neg \exists (\sigma-1) (\sigma-1 \in \text{node}(c) \land \text{pred}\_1(\sigma-1) = \sigma) \} \)

Note: If the control part is \( 0 \) the application of the functions \( \text{node} \) or \( \text{term-node} \) yields the empty set.

The application of the function \( \text{compute} \) to a state \( \xi \) and the selector \( \sigma \) identifying the instruction to be executed gives the successor state of \( \xi \). It is defined in terms of the function corresponding to the instruction name found in the node \( \sigma \) of the control part of \( \xi \). Note that any state whose control part is \( 0 \) is an end-state since the set of its successor states is empty.

(49) \( \text{compute}(\xi, \sigma) = f\text{-in}(\text{elem}(1, \alpha), \text{elem}(2, \alpha), \ldots, \text{elem}(n\text{-in}, \alpha), \delta(\xi; \sigma), \sigma, \text{ri}) \)

where: \( \text{in} = \text{s-in} \cdot \sigma(\xi) \)
\( \alpha = \text{s-al} \cdot \sigma(\xi) \)
\( \text{ri} = \text{s-ri} \cdot \sigma(\xi) \)

The execution of an instruction is defined by the definition of a particular function \( f\text{-in} \) for each instruction name \( \text{in} \), which is used. There is an instruction definition which specifies the function \( f\text{-in} \) with \( (n\text{-in})+3 \) arguments, where \( n\text{-in} \) is characteristic for the specific instruction name. The ranges of the...
first \( n\)-in arguments are also specific for the instruction name, the ranges of the last three arguments are states, composite selectors out of \text{SUCC} and return informations, respectively.

For convenience, some special notational conventions, namely \textit{instruction-definitions}, have been adopted for the specification of the functions \( f\text{-in} \).

The format for instruction-definitions is:

\[
\text{in}(x_1, x_2, \ldots, x_n) =
\]

\[
\text{prop-1} \rightarrow \text{group-1}
\]

\[
\text{prop-2} \rightarrow \text{group-2}
\]

\[
\cdot
\]

\[
\cdot
\]

\[
\text{prop-m} \rightarrow \text{group-m}
\]

for: \( m \geq 1 \)

where: \( x_i \), \( 1 \leq i \leq n \) are metavariables

\[
\text{prop-1}, \ 1 \leq i \leq m \text{ are propositional meta-expressions which may depend on } x_i, \ 1 \leq i \leq n, \text{ and } \xi.
\]

\[
\text{group-1}, \ 1 \leq i \leq m \text{ are groups. A group is a meta-expression which has a special format and meaning to be described below and denotes a state. Each group may depend on the parameters } x_i \text{ and the metavariables } \xi, \sigma, \text{ and } ri.
\]

The function \( f\text{-in} \) defined by the above format is simply:

\[
(50) \ f\text{-in}(x_1, x_2, \ldots, x_n, \xi, \sigma, ri) =
\]

\[
(\text{prop-1} \rightarrow \text{group-1},
\]

\[
\text{prop-2} \rightarrow \text{group-2},
\]

\[
\cdot
\]

\[
\cdot
\]

\[
\cdot
\]

\[
\text{prop-m} \rightarrow \text{group-m})
\]

A group is either a basic group or self-replacing, corresponding to the two types of state transitions mentioned in 1.3.1.

The format of a basic group is:
PASS : m-expr-0
s-sc-1 : m-expr-1
s-sc-2 : m-expr-2
...
where: s-sc-i are simple selectors of immediate components of the state (e.g. s-e, s-d, s-c, ...) and m-expr-i are meta-expressions depending on the parameters of the instruction definition in which the group occurs, and on \( \xi \).

The intuitive meaning of an instruction defined by a basic group is the following. Upon execution the instruction produces an intermediate result the object denoted by m-expr-0. The intermediate result is inserted in predecessor instructions according to the return information. The immediate components of the state indicated by s-sc-i are replaced by the objects denoted by the meta-expressions m-expr-i, for \( 1 \leq i \leq r \).

Expressed formally, the meaning of a basic group is:

\[
\mu(\xi; \langle s\text{-comp}(ri)\rangle \ast \text{elem}(j) \ast s\text{-al}\text{pred}(i, \xi) : m\text{-expr}-0 > | \langle i, j \rangle \in s\text{-ar}(ri)));
\langle s\text{-sc}-1 : m\text{-expr}-1 >, \langle s\text{-sc}-2 : m\text{-expr}-2 >, ..., \langle s\text{-sc}-r : m\text{-expr}-r > )
\]

Remembering the use of f-in in the definition of the function compute one sees that \( \xi \) is the node with which the instruction being executed was associated and ri is the return information of that node. The state \( \xi \) is the state before execution of the instruction in which the \( s \)-component of the control has been deleted.

A self replacing instruction is a meta-expression m-expr, which denotes a control-tree.

The intuitive meaning of an instruction defined by a self replacing alternative is that the instruction upon execution is replaced by the control-tree denoted by the self replacing instruction, i.e. by m-expr.

Expressed formally the meaning of a self-replacing alternative is:

\[
\mu(\xi; \langle s\text{-exor}; s\text{-ri}; ri \rangle) >
\]

Note that the return information of the node \( \xi \) where the replacement takes place is not destroyed. Section 1.3.4 will introduce a special notation for meta-expressions denoting control-trees.

Two minor abbreviations are introduced:
1) An instruction definition of the form:

\[ \text{in}(f-1, f-2, \ldots, f-n) = \]

\[ T \rightarrow \text{group} \]

may be abbreviated to:

\[ \text{in}(f-1, f-2, \ldots, f-n) = \]

\[ \text{group} \]

2) A basic group of the form:

\[ \text{PASS} : \emptyset \]

\[ s-sc-1 : \text{m-expr-1} \]

\[ s-sc-2 : \text{m-expr-2} \]

\[ \ldots \]

\[ s-sc-r : \text{m-expr-r} \]

may in the case of \( r \geq 1 \) be abbreviated to:

\[ s-sc-1 : \text{m-expr-1} \]

\[ s-sc-2 : \text{m-expr-2} \]

\[ \ldots \]

\[ s-sc-r : \text{m-expr-r} \]

1.3.4 A SPECIAL NOTATION FOR META-EXPRESSIONS DENOTING CONTROL-TREES

1.3.4.1 Informal introduction

Control-trees are objects of type \( \text{is-ct} \). Meta-expressions to denote objects of that class can be built by the means which have been introduced already. However, the bulk of the definition of PL/I consists of instruction definitions, which in most cases involve the specification of a control-tree. Therefore it is advisable to introduce some notational conventions which facilitate the specification of control-trees.

The choice to define the return-information attached to each node of a control-tree as a set of pairs of natural numbers and one composite selector (see the definition of \( \text{is-ct} \) of section 1.3.2) is convenient for the formal treatment of the control of the PL/I machine. It is, however, less convenient for the specification of particular control-trees. The notational device which is used instead applies dummy names which link the source and the destinations for the substitution mechanism of intermediate results. In particular, a subtree which eventually gives rise to an intermediate result is prefixed by a suitable dummy name; the same dummy name is written in all argument positions to which the intermediate result is to be passed.
Furthermore, control-trees are represented in a linear notation by representing the instruction attached to the top node followed by a semicolon followed by a representation of the set of immediate successor trees.

1.3.4.2 Format and meaning of meta-expressions denoting control-trees

In the sequel the formats for meta-expressions denoting control-trees and their intuitive meaning is given.

A meta-expression for a control-tree, ct-expr, has one of the following two forms:

(1) instr-expr
(2) instr-expr; succ-set

In the first alternative the control-tree consists only of one single instruction denoted by instr-expr. The second alternative denotes a control-tree whose top node is associated with the instruction denoted by instr-expr and whose immediate sub-trees are the elements of the set denoted by succ-set.

Meta-expressions, instr-expr, denoting instructions may be of the following forms:

(1) in
(2) in(s-expr-1, s-expr-2, ..., s-expr-n)

where the name of the instruction, in, is a string of small letters, digits and hyphens which is underlined. The expressions s-expr-1, s-expr-2, ..., s-expr-n are either meta-expressions or dummy names.

The set of immediate sub-trees is denoted by succ-set, which is a set-expression in usual set notation, i.e. implicit or explicit, using successor expressions, succ-expr.

Meta-expressions, succ-expr, denoting sub-trees are either meta-expressions for control-trees or prefixed meta-expressions for control-trees and may assume the following forms:

(1) ct-expr
(2) dum:ct-expr
(3) m-expr{dum}:ct-expr

where dum is a dummy name and m-expr (occurring in the third alternative) is a meta-expression denoting a composite selector. The initial parts of form (2) and (3), "dum:" and "m-expr{dum}:", are called the prefix of the successor expression.

A dummy name is defined by its occurrence in the prefix of a successor expression. Dummy names are used in argument positions of predecessor instructions. The significance of these dummy names is to define that an intermediate value produced in the node where the dummy name occurs in the prefix is to be inserted into all the argument places of the predecessor instructions where the dummy name has been used. The third alternative of the successor expression indicates that the value produced in the node is to be inserted as the s-component of the argument places indicated by the dummy name, where s is the composite selector denoted by the meta-expression m-expr.

Meta-expressions denoting control-trees are not only used to define self replacing instructions but also in basic groups following s-c's, with the effect that the entire control component is replaced by the denoted control-tree.
Finally, a minor abbreviation is defined. Control-tree-expressions of the special form:

\[ \text{instr-expr;\{succ-expr-1,succ-expr-2,...,succ-expr-n\}} \]

may alternatively be written, using indentation, as follows:

\[ \text{instr-expr; } \]
\[ \text{succ-expr-1, } \]
\[ \text{succ-expr-2, } \]
\[ \text{. } \]
\[ \text{. } \]
\[ \text{succ-expr-n} \]

A similar convention holds for successor expressions themselves.

1.3.4.3 Formalization of the meaning of control-tree expressions

The plan is to define the meaning of the forms given in the previous section in terms of meta-expressions using only means which are already introduced.

The first step to be taken is to eliminate the dummy names from the given control-tree expression according to the following two rules.

1) A set of integer pairs is constructed for each occurrence of a dummy name in the prefix of a successor expression. A pair \(<i,j>\) is element of this set if and only if there is a use of the dummy name as the \(j\)-th argument of an instruction being \(i\) levels higher up, i.e. if the difference between the number of left braces "(" and the number of right braces ")" which occur between the use of the dummy and its occurrence as a prefix is \(i\).

2) All uses of dummy names are replaced by \(0\). The scope of dummy names is, as a consequence of these rules always the entire control-tree expression in which they occur.

After this first step has been taken one may use the following table of replacements to determine the meaning of a given ct-expr. For the choice of selectors for the successor trees it is assumed that there is a function \(\text{sel}(x,\text{set})\) which yields for any control-tree \(x\) \(\in\) set a selector \(\text{succ} \in \text{SUCC}_o\), such that for any \(y\), \(y\#x\), and \(y \in \text{set}\), the function yields a different selector, i.e. \(\text{sel}(x,\text{set}) \neq \text{sel}(y, \text{set})\).
### 1.4. The Treatment of Program Errors

A successor state of a certain state is determined by the definition of one of the instructions which are candidates for execution in that state (cf. 3.1), i.e. a successor state is undefined if the relevant instruction is undefined for the state. For indicating that this situation occurs the instruction

\[(51) \text{ERROR} = \ldots \]

and the function

\[(52) \text{error} = \ldots \]

are used.

An instruction is undefined for a given state if:

1) When inspecting the propositions \( p \mid i \) in the instruction definition from top to bottom (cf. 1.3.3), an undefined proposition is encountered before arriving at the proposition which is true, in the given instance of the arguments of the instruction and the state.

2) The group in the instruction definition, selected by the propositions in the given instance of the arguments of the instruction and the state, specifies the instruction \text{ERROR}.

3) One of the meta-expressions specified in the selected group is undefined in the given instance.
A meta-expression is undefined in a given instance if:

1) It is a function undefined in that instance.
2) One of its sub-expressions required for the evaluation of the meta-expression is undefined.
3) It has the form \((\lambda x)(e(x))\), if \(e(x)\) denotes \(T\) for no \(x\) or for more than one \(x\).

A function is undefined in a given instance if:

1) It denotes the function error in that instance.
2) One of its arguments is undefined.
3) It is defined by a meta-expression which is undefined.
4) The function is \(\emptyset\) (applied to any argument).

No other cases of undefinedness occur in the definition.

It is important to distinguish the cases of undefined successor states from the cases of successor states defined by implementation defined functions or instructions. In the latter cases an implementation defined function or instruction is always explicitly mentioned. Range and domain or constraints for theses functions and instructions are specified in the definition as far as they are known.

1.5 Generally Used Simple Instructions

(53) \texttt{null} = 

The only effect of the null-instruction is that it is deleted from the control on execution and that \(0\) is returned.

(54) \texttt{pass(e)} =

\begin{verbatim}
pass:e 
\end{verbatim}

For any meta-expression \(e\).

(55) \texttt{pass-f(e-1, \ldots, e-n)} =

\begin{verbatim}
pass:f(e-1, \ldots, e-n) 
\end{verbatim}

A special convention has been adopted for the meaning of instructions of the form \texttt{pass-f}, where \(f\) stands for any function name. An instruction of this form returns the value of the named function applied to the arguments of the instruction.

(56) \texttt{pk-list(x, list)} =

\begin{verbatim}
pass:<x>"list 
\end{verbatim}
(57) \[ \text{mk-list-1}(\text{list}, x) = \]
\[ \text{PASS: list-<x>} \]
2. ABSTRACT SYNTAX OF PROGRAM

The abstract syntax describes the syntactical structure of abstract PL/I programs to be interpreted by the FL/I machine.

The abstract PL/I programs belong to a class of objects characterized by the predicate

\[ \text{is-program} \]

The contents of the abstract syntax is the definition of this predicate, given by a set of predicate definitions which, applied iteratively, describe the composition of a program by its elementary components.

The elementary components of a program belong to the following classes of elementary objects:

(a) A finite class of constant elementary objects, usually denoted by names written in capital letters (e.g. STATIC, GOTO, MINUS), including the two special elementary objects * and <> (empty list).

(b) The infinite class of FL/I identifiers, characterized by the predicate \text{is-id} (cf. chapter 1).

(c) The infinite class of numerical values, characterized by the predicate \text{is-num-val}, including the class of integer values characterized by the predicate \text{is-intg-val}, and the finite classes of values of characters and bits, characterized by the predicates \text{is-char-val} and \text{is-bit-val} (cf chapter 9).

(d) The class of sets of file attributes including the empty set \{\}. The file attributes are the fifteen elementary objects satisfying the predicate \text{is-file-atrr}.

During the interpretation of a PL/I program parts of it are modified by insertion of unique names into certain positions (by the prepass, 4.3, and the unique qualification of names, 6.1.1). In order to have predicates describing also the so modified FL/I text, the predicates of the abstract syntax define wider classes of objects, in such a way that they contain not only original PL/I text but also the corresponding modified text. Within the wider class of programs, characterized by the predicate \text{is-program}, the original PL/I programs are uniquely characterized by the property that none of its elementary components is a unique name.

1. \text{is-proper-program}(t) =
   \text{is-program}(t) \land \neg(\exists x)(\text{is-n\cdot x}(t))

   These locations in a program into which a unique name may be inserted may be recognized in the abstract syntax by the occurrence of the following predicate:

2. \text{is-cpt-n} =
   \text{is-n} \lor \text{is-0}

   A reader who is interested only in the structure of a proper unmodified PL/I program has to take only the second alternative is-0 in all occurrences of this predicate.
2.1 PROGRAM DECLARATION

(3) \texttt{is-program =}
\begin{align*}
&\langle \texttt{s-decl-part:is-decl-part} , \\
&\texttt{s-body-part:is-body-part} \rangle
\end{align*}

(4) \texttt{is-body-part =}
\begin{align*}
&\langle \texttt{id:is-body} \mid \texttt{is-id(id)} \rangle
\end{align*}

(5) \texttt{is-body =}
\begin{align*}
&\langle \texttt{s-entry-part:is-entry-part} , \\
&\texttt{s-decl-part:is-decl-part} , \\
&\texttt{s-body-part:is-body-part} , \\
&\texttt{s-cond-part:is-cond-part} , \\
&\texttt{s-st-list:is-st-list} , \\
&\texttt{s-record:is-cpt} , \\
&\texttt{recursive:is-opt} \rangle
\end{align*}

(6) \texttt{is-entry-part =}
\begin{align*}
&\langle \texttt{id:is-entry-point} \mid \texttt{is-id(id)} \rangle
\end{align*}

(7) \texttt{is-entry-point =}
\begin{align*}
&\langle \texttt{s-st:is-index-list} , \\
&\texttt{s-rare-list:is-id-ref-list} , \\
&\texttt{ret-type:is-descr-scalar} \rangle
\end{align*}

(8) \texttt{is-id-ref =}
\begin{align*}
&\langle \texttt{id:is-id} , \\
&\texttt{nil:is-cpt-n} \rangle
\end{align*}

(9) \texttt{is-cond-part =}
\begin{align*}
&\langle \texttt{ccns:prefix-cond-list} , \\
&\texttt{nc:prefix-cond-list} \rangle
\end{align*}

(10) \texttt{is-cpt =}
\begin{align*}
&\texttt{is-s} \lor \texttt{is-n}
\end{align*}

2.2 DECLARATIONS

2.2.1 GENERAL

(11) \texttt{is-decl-part =}
\begin{align*}
&\langle \texttt{id:is-decl} \mid \texttt{is-id(id)} \rangle
\end{align*}
(12) is-decl =
  is-prop-var v is-defined v is-based v is-entry-const v is-file-const v
  is-label-const v is-format-const v is-generic v is-BUILTIN v is-COND v is-attr

(13) is-prop-var =
  (<s-scope:is-INT v is-EXT v is-FAIL>,
   <s-stg-cl:is-AUTO v is-STATIC v is-CTL v is-O>,
   <s-aggr:is-prop-aggr>,
   <s-connected:is-opt>,
   <s-den:is-cpt-n>)

(14) is-defined =
  (<s-base:is-ref>,
   <s-aggr:is-prop-aggr>,
   <s-pcs:is-cpt-expr>)

(15) is-based =
  (<s-ptr:is-opt-ref>,
   <s-aggr:is-prop-aggr>)

(16) is-entry-const =
  (<s-scope:is-INT v is-EXT>,
   <s-descr-list:is-descr-list v is->>,
   <s-ret-type:is-descr-scalar>,
   <s-bdy:is-id v is-O>,
   <s-reducible:is-opt>,
   <s-den:is-cpt-n>)

(17) is-descr =
  is-* v
  (<s-stg-cl:is-CTL v is-O>,
   <s-aggr:is-descr-aggr>,
   <s-connected:is-opt>)

(18) is-file-ccnst =
  (<s-scope:is-INT v is-EXT>,
   <s-file-attr:is-file-attr-set>,
   <s-env-attr:is-env-attr>,
   <s-den:is-cpt-n>)

(19) is-file-attr =
  is-CTI v is-ESI v is-REC v is-INE v is-OUT v is-PEE v is-SEQ v is-TRA v
  is-DIR v is-BUF v is-ORB v is-PRT v is-BAC v is-EXC v is-KEY

(20) is-env-attr =

Note: This predicate is implementation defined.
(21) \textit{is-label-const} = \\
\quad \textit{is-index-list-1} \\
(22) \textit{is-index} = \\
\quad \textit{is-intg-val} \lor \textit{is-T} \lor \textit{is-F} \\
\textit{Ref.}: \textit{is-intg-val} 9-3(5) \\
(23) \textit{is-format-const} = \\
\quad (\langle \textit{s-cond-part:is-cond-part}, \langle \textit{s-format-list:is-format-list-1}, \langle \textit{s-ident:is-id}> \\
(24) \textit{is-generic} = \\
\quad \textit{is-generic-member-list-1} \\
(25) \textit{is-generic-member} = \\
\quad (\langle \textit{s-entry:is-ref}, \langle \textit{s-descr-list:is-generic-descr-list} \\
(26) \textit{is-generic-descr} = \\
\quad (\langle \textit{s-stg-cl:is-CTL} \lor \textit{is-Q}, \langle \textit{s-aggr:is-generic-aggr} \\
(27) \textit{is-attn} = \\
\quad (\langle \textit{s-env-attr:is-env-attr}, \langle \textit{s-den:is-opt-n} \\

2.2.2 \textbf{AGGREGATE ATTRIBUTES} \\

2.2.2.1 \textbf{AGGREGATES IN VARIABLE DECLARATIONS} \\

(28) \textit{is-prop-aggr} = \\
\quad \textit{is-prop-array} \lor \textit{is-prop-struct} \lor \textit{is-prop-scalar} \\
(29) \textit{is-prop-array} = \\
\quad (\langle \textit{s-lbd:is-expr} \lor \textit{is-} \lor \textit{is-refer}, \langle \textit{s-ubd:is-expr} \lor \textit{is-} \lor \textit{is-refer}, \langle \textit{s-elem:is-prop-aggr} \\
(30) \textit{is-refer} = \\
\quad (\langle \textit{s-expr:is-expr}, \langle \textit{s-refer:is-id-list} \\

4. \textbf{2. ABSTRACT SYNTAX OF PROGRAM}
(31) \( \text{is-prop-struct} = \) 
\( \text{is-prop-succ-list} \)

(32) \( \text{is-prop-succ} = \) 
\( \text{(<is-qual-is-id>,} \) 
\( \text{<is-aggr-is-prop-aggr>)} \)

(33) \( \text{is-prop-scalar} = \) 
\( \text{(<s-da-is-prop-da>,} \) 
\( \text{<s-density-is-AL \& is-UNAL>,} \) 
\( \text{<s-init-is-init>} \) \)

(34) \( \text{is-prop-da} = \) 
\( \text{is-prop-arithmetic \& is-prop-string \& is-pic \& is-prop-area \& is-entry-da \&} \) 
\( \text{is-label-da \& is-offset \& is-PTR \& is-FILL \& is-TASK \& is-EVENT} \)

(35) \( \text{is-prop-arithmetic} = \) 
\( \text{(<s-node-is-REAL \& is-CPLX>,} \) 
\( \text{<s-base-is-DEC \& is-BIN>,} \) 
\( \text{<s-scale-is-FLT \& is-FIX>,} \) 
\( \text{<s-prec-is-int-val>,} \) 
\( \text{<s-scale-f-is-int-val \& is-G>} \) \)

Ref.: \( \text{is-int-val} \ 9-3(5) \)

(36) \( \text{is-prop-string} = \) 
\( \text{(<s-base-is-CHAR \& is-BIT>,} \) 
\( \text{<s-length-is-exp \& is-\* \& is-ref>,} \) 
\( \text{<s-varying-is-VAR \& is-G>} \) \)

(37) \( \text{is-prop-area} = \) 
\( \text{(<s-size-is-exp \& is-\* \& is-ref>)} \)

(38) \( \text{is-entry-da} = \) 
\( \text{(<s-descr-list-is-descr-list \& is-\*>,} \) 
\( \text{<s-ret-type-is-descr-scalar>,} \) 
\( \text{<s-reducible-is-opt>} \) \)

(39) \( \text{is-label-da} = \) 
\( \text{is-LABEL \v} \) 
\( \text{(<s-label-list-is-id-ref-list>)} \)

(40) \( \text{is-offset} = \) 
\( \text{is-OFFSET \v} \) 
\( \text{(<s-area-is-ref>)} \)
ABSTRACT SYNTAX AND INTERPRETATION OF PL/I

2.2.2.2 Aggregates in Parameter Descriptors

(41) \textit{is-init} = \\
\textit{is-init-elem-list} \lor \textit{is-call-st} \lor \textit{is-spec-init-elem-list}

(42) \textit{is-init-elem} = \\
\textit{is-init-iter} \lor \textit{is-expr} \lor \textit{is-}=

(43) \textit{is-init-iter} = \\
\langle \textit{s-rep:is-expr}, \textit{s-init:is-init-elem-list-1} \rangle

(44) \textit{is-spec-init-elem} = \\
\langle \textit{s-el:is-intg-val-list-1}, \textit{s-id:is-id}, \textit{s-n:is-arg} \rangle

Ref. : \textit{is-intg-val 9-3(5)}

2.2.2.2 Aggregates in Parameter Descriptors

(45) \textit{is-descr-aggr} = \\
\textit{is-descr-array} \lor \textit{is-descr-struct} \lor \textit{is-descr-scalar}

(46) \textit{is-descr-array} = \\
\langle \textit{s-lbd:is-intg-val} \lor \textit{is-}>, \textit{s-ubd:is-intg-val} \lor \textit{is-}>, \textit{s-elem:is-descr-aggr} \rangle

Ref. : \textit{is-intg-val 9-3(5)}

(47) \textit{is-descr-struct} = \\
\textit{is-descr-succ-list-1}

(48) \textit{is-descr-succ} = \\
\langle \textit{s-aggr:is-descr-aggr} \rangle

(49) \textit{is-descr-scalar} = \\
\langle \textit{s-da:is-descr-da}, \textit{s-dens:is-AL} \lor \textit{is-UNAL} \rangle

(50) \textit{is-descr-da} = \\
\textit{is-prop-ariths} \lor \textit{is-descr-string} \lor \textit{is-pic} \lor \textit{is-descr-area} \lor \textit{is-entry-da} \lor \textit{is-LABEL} \lor \textit{is-offset} \lor \textit{is-FTR} \lor \textit{is-FILE} \lor \textit{is-TASK} \lor \textit{is-EVENT}

6 2. ABSTRACT SYNTAX OF PROGRAM
30 April 1969

ABSTRACT SYNTAX AND INTERPRETATION OF PL/I

(51) \( \text{is-descr-string} = \)
\[
\langle s\text{-base:is-CHAR v is-BIT},
\langle s\text{-length:is-intg-val v is-},
\langle s\text{-varying:is-VAR v is-O} \rangle
\rangle
\]
Ref.: is-intg-val 9-3(5)

(52) \( \text{is-descr-area} = \)
\[
\langle s\text{-size:is-intg-val v is-} \rangle
\]
Ref.: is-intg-val 9-3(5)

2.2.2.3 Aggregates in generic parameter descriptors

(53) \( \text{is-generic-aggr} = \)
\( \text{is-generic-array} \lor \text{is-generic-struct} \lor \text{is-generic-scalar} \)

(54) \( \text{is-generic-array} = \)
\[
\langle s\text{-elem:is-generic-aggr} \rangle
\]

(55) \( \text{is-generic-struct} = \)
\( \text{is-generic-succ-list-1} \)

(56) \( \text{is-generic-succ} = \)
\[
\langle s\text{-aggr:is-generic-aggr} \rangle
\]

(57) \( \text{is-generic-scalar} = \)
\[
\langle s\text{-da:is-generic-da v is-},
\langle s\text{-dens:is-AL v is-UNAL v is-O} \rangle
\rangle
\]

(58) \( \text{is-generic-da} = \)
\( \text{is-generic-ariths} \lor \text{is-generic-string} \lor \text{is-pic} \lor \text{is-AREA} \lor \text{is-ENTRY} \lor \text{is-LABEL} \lor \text{is-OFFSET} \lor \text{is-FILE} \lor \text{is-TASK} \lor \text{is-EVENT} \)

(59) \( \text{is-generic-arith} = \)
\[
\langle s\text{-mode:is-REAL v is-CFLX v is-O},
\langle s\text{-base:is-DIC v is-BIN v is-O},
\langle s\text{-scale:is-FLT v is-FIX v is-O},
\langle s\text{-prec:is-intg-val v is-O},
\langle s\text{-scale-f:is-intg-val v is-O} \rangle
\rangle
\]
Ref.: is-intg-val 9-3(5)

2. ABSTRACT SYNTAX OF PROGRAM
2.2.4 Aggregates in allocate statements

(60)  is-generic-string =
   (<s-base:is-CHAR v is-BIT v is-Q>,
   <s-varying:is-VAR v is-Q>)

2.2.3 PICTURES

(68)  is-pic =
   is-dec-pic v is-sterling-pic v is-bin-pic v is-char-pic
(69) \text{is-dec-pic} =
\begin{align*}
\langle &\text{mode:is-CFLX} \vee \text{is-0}\rangle, \\
\langle &\text{st-field:is-dec-spec-list-1}\rangle, \\
\langle &\text{unit:is-intg-val} \vee \text{is-0}\rangle, \\
\langle &\text{scale-f:is-intg-val} \vee \text{is-0}\rangle, \\
\langle &\text{exp-sep:is-I-CHAR} \vee \text{is-0}\rangle, \\
\langle &\text{exp-field:is-dec-spec-list-1} \vee \text{is-0}\rangle
\rangle
\end{align*}

Ref.: is-intg-val 9-3(5)

(70) \text{is-dec-spec} =
\begin{align*}
\text{is-9-CHAR} \vee \text{is-E-CHAR} \vee \text{is-ASTER} \vee \text{is-Y-CHAR} \vee \text{is-T-CCHAR} \vee \text{is-I-CHAR} \vee \\
\text{is-E-CHAR} \vee \text{is-SIGN} \vee \text{is-PLUS} \vee \text{is-MINUS} \vee \text{is-DOLLAR} \vee \text{is-BlANK} \vee \text{is-COMMA} \vee \\
\text{is-SLASH} \vee \text{is-FCINT} \vee \text{is-C-CHAR} \vee \text{is-I-CHAR} \vee \text{is-E-CHAR}
\end{align*}

(71) \text{is-sterling-pic} =
\begin{align*}
\langle &\text{st-field:is-sterling-spec-list-1}\rangle, \\
\langle &\text{stat-part-end:is-intg-val}\rangle, \\
\langle &\text{round-end:is-intg-val}\rangle, \\
\langle &\text{shill-end:is-intg-val}\rangle, \\
\langle &\text{unit:is-intg-val} \vee \text{is-0}\rangle
\rangle
\end{align*}

Ref.: is-intg-val 9-3(5)

(72) \text{is-sterling-spec} =
\begin{align*}
\text{is-dec-spec} \vee \text{is-6-CHAR} \vee \text{is-7-CCHAR} \vee \text{is-8-CHAR} \vee \text{is-S-CCHAR}
\end{align*}

(73) \text{is-bin-pic} =
\begin{align*}
\langle &\text{mode:is-CFLX} \vee \text{is-0}\rangle, \\
\langle &\text{st-field:is-bin-spec-list-1}\rangle, \\
\langle &\text{unit:is-intg-val} \vee \text{is-0}\rangle, \\
\langle &\text{scale-f:is-intg-val} \vee \text{is-0}\rangle, \\
\langle &\text{exp-field:is-bin-spec-list-1} \vee \text{is-0}\rangle
\rangle
\end{align*}

Ref.: is-intg-val 9-3(5)

(74) \text{is-bin-spec} =
\begin{align*}
\text{is-1-CHAR} \vee \text{is-2-CCHAR} \vee \text{is-3-CHAR} \vee \text{is-SIGN}
\end{align*}

(75) \text{is-char-pic} =
\begin{align*}
\langle &\text{field:is-char-spec-list-1}\rangle
\end{align*}

(76) \text{is-char-spec} =
\begin{align*}
\text{is-X-CHAR} \vee \text{is-A-CCHAR} \vee \text{is-9-CHAR} \vee \text{is-ASTER} \vee \text{is-BlANK} \vee \text{is-COMMA} \vee \text{is-SLASH} \vee \text{is-FCINT}
\end{align*}
2.2.4 FORMATS

(77) \( \text{is-format} = \)
    \( \text{is-format-iter} \lor \text{is-format-item} \)

(78) \( \text{is-format-iter} = \)
    \( (\langle s\text{-rep:is-exp}, \right) \)
    \( \langle s\text{-format-list:is-format-list-item} \rangle \)

(79) \( \text{is-format-item} = \)
    \( \text{is-data-format} \lor \text{is-control-format} \lor \text{is-remote-format} \)

(80) \( \text{is-data-format} = \)
    \( \text{is-real-format} \lor \text{is-cplx-format} \lor \text{is-string-format} \lor \text{is-pic-format} \)

(81) \( \text{is-real-format} = \)
    \( (\langle s\text{-type:is-FLT} \lor \text{is-FIX}, \right) \)
    \( \langle s\text{-int:is-exp}, \right) \)
    \( \langle s\text{-d:is-exp}, \right) \)
    \( \langle s\text{-p:is-opt-exp} \rangle \)

(82) \( \text{is-cplx-format} = \)
    \( (\langle s\text{-type:is-CFLX}, \right) \)
    \( \langle s\text{-real:is-real-format} \lor \text{is-pic-format}, \right) \)
    \( \langle s\text{-imag:is-real-format} \lor \text{is-pie-format} \rangle \)

(83) \( \text{is-string-format} = \)
    \( (\langle s\text{-type:is-CHAR} \lor \text{is-BIT} \lor \text{is-BB}, \right) \)
    \( \langle s\text{-w:is-opt-exp} \rangle \)

(84) \( \text{is-pic-format} = \)
    \( (\langle s\text{-type:is-FIC} \lor \text{is-FPIC}, \right) \)
    \( \langle s\text{-pic:is-pic} \rangle \)

(85) \( \text{is-control-format} = \)
    \( (\langle s\text{-type:is-control-format-type}, \right) \)
    \( \langle s\text{-w:is-opt-exp} \rangle \)

(86) \( \text{is-control-format-type} = \)
    \( \text{is-SPACE} \lor \text{is-ESPACE} \lor \text{is-COL} \lor \text{is-ECOL} \lor \text{is-LINK} \lor \text{is-PAGE} \lor \text{is-SKIP} \)

(87) \( \text{is-remote-format} = \)
    \( (\langle s\text{-type:is-REMOTE}, \right) \)
    \( \langle s\text{-ref:is-ref} \rangle \)
2.3 STATEMENTS

(88) \textit{is-st} =
\begin{itemize}
\item \textit{<s-cond-part:is-cond-part>},
\item \textit{<s-label-list:is-id-ref-list>},
\item \textit{<s-prop-st:is-prop-st>}
\end{itemize}

(89) \textit{is-prop-st} =
\begin{itemize}
\item \textit{is-block} \textit{v} \textit{is-group} \textit{v} \textit{is-st-list-1} \textit{v} \textit{is-if-st} \textit{v} \textit{is-goto-st} \textit{v} \textit{is-call-st} \textit{v}
\item \textit{is-return-st} \textit{v} \textit{is-incorporate-st} \textit{v} \textit{is-fetch-st} \textit{v} \textit{is-release-st} \textit{v} \textit{is-end-st} \textit{v}
\item \textit{is-dummy-st} \textit{v} \textit{is-stop-st} \textit{v} \textit{is-return-st} \textit{v} \textit{is-assign-st} \textit{v} \textit{is-allocate-st} \textit{v}
\item \textit{is-free-st} \textit{v} \textit{is-on-st} \textit{v} \textit{is-reset-st} \textit{v} \textit{is-signal-st} \textit{v} \textit{is-access-st} \textit{v}
\item \textit{is-enable-st} \textit{v} \textit{is-disable-st} \textit{v} \textit{is-open-st} \textit{v} \textit{is-close-st} \textit{v} \textit{is-get-st} \textit{v}
\item \textit{is-put-st} \textit{v} \textit{is-read-st} \textit{v} \textit{is-write-st} \textit{v} \textit{is-rewrite-st} \textit{v} \textit{is-locate-st} \textit{v}
\item \textit{is-delete-st} \textit{v} \textit{is-unlock-st} \textit{v} \textit{is-display-st} \textit{v} \textit{is-null-st}
\end{itemize}

(90) \textit{is-null-st} =
\begin{itemize}
\item \textit{<s-st:is-null>}
\end{itemize}

2.3.1 BLOCK, GROUP

(91) \textit{is-block} =
\begin{itemize}
\item \textit{<s-decl-part:is-decl-part>},
\item \textit{<s-body-part:is-body-part>},
\item \textit{<s-cond-part:is-cond-part>},
\item \textit{<s-st-list:is-st-list-1>},
\item \textit{<s-indent:is-opt>}
\end{itemize}

(92) \textit{is-group} =
\begin{itemize}
\item \textit{is-while-group} \textit{v} \textit{is-centr-group}
\end{itemize}

(93) \textit{is-while-group} =
\begin{itemize}
\item \textit{<s-while:is-expr>},
\item \textit{<s-do-list:is-st-list-1>}
\end{itemize}

(94) \textit{is-centr-group} =
\begin{itemize}
\item \textit{<s-centr-var:is-ref>},
\item \textit{<s-spec-list:is-do-spec-list-1>},
\item \textit{<s-do-list:is-st-list-1>}
\end{itemize}

(95) \textit{is-do-spec} =
\begin{itemize}
\item \textit{<s-init:is-expr>},
\item \textit{<s-by:is-opt-expr>},
\item \textit{<s-to:is-opt-expr>},
\item \textit{<s-while:is-opt-expr>}
\end{itemize}
2.3.2 FLOW OF CONTROL STATEMENTS

(96) \text{is-if-st} =
\begin{align*}
\langle \text{s-st:is-IF}, \\
\text{s-expr:is-expr}, \\
\text{s-then:is-st}, \\
\text{s-else:is-st} \rangle
\end{align*}

(97) \text{is-goto-st} =
\begin{align*}
\langle \text{s-st:is-GOTO}, \\
\text{s-label:is-ref} \rangle
\end{align*}

(98) \text{is-call-st} =
\begin{align*}
\langle \text{s-st:is-CALL}, \\
\text{s-entry:is-ref}, \\
\text{s-arg-list:is-expr-list}, \\
\text{s-fa-cpt:is-fa-opt \lor is-0} \rangle
\end{align*}

(99) \text{is-fa-cpt} =
\begin{align*}
\langle \text{s-task:is-ref \lor is-*}, \\
\text{s-event:is-ref \lor is-*}, \\
\text{s-pri:is-opt-expr} \rangle
\end{align*}

(100) \text{is-return-st} =
\begin{align*}
\langle \text{s-st:is-RETURN}, \\
\text{s-expr:is-cpt-expr} \rangle
\end{align*}

(101) \text{is-incorporate-st} =
\begin{align*}
\langle \text{s-st:is-INCCEPARE}, \\
\text{s-text:is-incorporate-text} \rangle
\end{align*}

(102) \text{is-incorporate-text} =
\begin{align*}
\text{Note: This predicate is implementation defined.}
\end{align*}

(103) \text{is-fetch-st} =
\begin{align*}
\langle \text{s-st:is-FETCH}, \\
\text{s-entry-list:is-ref-list-1} \rangle
\end{align*}

(104) \text{is-release-st} =
\begin{align*}
\langle \text{s-st:is-RELEASE}, \\
\text{s-entry-list:is-ref-list-1} \rangle
\end{align*}

(105) \text{is-wait-st} =
\begin{align*}
\langle \text{s-st:is-WAIT}, \\
\text{s-event-list:is-ref-list-1}, \\
\text{s-event-number:is-opt-expr} \rangle
\end{align*}
(106) \text{is-delay-st} = \\
\quad (\langle \text{s-st:is-DELAY}, \\
\quad \text{s-time:is-expr} \rangle)

(107) \text{is-exit-st} = \\
\quad (\langle \text{s-st:is-EXIT} \rangle)

(108) \text{is-stop-st} = \\
\quad (\langle \text{s-st:is-STOP} \rangle)

2.3.3 STORAGE MANIPULATING STATEMENTS

(109) \text{is-assign-st} = \\
\quad (\langle \text{s-st:is-ASSIGN}, \\
\quad \text{s-lp:is-ref-list-1}, \\
\quad \text{s-expr:is-expr}, \\
\quad \text{s-bynae:is-opt} \rangle)

(110) \text{is-allocate-st} = \\
\quad (\langle \text{s-st:is-ALLOCATE}, \\
\quad \text{s-list:is-al-list-1} \rangle)

(111) \text{is-al} = \\
\quad (\langle \text{s-id:is-id}, \\
\quad \text{s-n:is-opt-n}, \\
\quad \text{s-aggr:is-al-aggr \& is-\#}, \\
\quad \text{s-ptr:is-opt-ref}, \\
\quad \text{s-area:is-opt-ref} \rangle)

(112) \text{is-free-st} = \\
\quad (\langle \text{s-st:is-FREE}, \\
\quad \text{s-list:is-free-list-1} \rangle)

(113) \text{is-free} = \\
\quad (\langle \text{s-ref:is-ref}, \\
\quad \text{s-area:is-opt-ref} \rangle)
2.3.4 CONDITION AND ATTENTION HANDLING STATEMENTS

(114) is-or-st =
\[
\langle\text{s-st:is-CH}, \\
\text{s-cond:is-cond}, \\
\text{s-snap:is-opt}, \\
\text{s-ch-unit:is-st} \land \text{is-SYSTEM}\rangle
\]

(115) is-revert-st =
\[
\langle\text{s-st:is-EEVET}, \\
\text{s-cond:is-cond}\rangle
\]

(116) is-signal-st =
\[
\langle\text{s-st:is-SIGNAL}, \\
\text{s-cond:is-cond}\rangle
\]

(117) is-cond =
\[
is-prefix-cond \land \text{is-AREA} \land \text{is-named-io-cond} \land \text{is-ERROR} \land \text{is-FINISH} \land \\
is-progr-named-cond \land \text{is-attn-cond}
\]

(118) is-prefix-cond =
\[
is-CONV \land \text{is-FOFL} \land \text{is-OFL} \land \text{is-SIZE} \land \text{is-STRG} \land \text{is-STRZ} \land \text{is-SUBRG} \land \text{is-UPFL} \land \\
is-ZDIV \land \text{is-check-cond}
\]

(119) is-check-cond =
\[
\langle\text{s-ref-list:is-ref-list-1}, \\
\text{s-cond:is-CHECK}\rangle
\]

(120) is-named-io-cond =
\[
\langle\text{s-ref:is-ref}, \\
\text{s-cond:is-io-cond}\rangle
\]

(121) is-io-cond =
\[
is-BOV \land \text{is-BOV} \land \text{is-ENDF} \land \text{is-ENDP} \land \text{is-KEY} \land \text{is-NAME} \land \text{is-REC} \land \text{is-TMT} \land \\
is-UNDF \land \text{is-PEND}
\]

(122) is-progr-named-cond =
\[
\langle\text{s-id:is-id}, \\
\text{s-n:is-opt-n}, \\
\text{s-cond:is-COND}\rangle
\]

(123) is-attn-cond =
\[
\langle\text{s-atttn-list:is-id-ref-list}, \\
\text{s-cond:is-ATTN}\rangle
\]
30 April 1969

(124) \text{is-access-st} = 
\begin{align*}
&\langle s-st:is-ACCESS, \\
&\quad s-cond:is-attr-cond, \\
&\quad s-else:is-st \lor is-0 \rangle
\end{align*}

(125) \text{is-enable-st} = 
\begin{align*}
&\langle s-st:is-ENABLE, \\
&\quad s-list:is-enable-list-1 \rangle
\end{align*}

(126) \text{is-enable} = 
\begin{align*}
&\langle s-cond:is-attr-cond, \\
&\quad s-spec:is-ACC \land is-ASYN, \\
&\quad s-event:is-opt-ref \rangle
\end{align*}

(127) \text{is-disable-st} = 
\begin{align*}
&\langle s-st:is-DISABLE, \\
&\quad s-cond:is-attr-cond \rangle
\end{align*}

2.3.5 INPUT AND OUTPUT STATEMENTS

(128) \text{is-open-st} = 
\begin{align*}
&\langle s-st:is-OPEN, \\
&\quad s-list:is-open-list-1 \rangle
\end{align*}

(129) \text{is-open} = 
\begin{align*}
&\langle s-file:is-ref, \\
&\quad s-title:is-opt-exp, \\
&\quad s-isz:is-opt-exp, \\
&\quad s-bisz:is-opt, \\
&\quad s-pez:is-opt-exp, \\
&\quad s-open-attr:is-file-attr-set, \\
&\quad s-env-attr:is-env-attr, \\
&\quad s-volume:is-opt \rangle
\end{align*}

(130) \text{is-close-st} = 
\begin{align*}
&\langle s-st:is-CLOSE, \\
&\quad s-list:is-close-list-1 \rangle
\end{align*}

(131) \text{is-close} = 
\begin{align*}
&\langle s-file:is-ref, \\
&\quad s-env-attr:is-env-attr, \\
&\quad s-volume:is-opt \rangle
\end{align*}

(132) \text{is-get-st} = 
\text{is-file-get-st} \lor \text{is-string-get-st}
ABSTRACT SYNTAX AND INTERPRETATION OF PL/I

(133) is-file-get-st =
    (<s-st:is-GET>,
    <s-file:is-ref>,
    <s-spec:is-data-spec v is-D>,
    <s-copy:is-opt>,
    <s-skip:is-opt-expr>)

(134) is-string-get-st =
    (<s-st:is-GET>,
    <s-string:is-expr>,
    <s-base:is-CHAR v is-BIT>,
    <s-spec:is-data-spec>)

(135) is-put-st =
    is-file-put-st v is-string-put-st

(136) is-file-put-st =
    (<s-st:is-PUT>,
    <s-file:is-ref>,
    <s-spec:is-data-spec v is-U>,
    <s-page:is-opt>,
    <s-line:is-opt-expr>,
    <s-skip:is-opt-expr>)

(137) is-string-put-st =
    (<s-st:is-PUT>,
    <s-string:is-ref>,
    <s-base:is-CHAR v is-BIT>,
    <s-spec:is-data-spec>)

(138) is-data-spec =
    is-list-data-dir v is-edit-dir-list-1

(139) is-list-data-dir =
    (<s-data-list:is-item-list-1>,
    <s-type:is-LIST v is-DATA v is-ALL-DATA>)

(140) is-edit-dir =
    (<s-data-list:is-item-list-1>,
    <s-format-list:is-format-list-1>)

(141) is-item =
    is-contr-item v is-expr

(142) is-contr-item =
    (<s-contr-var:is-ref>,
    <s-spec-list:is-do-spec-list-1>,
    <s-do-list:is-item-list-1>)}
(143) is-read-st =
    is-into-read-st \lor is-set-read-st \lor is-ignore-read-st

(144) is-into-read-st =
    
    \{<s-st:is-READ>,
    <s-file:is-ref>,
    <s-into:is-ref>,
    <s-ident:is-opt-expr>,
    <s-into:is-opt-ref>,
    <s-nolock:is-opt>,
    <s-event:is-opt-ref>\}

(145) is-set-read-st =
    
    \{<s-st:is-READ>,
    <s-file:is-ref>,
    <s-set:is-ref>,
    <s-ident:is-opt-expr>,
    <s-ident:is-opt-ref>\}

(146) is-ignore-read-st =
    
    \{<s-st:is-READ>,
    <s-file:is-ref>,
    <s-ignore:is-opt-expr>,
    <s-event:is-opt-ref>\}

(147) is-write-st =
    
    \{<s-st:is-BWRITE>,
    <s-file:is-ref>,
    <s-from:is-ref>,
    <s-ident:is-opt-expr>,
    <s-event:is-opt-ref>\}

(148) is-rewrite-st =
    
    \{<s-st:is-REWRITE>,
    <s-file:is-ref>,
    <s-from:is-opt-ref>,
    <s-ident:is-opt-expr>,
    <s-event:is-opt-ref>\}

(149) is-locate-st =
    
    \{<s-st:is-LOCATE>,
    <s-file:is-ref>,
    <s-id:is-id>,
    <s-pnt:is-opt>,
    <s-ptr:is-opt-ref>,
    <s-ident:is-opt-expr>\}
(150) \texttt{is-delete-st} =
\begin{align*}
&\langle \texttt{s-st:is-DELETE}, \\
&\texttt{s-file:is-ref}, \\
&\texttt{s-ident:is-opt-expr}, \\
&\texttt{s-event:is-opt-ref} \rangle 
\end{align*}

(151) \texttt{is-unlock-st} =
\begin{align*}
&\langle \texttt{s-st:is-UNLOCK}, \\
&\texttt{s-file:is-ref}, \\
&\texttt{s-ident:is-expr} \rangle 
\end{align*}

(152) \texttt{is-display-st} =
\begin{align*}
&\langle \texttt{s-st:is-DISPLAY}, \\
&\texttt{s-ident:is-expr}, \\
&\texttt{s-idto:is-opt-ref}, \\
&\texttt{s-event:is-opt-ref} \rangle 
\end{align*}

2.4 EXPRESSIONS

(153) \texttt{is-expr} = 
\begin{align*}
\texttt{is-infix-expr} \vee \texttt{is-prefix-expr} \vee \texttt{is-paren-expr} \vee \texttt{is-ref} \vee \texttt{is-const} \vee \texttt{is-isub} 
\end{align*}

(154) \texttt{is-infix-expr} =
\begin{align*}
&\langle \texttt{s-opr:is-infix-opr}, \\
&\texttt{s-op-1:is-expr}, \\
&\texttt{s-op-2:is-expr} \rangle 
\end{align*}

(155) \texttt{is-infix-opr} =
\begin{align*}
\texttt{is-OR} \vee \texttt{is-AND} \vee \texttt{is-GT} \vee \texttt{is-GE} \vee \texttt{is-EQ} \vee \texttt{is-LE} \vee \texttt{is-LT} \vee \texttt{is-NE} \vee \texttt{is-EQV} \vee \texttt{is-ADD} \vee \texttt{is-SUBLT} \vee \texttt{is-MULT} \vee \texttt{is-LIV} \vee \texttt{is-EXP} 
\end{align*}

(156) \texttt{is-prefix-expr} =
\begin{align*}
&\langle \texttt{s-opr:is-prefix-opr}, \\
&\texttt{s-op:is-expr} \rangle 
\end{align*}

(157) \texttt{is-prefix-opr} =
\begin{align*}
\texttt{is-NOT} \vee \texttt{is-PLUS} \vee \texttt{is-MINUS} 
\end{align*}

(158) \texttt{is-paren-expr} =
\begin{align*}
&\langle \texttt{s-op:is-expr} \rangle 
\end{align*}
(159) is-ref =
   (<s-id-list:is-id-list-1>,
    <s-n:is-opt-n>,
    <s-ptr:is-opt-ref>,
    <s-sl:is-subscr-expr-list>,
    <s-ap:is-expr-list-list>)

(160) is-subscr-expr =
   is-expr \is-*

(161) is-ccnst =
   is-arithm-const \is-string-const

(162) is-arithm-const =
   (<s-da:is-prop-arithm>,
    <s-v:is-num-val>)

Ref.: is-num-val 9-3(3)

(163) is-string-ccnst =
   (<s-da:is-fixed-string-eda>,
    <s-v:is-char-val-list \is-bit-val-list>)

Ref.: is-char-val 9-3(6)
       is-bit-val 9-4(11)

(164) is-fixed-string-eda =
   (<s-base:is-CHAR \is-EDT>,
    <s-length:is-intg-val>)

Ref.: is-intg-val 9-3(5)

(165) is-isub =
   (<s-i:is-intg-val>)

Ref.: is-intg-val 9-3(5)

(166) is-opt-expr =
   is-expr \is-Ω

(167) is-opt-ref =
   is-ref \is-Ω
2.5 APPENDIX: AUXILIARY DEFINITIONS

This section contains auxiliary predicate definitions for aggregate attributes. Though they do not describe program text itself, the objects described by them are closely related to objects occurring in the program text.

Section 2.5.1 describes the structure of evaluated aggregate attributes as produced by the instruction `eval-aggr` and occurring, e.g., in the `s-eva` component of generations. Section 2.5.2 describes the general structure of aggregate attributes, irrespective whether they are evaluated or not and whether they occur in variable declarations, parameter descriptors, etc.

2.5.1 EVALUATED AGGREGATE ATTRIBUTES

(168) \( \text{is-eva} = \)
\( \text{is-array-eva} \lor \text{is-struct-eva} \lor \text{is-scalar-eva} \)

(169) \( \text{is-array-eva} = \)
\( \langle s-lbd: \text{is-intg-val} \lor \text{is-*}, \)
\( s-ubd: \text{is-intg-val} \lor \text{is-*}, \)
\( s-elem: \text{is-eva} \rangle \)

Ref.: is-intg-val 9-3(5)

(170) \( \text{is-struct-eva} = \)
\( \text{is-succ-eva-list-1} \)

(171) \( \text{is-succ-eva} = \)
\( \langle s-aggr: \text{is-eva} \rangle \)

(172) \( \text{is-scalar-eva} = \)
\( \langle s-da: \text{is-eoa}, \)
\( s-dens: \text{is-AL} \lor \text{is-UNAL} \rangle \)

(173) \( \text{is-eoa} = \)
\( \text{is-arithm} \lor \text{is-descr-string} \lor \text{is-pic} \lor \text{is-descr-area} \lor \text{is-ENTRY} \lor \text{is-LABEL} \lor \)
\( \text{is-OFFSET} \lor \text{is-ITE} \lor \text{is-FILE} \lor \text{is-TASK} \lor \text{is-EVENT} \)

2.5.2 GENERAL STRUCTURE OF AGGREGATE ATTRIBUTES

(174) \( \text{is-aggr} = \)
\( \text{is-array} \lor \text{is-struct} \lor \text{is-scalar} \lor \text{is-} \)
is-array =
  (is-lbd:is-extent, is-ubd:is-extent, is-elem:is-aggr)

is-extent =
  is-expr v is-* v is-refer v is-intg-val v is-0

Ref.: is-intg-val 9-3(5)

is-struct =
  is-succ-list-1

is-succ =
  (is-qual:is-id v is-0, is-aggr:is-aggr)

is-scalar =
  (is-da:is-da v is-* v is-0, is-dens:is-AL v is-UNAL v is-0, is-init:is-init v is-0)

is-da =
  is-arith v is-string v is-pic v is-area v is-entry v is-label-da v is-offset v is-FTR v is-FILE v is-TASK v is-EVENT

is-arith =
  is-prop-arith v is-generic-arith

is-string =
  is-prop-string v is-descr-string v is-generic-string

is-area =
  is-prop-area v is-descr-area v is-AREA

is-entry =
  is-entry-da v is-entry-const v is-ENTRY

Note: Often it is convenient to handle the complete declaration of an entry constant in the same context as the data attribute of an entry variable.
The PL/I machine is an abstract sequential machine which defines PL/I by interpreting abstract PL/I programs satisfying the rules of the abstract syntax (cf. chapter 2). The interpretation of a PL/I program is defined by specifying a set of possible computations for that program. A computation is a sequence of machine states $t_0,t_1,t_2,...$ satisfying the two conditions that, first, $t_0$ is a possible initial state as defined in section 4.1 and, second, the transition from any state to its successor is a valid computation step as defined in section 3.7.

All machine states are objects satisfying the predicate is-state defined below. The present chapter describes the structure and use of the different components of a state $t$ of the PL/I machine and, in the last section, the transition between successive machine states of a computation.

(1) is-state =

<is-vr>,<is-es>,<is-un>,<is-at>,<is-dn>,<is-fu>,<is-td>,<is-et>,
<is-an>,<is-ev>,<is-tn>,<is-pa>

Ref.: is-vr 3-15(32)  
is-es 3-27(100)  
is-un 3-10(19)  
is-at 3-11(21)  
is-dn 3-12(22)  
is-fu 3-25(91)  
is-td 3-5(11)  
is-et 3-6(12)  
is-an 3-20(107)  
is-ev 3-19(66)  
is-tn 3-4(73)  
is-pa 3-3(2)
Abbreviations and terms

For the convenience of reference to parts of the state the following abbreviations and terms for components of a given state \( E \) are introduced and used throughout the document:

- \( s = s-s(E) \) — the storage
- \( s-e = s-es(E) \) — the external storage
- \( s-u = s-un(E) \) — the unique name counter
- \( s-at = s-at(E) \) — the attribute directory
- \( s-dn = s-dn(E) \) — the denotation directory
- \( s-fu = s-fu(E) \) — the file unique directory
- \( s-td = s-td(E) \) — the time and date part
- \( s-st = s-st(E) \) — the event trace
- \( s-m = s-m(E) \) — the message part
- \( s-an = s-an(E) \) — the attention directory
- \( s-ev = s-ev(E) \) — the attention environment directory
- \( s-tn = s-tn(E) \) — the current task-event name
- \( s-pa = s-pa(E) \) — the parallel action part
- \( s-te = s-te(s-tn(E)) \cdot s-pa(E) \) — the task-event specification
- \( s-ag = s-ag(s-tn(E)) \cdot s-pa(E) \) — the aggregate directory
- \( s-id = s-id(s-tn(E)) \cdot s-pa(E) \) — the file directory
- \( s-en = s-en(s-tn(E)) \cdot s-pa(E) \) — the enabling state
- \( s-ba = s-ba(s-tn(E)) \cdot s-pa(E) \) — the block activation name
- \( s-ei = s-ei(s-tn(E)) \cdot s-pa(E) \) — the epilogue information
- \( s-cs = s-cs(s-tn(E)) \cdot s-pa(E) \) — the condition status
- \( s-ev = s-ev(s-tn(E)) \cdot s-pa(E) \) — the event
- \( s-en = s-en(s-tn(E)) \cdot s-pa(E) \) — the enabling
- \( s-ba = s-ba(s-tn(E)) \cdot s-pa(E) \) — the block activation
- \( s-ei = s-ei(s-tn(E)) \cdot s-pa(E) \) — the epilogue
- \( s-cs = s-cs(s-tn(E)) \cdot s-pa(E) \) — the condition

For the use in basic instruction definitions the following abbreviations are used for the selectors selecting these state components.

- \( s-s = s-s \)
- \( s-es = s-es \)
- \( s-un = s-un \)
- \( s-at = s-at \)
- \( s-dn = s-dn \)
- \( s-fu = s-fu \)
- \( s-td = s-td \)
- \( s-et = s-et \)
- \( s-em = s-em \)
- \( s-an = s-an \)
- \( s-ev = s-ev \)
- \( s-tn = s-tn \)
- \( s-pa = s-pa \)
- \( s-te = s-te(s-tn(E)) \cdot s-pa \)
- \( s-ag = s-ag(s-tn(E)) \cdot s-pa \)
- \( s-id = s-id(s-tn(E)) \cdot s-pa \)
- \( s-en = s-en(s-tn(E)) \cdot s-pa \)
- \( s-ba = s-ba(s-tn(E)) \cdot s-pa \)
- \( s-ei = s-ei(s-tn(E)) \cdot s-pa \)
- \( s-cs = s-cs(s-tn(E)) \cdot s-pa \)
- \( s-ev = s-ev(s-tn(E)) \cdot s-pa \)
- \( s-en = s-en(s-tn(E)) \cdot s-pa \)
- \( s-ba = s-ba(s-tn(E)) \cdot s-pa \)
- \( s-ei = s-ei(s-tn(E)) \cdot s-pa \)
- \( s-cs = s-cs(s-tn(E)) \cdot s-pa \)

Note: A reader who is not interested in the existence of tasks may consider the current task-event name \( s-tn(E) \) as constant and the components of \( (s-tn(E)) \cdot s-pa(E) \) like immediate components of the state \( E \). So he can forget that some of the above selectors are composite ones.
3.1 PARALLEL ACTIONS

PL/I provides means to specify parallel execution of parts of a program, namely the parallel execution of a called procedure with the calling block activation, the so-called "task", and the parallel execution of data transmission of an input or output statement with the actions of the program itself, the so-called "input or output event". The language defines relations between different parallel tasks or input or output events, but it does not define the effects of actual parallel execution. So the effect of sharing of the same piece of storage (except that of event variables which are used for synchronization) or external storage by different tasks without synchronization by wait statements is undefined. Apart from this undefinedness, which is not reflected by the PL/I machine, the parallel actions can be described by a sequentialized model, which executes one action of one task after another action of possibly another task. This model is realized by the PL/I machine.

3.1.1 THE PARALLEL ACTION PART PA

Some of the state components of the PL/I machine, e.g. the storage \( \xi \), are "global", i.e., they are not associated with a special task or event. Other state components, e.g. the control \( \xi_c \), are "task local", i.e., each task and each event has its own one. The task local state components are collected in the parallel action part PA. It has for each active task or event an individual component under an individual unique selector. The main task, which is the only one in the initial state of the machine, has the constant selector \( s_{\text{main}} \). Any other task or event has a unique name, satisfying the predicate is-n, which is newly created when the task or event is activated. Whenever a parallel action is started an individual component for it is entered into the parallel action part PA, and whenever a parallel action is terminated its component is deleted. The component consists of ten task local state components, that for an input or output event of two only. Besides tasks and input and output events the parallel action part can possess a third type of components, those for attention events. They do not really specify parallel actions (they have no control component), but serve only to reserve event variable generations as active as long as they are associated with enabled attention events.

\[
\begin{align*}
\text{is-pa} & = \\
& = (\{ \text{is-task} \lor \text{is-ic-ev} \lor \text{is-attn-ev} \} \lor \text{is-tn}(tn))
\end{align*}
\]

\[
\begin{align*}
\text{is-task} & = \\
& = (\langle s^t: \text{is-task-te} \rangle, \langle s^a: \text{is-ag} \rangle, \langle s^f: \text{is-fd} \rangle, \langle s^e: \text{is-en} \rangle, \langle s^b: \text{is-ba} \rangle, \langle s^i: \text{is-ei} \rangle, \langle s^c: \text{is-cs} \rangle, \langle s^d: \text{is-d} \rangle, \langle s^c: \text{is-cl} \rangle, \langle s^c: \text{is-c} \rangle)
\end{align*}
\]

Ref.: is-ag 3-19(28)
      is-fd 3-24(88)
      is-en 3-19(71)
      is-ba 3-7(15)
      is-ei 3-8(16)
      is-cs 3-21(79)
      is-d 3-7(14)
      is-cl 3-9(17)

3. STATE COMPONENTS AND COMPUTATION OF THE PL/I MACHINE

3
ABSTRACT SYNTAX AND INTERPRETATION OF PL/I

3.1.2 THE CURRENT TASK-EVENT NAME TIN

The current task-event name TIN specifies during a computation step and in the following machine state the selector of that parallel action, out of the control of which an instruction is executed.

(6) \[ \text{is-tn}(\text{tn}) = \text{is-n}(\text{tn}) \lor (\text{tn} = \text{s-main}) \]

3.1.3 THE TASK-EVENT SPECIFICATION TE

Each task and each input and output event and each attention event has a task-event specification TE. It contains information relevant for the parallel execution, in particular for the termination of the task or event.

The specification TE for a task consists of the following five components:

(1) The generation of the associated task variable. Its storage part holds the priority value of the task, which is used in implementation defined way by the priority scheduler for the selection of the task for execution.

(2) The generation of the associated event variable which is used for synchronization of other tasks with this task's termination. Additionally it enables the programmer to store status information about the task.

(3) The wait state flag. It is set to * if the task is waiting and is not to be selected for execution.

(4) The set of all pointers of based storage allocated (and not yet freed) during the task. They have to be freed at task end.

(5) The set of task-event names of all input and output events activated by the task. They have to be terminated abnormally at task end if they did not terminate earlier.

The specification TE of an input or output event consists of the following seven components:

(1) The task-event name of the task which activated the event.

(2) The associated event generation as described above.

(3) The wait state flag as described above.

(4) An entry used by the completing wait statement to unlock the transmitted record if necessary.

(5) An entry used by the completing wait statement to handle end-of-volume and begin-of-volume conditions if necessary.

(6) The list of input/output on conditions accumulated during the event to be raised by the completing wait statement.
(7) In the case of input references of the record and key variables to be used for raising of the check condition by the completing wait statement if necessary.

The specification $TE$ of an attention event consists of an associated event generation only.

(7) $is-te =
    is-task-te \lor is-ic-te \lor is-attn-te$

(8) $is-task-te =
    (<s-tw:is-gen>,
     <s-ev:is-gen>,
     <s-wait:is-opt>,
     <s-free-set:is-ptr-val-set>,
     <s-io-ev:is-nil>)$

Ref.: $is-gen 3-14(30)$
      $is-ptr-val 3-15(36)$

(9) $is-ic-te =
    (<s-tm:is-tn>,
     <s-ev:is-gen>,
     <s-wait:is-opt>,
     <s-unlock:is-e-delete-st \lor is-e-write-st \lor is-e-rewrite-st \lor is-9>,
     <s-eov-bov:is-e-delete-st \lor is-e-rewrite-st \lor is-e-write-st \lor is-e-into-read-st \lor is-e-ignore-read-st \lor is-9>,
     <s-cond:is-f-cond-list \lor is-9>,
     <s-check:is-ref-list>)$

Ref.: $is-gen 3-14(30)$
      $is-e-[pred] 11-36(99)$
      $is-f-cond 3-21(78)$

(10) $is-attn-te =
    (<s-ev:is-gen>)$

Ref.: $is-gen 3-14(30)$

3.1.4 THE TIME AND DATE PART $TP$

The time and date part has two integer components giving the current time and the date to be used by the $TIME$ and $DATE$ built-in functions and by the delay statement. Additionally it has the set of those times for which delay statements in different tasks currently are waiting. In each computation step the time is updated in implementation defined way and it is tested whether the time for which any delay statement is waiting has been passed.
(11) is-ta =
    \{(s-time:is-intg-val),
     (s-date:is-intg-val),
     (s-delay:is-intg-val-set)\}

Ref.: is-intg-val 9-3(5)

3.1.5 THE EVENT TRACE ET

A wait statement has to inspect a list of specified event variable generations for completion before the actions of the containing task continue. In this respect not only those generations have to be recognized as complete whose value is currently "completed" at the time when inspected, but also those whose value has been "completed" at any time later than the start of the wait statement (and is possibly "incomplete" when inspected). To recognize this situation serve the event trace ET, a global state component, which keeps track of the sequence in time of all completions of event generations and of all starts of wait statements. Whenever an event variable is set complete its generation is added to the event trace and whenever a wait statement starts a unique name identifying it is added. So each wait statement can recognize which generations were added to the event trace later than its start.

(12) is-et =
    is-et-entry-list

(13) is-et-entry =
    is-gen ∨ is-n

Ref.: is-gen 3-14(30)

3.2 FLOW OF CONTROL

The correct handling of block activations within a single task is governed by the dump D, the block activation name BA and the epilogue information EI. The flow of control through the nested structure of statements within a block activation is governed by the control information CI.

3.2.1 THE DUMP D

The dump serves to maintain the history of block activations as long as they are yet active. It is handled as a pushdown stack in order to control the dynamic nesting of block activations.

All information in the state of the PL/I machine that is associated with a single block activation is contained in five block local state components: the block activation name BA, the epilogue information EI, the condition status CS, the control information CI and the control C. These state components have to be available for the complete time of a block activation; in particular, they have to be reserved for the time of a nested block activation, which maintains its own local state components, in order to be available after termination of the nested block activation.

This is realized by use of the dump D. Whenever a new block activation starts, the five block local state components of the old block activation and the old dump are entered as components into the new dump D. Thereby the former components of the dump become now components of the dump component of the dump, and so on, i.e.
all components of the dump are pushed down for one level. Wherever a block activation terminates, the six components of the dump $D$ are reinstalled as components of the state; thereby, all components of the dump are popped up for one level.

This mechanism starts with $D$ as dump component of the initial state, so that $D=\emptyset$ is characteristic for the outermost block activation of the program. When a new task is created, the dump of the current block activation of the attaching task is inherited to the attached task, since in some instances information of the dynamically surrounding block activations of the attaching task are necessary (call of entry variables, cf. 6.2.1, the revert statement, cf. 10.2.3).

Apart from stacking and unstacking of the dump when activating and terminating a block, the information in the dump $D$ is usually never changed. The only exception is the complete overwriting of the control component in the dump in the case of abnormal termination of a block by means of a goto (cf. 6.4), return (cf. 6.2.4) and task exit (cf. 5.2).

(14) $is-d = \begin{align*}
&\text{is-G} \vee \\
&(\text{is-ba} \wedge \text{is-ha}), \\
&(\text{is-ci} \wedge \text{is-ci}), \\
&(\text{is-ds} \wedge \text{is-ds}), \\
&(\text{is-ci} \wedge \text{is-ci}), \\
&(\text{is-ci} \wedge \text{is-ci})
\end{align*}
$

Ref.: $is-cs\ 3-21(79)$

3.2.2 THE BLOCK ACTIVATION NAME $\text{BA}$

The meaning of an entry, label or format constant includes the knowledge of the block activation in which it is declared: An entry constant may only be called if this block activation is yet active, even if the entry constant is passed outside that block activation via assignment to entry variables. A goto statement to a label constant shall lead into that block activation, and it is an error if it is no more active. A format constant may be used by means of a record format only in that block activation itself.

For this purpose each block activation is identified by a unique name, the block activation name $\text{BA}$ which is one of its local state components. It is created and entered into the state before the denotations of declarations are evaluated by the prologue. It is then copied into the denotations of entry, label, and format constants.

The outermost block activation of any task (it does not contain any declarations except the external procedures which are declared in the outermost block activation of the main task) has $\emptyset$ instead of a unique name as block activation name. This is tested by the goto statement when terminating one block activation after the other, in order to avoid a goto out of a task.

To ensure that a goto statement works correctly if it leads out of an attention on unit called by an asynchronous interrupt, $\text{BA}$ is initialized to $\ast$ until a unique name is created and inserted (this effects that a goto will in any case terminate this block activation without error, since it is neither the block activation name of the label denotation, nor $\emptyset$). In those block activations which have no declarations, namely those of condition and attention calls, the $\ast$ as block activation name needs not to be replaced by a unique name.

(15) $is-ba = \begin{align*}
&\text{is-b} \wedge \text{is-} \wedge \text{is-q}
\end{align*}$

3. STATE COMPONENTS AND COMPUTATION OF THE PL/I MACHINE
3.2.3 THE EPILOGUE INFORMATION $E_I$

The epilogue information $E_I$ contains all information necessary to terminate a block activation. This is the set of all local automatic and dummy variables to be freed at block end (represented by their aggregate names) and the set of all tasks attached during the block activation to be terminated at block end (represented by their task names).

In addition to this task information necessary at each block end, the return from a procedure, in particular by a return statement, needs further information (cf. 5.2.4):

1) Whether the current block activation is that of a begin block (in this case also the containing block activation has to be terminated), a procedure, an on-unit (in this case a return statement is erroneous), or an attention (in this case a return statement cannot occur).

2) Whether the innermost procedure activation was initiated by a call statement or a function reference; in the latter case the aggregate name for the function value is needed.

3) If the innermost procedure activation was initiated by a function reference, the return type of the called entry point is needed for conversion of the returned function value.

4) Whether the finish condition is to be raised or not, i.e. whether the innermost procedure activation was initiated by the initial program call or not.

This information is contained in the different components of the epilogue information $E_I$. The last three mentioned components, i.e. those giving information about the procedure activation to be terminated by a return statement (the innermost active procedure activation), are needed also within contained begin blocks. They are therefore inherited into the epilogue information of starting begin blocks, but not of procedures or on units.

The outermost block activation of a task does not need an epilogue information, it is $Q$ in this case.

(16) $is\text{-}ei =$

$\langle s\text{-}block\text{-}act : is\text{-}BLOCK \quad \langle s\text{-}free\text{-}set : is\text{-}n\text{-}set \rangle ,$
$\langle s\text{-}task\text{-}set : is\text{-}n\text{-}set \rangle ,$
$\langle s\text{-}ret\text{-}type : descr\text{-}scalar \quad is\text{-}Q \rangle ,$
$\langle s\text{-}main : is\text{-}opt \rangle \rangle$

3.2.4 THE CONTROL INFORMATION $C_I$

The control information $C_I$ governs the flow of control through the nested structure of statements within a single block activation.

It contains a text component which usually is a statement list, an if-statement or an access statement and an index component which usually is either an integer value or a truth value (T or F). The index identifies a component of the text, which is the statement currently being interpreted: If the text is a statement list, the index is an integer value $i$ identifying the $i$-th element of the statement list; if the text is an if-statement or access statement, the index is T or F identifying the s-then or s-else component, respectively, of the if-statement or access statement.
Whenever a statement list or a component of an if-statement or access statement is to be interpreted, the complete statement list, if-statement or access statement is inserted into the text component of CI, and the index component of CI is set to the index identifying the component statement to be interpreted first: 1 in the case of the statement list, the truth value of the decision expression in the case of the if statement, and ? in the case of the access statement. Whenever the interpretation of a statement in a statement list, which is not the last one, terminates, the index is incremented by 1 and the next statement is interpreted.

Whenever during the interpretation of a statement at a statement list, or a component of an if-statement or access statement is to be interpreted, by entering the text and index into the control information the old text and index components of CI would be overwritten. But since they are needed for continuation after completion of the statement st, they have to be reserved somewhere during the interpretation of the nested statement list, if-statement, or access statement. For this purpose the complete control information CI is stacked into a third component s-ci (CI) of the new control information whenever a new text component is inserted, i.e. whenever a new level of statement nesting is entered.

Similarly, the control c is stacked in a fourth component of the control information, whenever a new level of statement nesting is entered. This control contains the actions to be performed after return from the nested statement level, while the control c itself specifies only the actions to be performed for interpretation of the new level.

The complete control information works in this way as a stack localizing uniquely that statement within the nested structure of statement lists, if statements and access statements of a block activation, which is currently being interpreted. Whenever a new nested level is entered a new level in CI is introduced, whenever a level is completed the old control information and control is reinstalled from CI. This mechanism is started by initializing CI to 0 on start of a block activation.

The control information CI is a block local state component stacked into the dump D by nested block activations and reinstalled by their epilogues. So, on return from a nested block activation, the flow of control continues at that point, where it was interrupted by the nested block activation.

(17) is-ci =
    is-c v
    (is-text:is-prop-st v is-D),
    is-index:is-index v is-D),
    is-ci:is-ci v
    is-ci:is-c) v
    s-fol:(<s-init:is-format-list v is-D),
        <s-expand:is-format-1-list v is-D>)

Note: In erroneous cases during a goto statement the text component may be any proper statement, also during if-statement, the index component may be 0 immediately. The s-fol component is used immediately during edit directed get or put statements to control the expansion of the format specification.

(18) is-format-1 =
    is-format v is-id

3.3 ASSOCIATING OF IDENTIFIERS WITH MEANING

For each declaration occurring in the text of a block, a unique name n is created when the block is activated. On the one hand, this unique name n is copied into the program text of the block, wherever the declared identifier (and
the declared identifier lists in the case of a structure declaration) occurs, serving as a unique qualification of the identifier. On the other hand, under the unique name \( n \), entries into two state components, the attribute directory \( \mathcal{A}T \) and the denotation directory \( \mathcal{D}N \), are made which hold information about the meaning of the declared identifier. So it is possible, when interpreting a piece of program text, to have access to all information necessary to interpret occurring identifiers correctly.

3.3.1 CREATION OF UNIQUE NAMES

In many instances during interpretation of a PL/I program, so-called "unique names" are needed (e.g., for unique qualification of identifiers as described above). The only property required for them is that they are simple selectors which are different from each other. I.e., whenever a unique name \( n \) is needed, it must be guaranteed, that it is different from all unique names occurring elsewhere during the interpretation, except of course all occurrences of an explicit copy of \( n \) itself.

For this purpose the property of the function \( \text{elem} \) (which gives the different selectors for the components of lists) is used that it maps the integer values one-to-one into the set of simple selectors. Whenever a new unique is needed, a value of \( \text{elem} \) is used which has not been used before as unique name. There is a state component, the unique name counter \( \mathcal{U}N \) which has the only purpose to keep the information which unique names have already been used. It is an integer initialized to 0 in the initial state, increased by 1, whenever a new unique name is needed and never changed else. The process, to take the selector \( \text{elem}(\mathcal{U}N) \) as new unique name and to increase \( \mathcal{U}N \) by 1 is called "creation of a unique name" throughout the document. The unique names satisfy the predicate \( n \).

Note: The property, that the unique names are ordered by their correspondence to the integers, is never used during the interpretation, except to guarantee their distinctness. The uses of the values of \( \text{elem} \) as list selectors and as unique names are never in conflict with each other.

(19) \[ \text{is-\( \mathcal{U}N \) = \text{is-intg-val}} \]

Ref.: is-intg-val 9-3(5)

(20) \[ \text{un-name} = \text{PASS} \cdot \text{elem}(\mathcal{U}N) \]
\[ \mathcal{U}N = \mathcal{U}N + 1 \]

3.3.2 THE ATTRIBUTE DIRECTORY \( \mathcal{A}T \)

The attribute directory \( \mathcal{A}T \) contains all declarations of all block activations, which have been established, entered under the unique names associated with the declaration by the block activations. It is the text of the declarations as occurring in the program and modified by the prepass (cf.4.3) and by the unique qualification of names by the current and all (statically) surrounding block activations.

The entries into the attribute directory \( \mathcal{A}T \), which is a global state component, are made by the prologue of each block activation. They are never changed or removed. After a block activation has terminated, generally its entries, though yet contained in the attribute directory, are no more accessible since no program text qualified by the corresponding block activations is used any more. The only exception are entry, label and format constants, which could be displaced into other block activations by entry or label variables; their misuse is tested by means of the block activation name.

10 3. STATE COMPONENTS AND COMPUTATION OF THE PL/I MACHINE
**Abstract Syntax and Interpretation of PL/I**

(21) \( \text{is-at} = \)

\[
is-n((n) \iff is-decl) \land is-n(n) \land n = s\text{-builtin})
\]

Note: To guarantee correct interpretation of builtin function identifiers during the prepass, i.e. before their declarations have been entered by any block activations, their declaration `BUILTIN` is contained under the fixed selector `s\text{-builtin}` in the attribute directory `AT` of the initial state. This selector `s\text{-builtin}` is used to qualify all builtin identifiers in the program text before the prepass.

### 3.3.3 The Denotation Directory `DN`

The denotation directory `DN` contains the denotations of the declarations of all block activations, which have been established, entered under the unique names associated with the declarations by the block activations. These entries into the denotation directory `DN`, which is a global state component, are made by the prologue of each block activation. They are never changed or removed. For the access of an entry after termination of its block activation applies the same what has been mentioned for the attribute directory `AT` above.

The denotation of a declaration consists of all information necessary for correct interpretation of the declared identifier as far as it is fixed throughout the block activation and not yet contained in the text of the declaration itself. The information contained in the text of the declaration is to be found in the attribute directory `AT`. The information which changes during the block activation (e.g. values of variables) is to be found in other state components by an information chain in a unique name in the denotation directory.

The denotations of the different types of declarations are the following:

1) For a proper variable (or a parameter) the denotation is a unique name `b`, called its "aggregate name", used as selector to the aggregate directory `AG` to find the generations and by that the allocated storage and finally the values of the variable.

2) For a defined variable the denotation is its evaluated aggregate attribute.

3) For an entry constant the denotation consists of four components: The corresponding procedure body, the declared entry identifier (to be used to find the right entry point into the body when called), the block activation name (to be used for testing), and the block prefix part of `CS` (to be inherited into the procedure when called). The block activation name and the block prefix part are taken from that block activation in which the denotation was established. This is the outermost block activation of the program (i.e. both components are `D`) for an external entry constant, and the block activation in which it was declared for an internal entry constant.

4) For a file constant the denotation is a unique name `f`, called its "file name", used as selector to the file directory `FD` to find all information about the corresponding file, finally its data set in the external storage.

5) For a label constant the denotation consists of the block activation name of the block activation in which it was declared (i.e. into which a goto with that label shall load) and of an index list specifying a statement within that block activation.

6) For a format constant the denotation consists of the declared format list (to be used by means of remote formats in `GET` and `PUT` statements) and three components used for testing only, namely: the block activation name, an identifier uniquely identifying the format constant within the block activation, and a condition prefix part.

---

3. **STATE COMPONENTS AND COMPUTATION OF THE PL/I MACHINE**
7) For an attention identifier the denotation is a unique name used as selector into the attention environment directory to find its evaluated environment attribute.

8) Based, generic, builtin and programmer-named condition declarations need no denotations. The knowledge of their declarations in the attribute directory is sufficient for their interpretation.

Apart from these denotations of declarations, generic builtin functions, which are assigned to entry variables (on parameter passing), receive denotations entered under newly created unique names immediately before assignment (cf. 12.4). Such a denotation consists of the builtin identifier and the list of the data attributes of the parameters allowed on a call. This list of data attributes characterizes a single member out of the generic family denoted by the builtin identifier.

(22) is-den =
(\{<n:is-den> | | is-n(m)\})

(23) is-den =
is-n \lor is-eva \lor is-entry-den \lor is-label-den \lor is-format-den \lor is-builtin-den

(24) is-entry-den =
(s-body:is-body),
(s-id:is-id),
(s-basis-ba),
(s-bfp:is-pref-part))

Ref.: is-ba 3-7(15)
is-pref-part 3-21(60)

(25) is-label-den =
(<s-basis-n>,
<s-st-loc:is-index-list>)

(26) is-format-den =
(<s-format-list:is-format-list>,
<s-basis-n>,
<s-ident:is-id>,
<s-spp:is-pref-part>)

Ref.: is-pref-part 3-21(60)

(27) is-builtin-den =
(<s-id:is-id>,
<s-da-list:is-da-list>)

3.3.4 The Aggregate Directory AG

The aggregate directory \( \text{AG} \) contains for each proper variable and for each dummy variable the list of its generations, entered under a unique name \( b \). This unique name \( b \), called the "aggregate name" of the variable is for a proper variable to be
found as its denotation in the denotation directory DN.

A generation gen contains all information necessary to perform the access to the storage $S$ and to determine the values of a variable. It consists of three components:

1) The pointer part s-pp(gen) specifies in which parts of the storage $S$ the variable is allocated. For a newly allocated variable it is always a single pointer. The storage associated with variables that are cross-sections of arrays or sub-structures of arrays of structures may be disconnected. In this case the pointer part is a (possibly nested) list of pointers.

2) The mapping information s-mi(gen) gives the argument to the storage mapping function determining how the various parts of the variable are located in those parts of storage which are given by the pointer part.

3) The aggregate attribute s-va(gen) is relevant for referencing the variable. It specifies the aggregate structure and the data attributes of the scalar components. The scalar data attribute of a scalar component is the key for the function which determines the actual value of the value representation found in the storage by means of the pointer part and the mapping information.

Whenever a variable is allocated a generation for it is entered into the aggregate directory $AG$, whenever a variable is freed its current generation is removed. This is done for the variables of the different storage classes in different way:

For an automatic variable the list of generations consists of a single generation which is allocated by the prologue of the block activation in which it is declared and freed by the corresponding block epilogue. The information which variables are to be freed is contained in the epilogue information.

For a controlled variable the prepass of the program initializes the generation list by making an entry which gives the aggregate structure of the variable (this is used for testing if a controlled variable which has not yet any generation is passed to a controlled parameter). Each programmed allocate statement for the variable enters a generation on top of the list, each free statement removes the top generation. Always the top generation is taken as the current generation of the variable. Controlled variables are the only ones which may have more than one generation in the generation list.

For a static variable the prepass allocates a generation and enters it together with an as a two-element list into the aggregate directory. It is freed at program end.

Dummy variables are allocated and entered into the aggregate directory in the same way as static variables when they are needed. Usually they are freed when they are no more necessary.

The aggregate directory is a task local state component, i.e., each task maintains its own one. When a new task is created, it inherits only the top entries of the aggregate directory of the attaching task. When the task is terminated, all generations, except the bottom ones are freed. This applies for those generations of controlled variables which have been allocated by the task, for dummy variables created by a task, and for the static variables in the case of the main task. Also freeing of controlled variables by means of a programmed free statement is possible only for all but the bottom generations. By this mechanism it is guaranteed that each task frees those and only those generations which were allocated by the task itself. The only exception for this rule are dummy arguments passed to the task (cf. 5.1).

When generations of variables are passed to non controlled parameters (cf. 6.2.3.2), they are entered under the aggregate name of the parameter as one-element lists in order to avoid that they are freed by the aggregate name of the parameter.

3. STATE COMPONENTS AND COMPUTATION OF THE PL/I MACHINE 13
The allocation and freeing of based storage is not controlled by the aggregate directory AG, but by the s-free-set component of the task-event specification TE. The allocation and freeing of buffers for record input and output statement is controlled by the file union directory FU.

(28) \[ \text{is-ag} = \]
\[ \{ \text{<is-ag-entry-list> | | is-n(h)} \} \]

(29) \[ \text{is-ag-entry} = \]
\[ \text{is-gen} \lor \text{is-} \]
\[ \text{<s-stg:诡-c-CTL>,} \]
\[ \text{<s-eva:is-eva>} \]

Note: Aggregate directory entries which are no generations occur only as bottom entries for controlled, static and dummy variables.

(30) \[ \text{is-gen} = \]
\[ \text{<s-eva:is-eva>,} \]
\[ \text{<s-mil:is-evva>,} \]
\[ \text{<s-pat:is-vr>} \]

Note: The s-eva and s-nil components have the same aggregate structure, they may differ only in array bounds and string lengths, if the generation is produced by correspondence defining.

(31) \[ \text{is-pp} = \]
\[ \text{is-ptr-val} \lor \text{is-no-list} \]

Ref.: is-ptr-val 3-15(36)

3.4 THE STORAGE PART S

The storage part S of the PL/I machine is a model of actual machine storage. The model shows the essential properties of machine storage without, however, showing the features of a specific machine.

The storage part is defined implicitly by stating properties of and relations between the various elementary storage functions. This means that storage is not described as an object. There are many possible realizations as an object, but it is inessential as to which one is chosen. The definition leaves these possibilities open.

Storage is used to represent values of some kind. Storage parts, therefore are called value representations. Also the entire storage part S in its nature is a value representation, and satisfies the predicate is-vr.

Metavariables

\[ p \quad \text{is-ptr-val} \quad \text{pointer value} \]
3.4.1 VALUE REPRESENTATIONS AND POINTER VALUES

A value representation $vr$ satisfies the predicate

$$\text{is-vr}(vr) =$$

For any value representation the function

$$\text{size}(vr) =$$

is defined. The range of the function is characterized by the predicate

$$\text{is-size}(z) =$$

which is implementation defined:

$$\text{is-size} \cdot \text{size}(vr)$$

Parts of value representations may be selected by functions $p$ which are called pointer values, and which satisfy the predicate

$$\text{is-ptr-val} =$$

A pointer value may or may not be applicable to a certain value representation. The predicate

$$\text{is-appl}(p,z) =$$

is true if $p$ is applicable to a value representation of size $z$, false otherwise. The result of the application is a value representation. The size of the resulting value representation is determined by the function

$$\text{size}^{-1}(p) =$$

$$\text{is-appl}(p, \text{size}^{-1}(vr)) \Rightarrow \text{is-vrep}(vr) \land \text{size}^{-1}(vr) = \text{size}^{-1}(p)$$

Provided applicability, pointer values may be composed:

$$\text{is-appl}(p^{-1}, \text{size}^{-1}(p^{-2})) \Rightarrow \text{is-ptr-val}(p^{-1}p^{-2})$$

$$\text{is-appl}(p^{-1}, \text{size}^{-1}(p^{-2})) \land \text{is-appl}(p^{-1}p^{-2}, z) \Rightarrow \text{is-appl}(p^{-1}p^{-2}, z)$$
Two pointer values may select overlapping parts of value representations. If the selected parts do not overlap, the predicate

\( \text{is-indep}(p-1, p-2) = \)

is true, false otherwise. The independence relation is symmetric. Further, a part is never independent of the whole, and parts of independent parts are again independent. These properties are expressed by:

\( \neg \text{is-indep}(p, p) \)

\( \text{is-indep}(p-1, p-2) \Rightarrow \text{is-indep}(p-2, p-1) \)

\( \text{is-ptr-val}(p-1, p-2) \Rightarrow \neg \text{is-indep}(p-1, p-2, p-2) \)

\( \text{is-indep}(p-1, p-2) \land \text{is-ptr-val}(p, p-2) \Rightarrow \text{is-indep}(p-1, p, p-2) \)

3.4.2 ELEMENTARY ASSIGNMENT

A part of a value representation \( vr \), identified by a pointer value \( p \), can be replaced by the elementary assignment function:

\( \text{el-ass}(vr-1, p, vr) = \)

The result is a value representation which differs from \( vr \) only in that its \( p \)-part is now \( vr-1 \). This is expressed by:

\( \text{size}(\text{el-ass}(vr-1, p, vr)) = \text{size}(vr) \)

\( \text{size}(vr-1) = \text{size}(p) \Rightarrow p \cdot \text{el-ass}(vr-1, p, vr) = vr-1 \)

\( \text{is-indep}(p-1, p-2) \land \text{size}(vr-1) = \text{size}(p-2) \Rightarrow p-1 \cdot \text{el-ass}(vr-1, p-2, vr) = p-1(vr) \)

with the assumption that \( \text{is-applic}(p, \text{size}(vr)) \), \( \text{is-applic}(p-1, \text{size}(vr)) \), and \( \text{is-applic}(p-2, \text{size}(vr)) \). Assignment may be super-imposed:

\( \text{is-applic}(p-1, \text{size}(p-2)) \land \text{is-applic}(p-2, \text{size}(vr)) \land \text{size}(vr-1) = \text{size}(p-1) \Rightarrow \)

\( \text{el-ass}(vr-1, p-1, p-2, vr) = \text{el-ass}(\text{el-ass}(vr-1, p-1, p-2(vr)), p-2, vr) \)

3.4.3 ELEMENTARY ALLOCATION

Those parts of a value representation (the main storage or an area) which have been reserved by allocation of variables, are identified in the allocation state. The allocation state is a set of pointers identifying the reserved parts, it is obtained by the function

\( \text{alloc-state}(vr) = \)

applied to the value representation.
(54) \( \text{is-ptr-val-set-alloc-state(vr)} \)

On allocation the pointer value identifying the part to be reserved is added to the allocation state by using the elementary allocation function.

(55) \( \text{el-alloc(p, vr, type)} = \)

The result of the elementary allocation function is a value representation which differs from \( vr \) in that its allocation state is amended. The allocation is possible if the implementation defined predicate

(56) \( \text{is-free-space(p, vr, type)} = \)

is true, where \( type \) indicates the type of the allocation. The predicate guarantees the applicability of \( p \) to \( vr \), and the independence of the \( p \)-part of all other parts identified in the allocation state.

(57) \( \text{is-free-space(p, vr, type)} \Rightarrow \text{is-applic(p, size(vr))} \& \)

\( (p-1 \in \text{alloc-state(vr)} \Rightarrow \text{is-indep(p-1, p)}) \)

The properties of the allocation function are expressed by:

(58) \( \text{size•el-alloc(p, vr, type)} = \text{size(vr)} \)

(59) \( \text{is-free-space(p, vr, type)} \Rightarrow \)

\( \text{alloc-state•el-alloc(p, vr, type)} = (\text{alloc-state(vr)} \cup \{p\}) \)

(60) \( \text{is-free-space(p, vr, type)} \& p-1 \in \text{alloc-state(vr)} \Rightarrow \)

\( p-1\cdot\text{el-alloc(p, vr, type)} = p-1(vr) \)

(61) \( p \in \text{alloc-state(vr)} \& \text{size(vr-1)} = \text{size-1(p)} \Rightarrow \)

\( \text{alloc-state•el-ass(vr-1, p, vr)} = \text{alloc-state(vr)} \)

3.4.4 ELEMENTARY FREEING

The allocation state can be reduced by using the elementary freeing function

(62) \( \text{el-free(ps, vr)} = \)

The result is a value representation which differs from \( vr \) in that its allocation state is reduced by the set of pointer values \( ps \). The properties of the elementary freeing function are expressed by:

(63) \( \text{size•el-free(ps, vr)} = \text{size(vr)} \)

(64) \( \text{alloc-state•el-free(ps, vr)} = (\text{alloc-state(vr)} - ps) \)

(65) \( p-1 \in \text{alloc-state•el-free(ps, vr)} \Rightarrow p-1\cdot\text{el-free(ps, vr)} = p-1(vr) \)
3.5 STATE COMPONENTS FOR ATTENTIONS AND CONDITIONS

The attention directory \( \text{AN} \), the attention enabling state \( \text{EN} \) and the attention environment directory are solely used to describe the enabling and disabling mechanism of attentions in the various tasks and to stack the attention information.

The condition status \( \text{CS} \) contains the enabling information for prefix controlled conditions. The other two parts of \( \text{CS} \), holding information for interpreting on-units and condition builtin functions, are used for all conditions.

3.5.1 ATTENTION DIRECTORY \( \text{AN} \)

This global directory holds the information relevant for executing access statements and asynchronous interrupts. The updating of this directory is done by enable and disable statements.

The attention directory \( \text{AN} \) contains for each attention identification (a selector satisfying the implementation defined predicate \( \text{is-ident} \)) four components: the first contains the stack of attention information, which is updated from outside the PL/I machine; the second specifies the task for which the attention is enabled; the third characterizes the enabling mode; the last collects the name of the tasks which have this attention associated.

(66) \( \text{is-an} = \)  
\[
\{ \langle \text{id:is-attention} \rangle | \text{| | is-ident(ident)} \}  
\]

(67) \( \text{is-attention} = \)  
\[
\langle \text{s-info:is-info-list} \rangle,  
\langle \text{s-task:is-tn} \rangle,  
\langle \text{s-spcc:is-ACC v is-ASYN v is-ACC-1} \rangle,  
\langle \text{s-asscc:is-tr-set} \rangle  
\]

Ref.: is-tn 3-4(6)

Note: Whenever an asynchronous interrupt is interpreted the enabling mode is changed to ACC-1, to differentiate from normal ACC.

(68) \( \text{is-attn-occ} = \)  
\[
\langle \text{s-ident:is-ident} \rangle,  
\langle \text{s-info:is-info} \rangle  
\]

(69) \( \text{is-ident} = \)  

Note: Implementation defined predicate, characterizing selectors used as attention identifications.

(70) \( \text{is-info} = \)  
\[
\langle \text{s-strct:is-eva} \rangle,  
\langle \text{s-op:is-op} \rangle  
\]

Ref.: is-op 9-9(34)
3.5.2 ATTENTION ENABLING STATE EN

This task-local state component holds the enabling information of all attentions in a task. One component lists the attentions enabled in the task, another one the attentions associated with the task. The component s-wait-list enumerates the attentions which are only associated with the task but are not specified with an event, so that the task must wait until all these attentions are enabled for the task. The last component is a set which specifies for each event name the corresponding attentions, which are only associated with the task.

\[
\text{\{(s-enab-list:is-eattn-cond-list),} \\
\text{\langle s-assoc-list:is-eattn-cond-list\rangle,} \\
\text{\langle s-wait-list:is-eattn-cond-list\rangle,} \\
\text{\{<tn:is-eattn-cond-list> | / is-tn(tn)\}}\]
\]

Ref.: is-tn 3-4(6)

\[
\text{\{\langle s-eattn-cond =} \\
\text{\langle s-ident:is-ident\rangle,} \\
\text{\langle s-spec:is-ACC v is-ASYN v is-O\rangle,} \\
\text{\langle s-tn:is-tn-set v is-O\rangle\}}\]
\]

Ref.: is-tn 3-4(6)

3.5.3 ATTENTION ENVIRONMENT DIRECTORIES EV

This attention directory contains for each unique name an evaluated environment which is used to create the attention identification.

\[
\text{\{(s-ev =} \\
\text{\langle b:is-ea > | | is-2(b))\}}\]
\]

Ref.: is-ea 11-21(58)

3.5.4 CONDITION SELECTORS

For the access to components of major subparts of CS condition selectors are generated by the specific function cond-sel which returns a selector value. The auxiliary function sel is used which generates a selector value from a name list. The auxiliary function cond-ic-sel generates selectors from ic-condition names.

The condition is an evaluated condition which satisfies the predicate is-cond. The evaluated condition differs from the condition described by the abstract syntax only in three points: the check condition which is a reference now; the i/o condition; the attention condition which mainly consists of the attention identification.
(74) \[ \text{cond-sel} \text{(cond, at)} = \]
\[ \text{is-CONV} \text{(cond)} \rightarrow \text{s-conv} \]
\[ \text{is-FCFL} \text{(cond)} \rightarrow \text{s-fofl} \]
\[ \text{is-FLFL} \text{(cond)} \rightarrow \text{s-ofll} \]
\[ \text{is-SIZE} \text{(cond)} \rightarrow \text{s-size} \]
\[ \text{is-STRG} \text{(cond)} \rightarrow \text{s-strg} \]
\[ \text{is-STRZ} \text{(cond)} \rightarrow \text{s-strz} \]
\[ \text{is-UFL} \text{(cond)} \rightarrow \text{s-ufl} \]
\[ \text{is-ZDIV} \text{(cond)} \rightarrow \text{s-zdiv} \]
\[ \text{is-AREA} \text{(cond)} \rightarrow \text{s-area} \]
\[ \text{is-ERROR} \text{(cond)} \rightarrow \text{s-error} \]
\[ \text{is-FINISH} \text{(cond)} \rightarrow \text{s-finish} \]
\[ \text{is-check} \text{(cond)} \ \& \ \text{is-EXT-scope} \left( (\text{s-ref} \text{(cond)}) \text{(at)} \right) \rightarrow \]
\[ \text{is-check} \text{(cond)} \rightarrow \left( \text{s-ref} \text{(cond)} \right) \text{(sel-s-id-list-s-ref} \text{(cond)}) \text{s-check} \]
\[ \text{is-check} \text{(cond)} \rightarrow \left( \text{s-ref} \text{(cond)} \right) \text{(s-check} \text{(cond)}) \text{s-check} \]
\[ \text{is-f-cond} \text{(cond)} \rightarrow \left( \text{s-f} \text{(cond)} \right) \text{(cond-io-sel} \text{(cond)} \text{(cond)}) \]
\[ \text{is-progr-named-cond} \text{(cond)} \rightarrow \left( \text{s-name} \text{(cond)} \right) \text{s-progr-named-cond} \]
\[ \text{is-eattn-cond} \text{(cond)} \rightarrow \left( \text{s-ident} \text{(cond)} \right) \text{s-attn} \]
\[ \text{for:} \text{is-cond} \text{(cond), is-att} \text{(at)} \]

Ref.: \text{is-at 3-11(21)}

(75) \[ \text{cond-io-sel} \text{(iocond)} = \]
\[ \text{is-BOV} \text{(iocond)} \rightarrow \text{s-bov} \]
\[ \text{is-BOV} \text{(iocond)} \rightarrow \text{s-bov} \]
\[ \text{is-ENDF} \text{(iocond)} \rightarrow \text{s-endf} \]
\[ \text{is-ENDP} \text{(iocond)} \rightarrow \text{s-endp} \]
\[ \text{is-KEY} \text{(iocond)} \rightarrow \text{s-key} \]
\[ \text{is-NAME} \text{(iocond)} \rightarrow \text{s-name} \]
\[ \text{is-REC} \text{(iocond)} \rightarrow \text{s-rec} \]
\[ \text{is-TMT} \text{(iocond)} \rightarrow \text{s-tmt} \]
\[ \text{is-UNDF} \text{(iocond)} \rightarrow \text{s-undf} \]
\[ \text{is-PEND} \text{(iocond)} \rightarrow \text{s-pend} \]
\[ \text{for: is-ic-cond} \text{(iocond)} \]
ABSTRACT SYNTAX AND INTERPRETATION OF PL/I

3.5.5 THE CONDITION STATE CS

The condition state CS is composed of four major subparts. The block-prefix-part selected by s-bpp contains the enabling information for all conditions which may be prefixed to a block. The statement-prefix-part selected by s-spp contains all prefixes valid during interpretation of a statement.

The condition-action-part selected by s-cap contains the actions which have to be performed if an enabled condition is raised and interpreted.

The condition-built-in-function-part contains components for specifying the values returned on references to these functions and some auxiliary parts for obtaining these values.

(76) \( \text{is-econd} = \)
\[ \text{is-ccnd} \land \text{-is-check-ccnd} \land \text{-is-attn-ccnd} \land \text{-is-ic-ccnd} \lor \text{is-check} \lor \text{is-eattn-ccnd} \lor \text{is-f-ccnd} \]

(77) \( \text{is-check} = \)
\[ \text{is-ref} \]

(78) \( \text{is-f-ccnd} = \)
\[ (<s-f-ccnd:is-s-st-prt \lor \text{is-n}>, <s-cond:is-ic-ccnd>, <s-chif:is-chif>) \]

Ref.: \( \text{is-s-st-prt 3-24(50)} \)

3. STATE COMPONENTS AND COMPUTATION OF THE PL/I MACHINE 21
ABSTRACT SYNTAX AND INTERPRETATION OF PL/I

3. STATE COMPONENTS AND COMPUTATION OF THE PL/I MACHINE

(82) \texttt{is-cond-act} =

\[
\begin{align*}
\langle & \langle \texttt{s-cond:is-econd}\rangle, \\
& \langle \texttt{s-bpp:is-pref-part}\rangle, \\
& \langle \texttt{s-snap:is-opt}\rangle, \\
& \langle \texttt{s-on-unit:is-st} \land \texttt{is-SYSTEM}\rangle \rangle
\end{align*}
\]

(83) \texttt{is-cond-bif-part} =

\[
\begin{align*}
\langle & \langle \texttt{s-conloc:is-id} \land \texttt{is-O}\rangle, \\
& \langle \texttt{s-oncode:is-intg-val} \land \texttt{is-O}\rangle, \\
& \langle \texttt{s-oncount:is-intg-val} \land \texttt{is-O}\rangle, \\
& \langle \texttt{s-onfile:is-id} \land \texttt{is-O}\rangle, \\
& \langle \texttt{s-onkey:is-char-val-list} \land \texttt{is-O}\rangle, \\
& \langle \texttt{s-datafield:is-char-val-list} \land \texttt{is-O}\rangle, \\
& \langle \texttt{s-cnchar:is-intg-val} \land \texttt{is-O}\rangle, \\
& \langle \texttt{s-onsource:is-gen} \land \texttt{is-ps-gen} \land \texttt{is-O}\rangle, \\
& \langle \texttt{s-cnident:} \langle \texttt{s-gen:is-gen}\rangle, \\
& \langle \texttt{s-input:is-T} \land \texttt{is-T}\rangle \land \texttt{is-O}\rangle, \\
& \langle \texttt{s-onattn:is-ptr-val} \land \texttt{is-O}\rangle, \\
& \langle \texttt{s-type:is-SIGNAL} \land \texttt{is-O}\rangle, \\
& \langle \texttt{s-abn-ret:is-abn-ret}\rangle, \\
& \langle \texttt{s-cond:is-eattn-cond} \land \texttt{is-O}\rangle \rangle
\end{align*}
\]

Ref.: \texttt{is-intg-val} 9-3(5)
\texttt{is-char-val} 9-3(6)
\texttt{is-gen} 3-14(30)
\texttt{is-ps-gen} 12-67(169)
\texttt{is-ptr-val} 3-15(36)

(84) \texttt{is-abn-ret} =

\[
\begin{align*}
\langle & \langle \texttt{s-en:is-n}\rangle \rangle \\
& \langle \texttt{s-u:is-n}\rangle, \\
& \langle \texttt{s-cond:is-BOV} \land \texttt{is-EOV}\rangle \land \texttt{is-O}\rangle
\end{align*}
\]

(85) \texttt{is-chif} =

\[
\begin{align*}
\langle & \langle \texttt{s-oncode:is-intg-val} \land \texttt{is-O}\rangle, \\
& \langle \texttt{s-oncount:is-intg-val} \land \texttt{is-O}\rangle, \\
& \langle \texttt{s-onfile:is-id} \land \texttt{is-O}\rangle, \\
& \langle \texttt{s-onkey:is-char-val-list} \land \texttt{is-O}\rangle, \\
& \langle \texttt{s-datafield:is-char-val-list} \land \texttt{is-O}\rangle, \\
& \langle \texttt{s-cnchar:is-intg-val} \land \texttt{is-O}\rangle, \\
& \langle \texttt{s-onsource:is-gen} \land \texttt{is-ps-gen} \land \texttt{is-O}\rangle, \\
& \langle \texttt{s-cnident:} \langle \texttt{s-gen:is-gen}\rangle, \\
& \langle \texttt{s-input:is-T} \land \texttt{is-T}\rangle \land \texttt{is-O}\rangle, \\
& \langle \texttt{s-onattn:is-ptr-val} \land \texttt{is-O}\rangle, \\
& \langle \texttt{s-type:is-SIGNAL} \land \texttt{is-O}\rangle, \\
& \langle \texttt{s-abn-ret:is-abn-ret}\rangle, \\
& \langle \texttt{s-cond:is-econd} \land \texttt{is-O}\rangle \rangle
\end{align*}
\]

Ref.: \texttt{is-intg-val} 9-3(5)
\texttt{is-char-val} 9-3(6)
\texttt{is-gen} 3-14(30)
\texttt{is-ps-gen} 12-67(169)
\texttt{is-ptr-val} 3-15(36)
3.6. INPUT AND OUTPUT

The state parts dealing with input and output are the task local file directories (for the current task it is \( FD \)), and the program local file union directory \( FU \), the external storage \( ES \), and the message part \( M \).

The message part \( M \) does not store information about files but serves for the interpretation of the display statement and receives the comments of standard system actions for certain conditions.

A file is the total information which is accessible by a file value through the state components \( AT \), \( IN \), \( FD \), \( FU \), and \( ES \). File values are unique names (is-n) which are appended to references of file constants when the block is entered to which the declaration of the file constant is local. Assignment to a scalar file variable effectively consists in a file value assignment. In particular, a file value is given access to the declaration \( n(AT) \) and to the denotation \( n(FD) \) of the file constant. The latter is said to be a file name \( u \). All file names of a program are established by the prepass which evaluates all (internal or external) declarations of file constants. The result of this evaluation is deposited as an entry in the \( FD \) of the major task under a particular file name, which in turn is a unique name (is-n).

Two references to scalar file variables or file constants compare equal iff they refer to the same file name. Equality of file names doesn't imply equality of file values.

After proper assignment to a file variable, the legality of further references is restricted by the scope of the variable declaration but it doesn't depend on the scope of the assignee.

In general, a file value does refer to a file name and hence to an entry in \( FD \), but it will not refer to an entry in \( FU \) and to a data set in \( ES \), i.e., the file value doesn't refer to a file. The existence of a file depends on opening and closing. Proper opening establishes a link between \( FD \) and \( ES \) by making a new entry in \( FD \) under an new file union name \( u \) (is-n(\( FU \))). The entry in \( FD \) is deleted if proper closing takes place.

3.6.1 THE FILE DIRECTORY \( FD \)

There is one file directory \( FD \) for every task, in particular there is one \( FD \) for the current task. The \( FD \) for a particular task is established when the task is attached. It is empty when the main task is attached, but it is essentially a copy of the \( FD \) of the attaching task in all other cases. However, in this copy all components s-own and s-tmt are deleted. This is because owning of a file and the occurrence of transmission errors on stream files are strictly task local properties.

Proper opening of a file with a certain file name \( f \) changes the non-empty entry \( f(FD) \) by adding a part \( s-fd-st-f(FD) \) which describes the status of the file. This status, called the fd-status, consists of a new file union case \( u \) (linking \( f \) with \( u(FU) \)), and of the components s-own and s-tmt.

Proper closing deletes the part \( s-fd-st-f(FD) \) but leaves the non-empty part \( f(FD) \).

Termination of the particular task causes the deletion of the whole \( FD \).

If the above cases would be a complete enumeration of all instances where file directories are changed, one might conclude that after the prepass every \( FD \) has the same number of entries selected by the same file names all belonging to the class of unique names.

There are two types of files: common files and standard system print files. Only common files have been considered so far. Among all file names which may be related with common files, there is no or one file name which may (but need not) be related with a standard system print file. Whenever a standard system print file exists in a task, it is related with the system file name s-st-prt. If
a standard system print file has been opened by a copy option of a get statement or by a standard system action of a check on-condition, the file is exclusively related with the file name s-st-prt, i.e., s-fd-st*s-st-prt(FD) is the file status. As long as the exclusive relation holds it is an error to relate the file name f with this standard system print file. If a standard system print file has been opened with the file name f, the file is related with f and s-st-prt. The status of this standard system print file is again s-fd-st*s-st-prt(FD), and s-fd-st*f(FD) is the degenerated status ST-PRT.

(86) is-fd =
     (s-st-prt:is-standard-fd-el,
      f:is-common-fd-el | | is-n(f))

(87) is-common-fd-el =
     (s-fd-st:is-fd-status \ is-ST-PRT \ is-\Omega,
      s-attr:is-file-attr-set,
      s-ea:is-ea,
      s-id:is-id)

Ref.: is-ea 11-21(58)

Note: The file attributes s-attr*FD and the identifier s-id*FD are the
attributes and the identifier of the file constant which has been given the
file name f in the prepass. The evaluated environment attribute (selected
by s-ea) has been evaluated in the prepass.

(88) is-fd-status =
     (s-fu:is-n,
      s-own:is-opt,
      s-tmt:is-opt)

Note: The component s-fu is the file union name which links the FD entry with an
entry in FU.

(89) is-standard-fd-el =
     (s-fd-st:is-fd-status \ is-\Omega)

Note: As opposed to an entry for a common file, the entry for a standard system
print file consists only from a status, since the components s-attr, s-ea and
s-id are determined from defaults.

(90) is-s-st-prt(f) =
     f = s-st-prt

Note: The system file name s-st-prt is the only file name which is not a program
file name, hence is-s-st-prt(f) \= is-n(f).

3.6.2 THE FILE UNION DIRECTORY FU

The file union directory FU is program local. New entries are made at proper
opening of a file; the whole entry for a file is deleted when the file is closed.
There are as many entries as there exist files, i.e., file union names, regardless
of the type of the file, and independent of the sharing of core and the same file over tasks.

The components s-p and s-f, and the components s-1sz and s-psz (the latter two are optional) remain constant as long as the file exists.

In addition, there may be entries which serve for the interpretation of string options of get and put statements. These entries are selected by unique names, too. They have no connection with file names and no connection with file directories.

(91) \( \text{is-fu} = \)
\( (\{s:\text{is-fu-el} \mid \text{is-n}(s)\}) \)

(92) \( \text{is-fu-el} = \)
\( \text{is-fu-file} \land \text{is-fu-string} \)

(93) \( \text{is-fu-file}(\text{fu-el}) = \)
\( \text{is-fu-file-1}(\text{fu-el}) \land \text{is-fu-file-2}(\text{fu-el};\text{s-p},\text{s-f},\text{s-st},\text{s-csa}*\text{s-p}(\text{fu-el})) \)

Note: The predicate \( \text{is-fu-file-2} \) gives various correlations between components of an \( \text{FU} \) entry and the complete set of attributes.

(94) \( \text{is-fu-file-1} = \)
\( \langle s-p:is-p \rangle, \)
\( \langle s-f:is-s-st-prt \land is-n \rangle, \)
\( \langle s-st:is-fu-status \land is-w \rangle, \)
\( \langle s-volno:is-intg-val \land is-o \rangle, \)
\( \langle s-ccl:is-intg-val \land is-o \rangle, \)
\( \langle s-count:is-bintg-op \land is-o \rangle, \)
\( \langle s-1sz:is-intg-val \land is-o \rangle, \)
\( \langle s-line:is-bintg-op \land is-o \rangle, \)
\( \langle s-psz:is-intg-val \land is-o \rangle, \)
\( \langle s-buf:is-buf \land is-o \rangle, \)
\( \langle s-io-ev:is-n-set \land is-o \rangle, \)
\( \{tn:is-char-val-list-set \mid \text{is-tn}(tn)\} \)

Ref.: \( \text{is-intg-val} \text{ 9-3}(5) \)
\( \text{is-bintg-op} \text{ 9-10}(42) \)
\( \text{is-char-val} \text{ 9-3}(6) \)
\( \text{is-tn} \text{ 3-4}(6) \)

Note: The component \( s-f \) is the file name which is the basis for I/O condition raising.

(95) \( \text{is-p} = \)
\( \langle s-csa:is-csa \rangle, \)
\( \langle s-ea:is-ea \rangle, \)
\( \langle s-title:is-id \rangle \)

Ref.: \( \text{is-csa} \text{ 11-4}(8) \)
\( \text{is-ea} \text{ 11-21}(58) \)
Note: The components s-csa, s-ea, s-title of the file parameter p are related with the components s-attr, s-ea, s-id of the ID entries. In general, however, the components are no copies of each other. All components are required for data set mapping; the components s-ea and s-title provide a linkage to ES.

(96) is-fu-status =

is-SW-Bov v is-EOV v is-EOV v is-SW v is-ENDF

Note: The fu-status characterizes the transitions of a file with respect to data set switching, begin and end of volume processing, and the ultimate end status.

(97) is-buf =

(<s-o:is-ptr-val>,
 <s-area-p:is-ptr-val v is-U>,
 <s-key:is-char-val-list v is-U>)

Ref.: is-ptr-val 3-15(36)
      is-char-val 9-3(6)

Note: Such an entry exists if for a buffered file a buffer has been allocated. Allocation is in main storage (the component s-o is the pointer) or in an area (the component s-area-p is the area pointer, and s-o is the offset).

(98) is-fu-file-2(fu-el,cSa) =

(is-intg-val•s-vidno(fu-el) ~ KEY & cSa) &
(is-intg-val•s-cl(fu-el) = CST « cSa & EST « cSa) &
(is-bintg-op•s-count(fu-el) = CST « cSa & EST « cSa) &
(is-intg-val•s-lsz(fu-el) = (CST « cSa & EST « cSa) & OUT « cSa) &
(is-bintg-op•s-line(fu-el) = PRT « cSa) &
(is-intg-val•s-psz(fu-el) = PRT « cSa) & (is-buf•s-buf(fu-el) = BUF « cSa) &
(is-char-val•list•s-key•s-buf(fu-el) = KEY « cSa & OUT « cSa) &
(is-n-set•s-lo-ev(fu-el) = ELC « cSa & BUF « cSa & TRA & cSa) &
((3tn)(is-tn(tn) & is-char-val•list•set•tn(fu-el)) = ELC « cSa)

for:is-csa(cSa)

Ref.: is-intg-val 9-3(5)
      is-bintg-op 9-10(42)
      is-char-val 9-3(6)
      is-tn 3-4(6)
      is-csa 11-4(8)

(99) is-fu-string =

(<s-g:is-gen v is-ps-gen>,
 <s-h:is-intg-val>,
 <s-lo:is-intg-val v is-U>)

Ref.: is-gen 3-14(30)
      is-ps-gen 12-67(169)
      is-intg-val 9-3(5)
3.6.3 THE EXTERNAL STORAGE IS

The external storage IS is program local. It is initialized by the initial call, i.e., its contents are initially program dependent. The IS has two immediate components s-ds-sh and s-ds-dir. The first is set by the initial call only and remains constant for the whole program. It is the information on data set sharing. The second component is the data set directory which is subject to various changes, either by the proper computation or by environmental influences. Such influences are data transmission errors, tele-processing data sets, data set switching (multiple volume handling), and any kind of system or operator interference.

(100) is-es =

\{(s-ds-sh:is-ds-sh),
  (s-ds-dir:is-ds-dir)\}

(101) is-ds-dir =

\{(k:is-tmt-ds) \mid \| is-ds-n(e)\}\)

Note: Environmental influences must not delete whole entries from the data set directory.

(102) is-tmt-ds =

\{(s-ds:is-ds),
  (s-tmt:is-TMT \wedge is-D)\}

Note: The transmission error component s-tmt is set exclusively by the environment. An empty component may be set to TMT which will be interpreted as a transmission error.

(103) is-ds =

Note: This predicate is implementation defined. It is related with other entities by several axioms (see below and chapter 11.7). A data set satisfying this predicate should be conceived of containing data and descriptive information about data. There is no limitation for the environmental influences on a data set. Data sets are elementary objects.

(104) is-ds-sh =

Note: This predicate is implementation defined.
(105) \[ \text{ds-sel}(\text{id}, \text{ea}, \text{ds-sh}) = \]
\[
\text{for:is-id(id),is-ea(ea),is-ds-sh(ds-sh)}
\]
\[
\text{Ref.: is-ea 11-21(58)}
\]

Note: This function is implementation defined. The function is used to map a title (taken from s-title of an FU entry), an evaluated environment attribute (taken from s-ea of the same FU entry), and the s-ds-sh component of ES into a data set name \( s \) which satisfies \( \text{is-ds-n}(s) \).

(106) \[ \text{is-ds-n}(s) \land \text{is-tmt-ds}\cdot s\cdot \text{ds-dir}(es) \Rightarrow (\exists \text{id}, \text{ea})(s = \text{ds-sel}(\text{id}, \text{ea}, \text{s-ds-sh}(es))) \]
\[
\text{for:is-id(id),is-ea(ea),is-es(es)}
\]
\[
\text{Ref.: is-ea 11-21(58)}
\]

Note: For any external storage, each entry actually present in the data set directory must be accessible given a title id and an evaluated environment attribute ea.

3.6.4 THE MESSAGE PART \( \text{ms} \)

The message part \( \text{ms} \) is program local. It receives messages put out by display statements and reply messages inserted by environmental influences. Comments put out as part of the standard system action of certain on-conditions are collected in the s-comment component.

(107) \[ \text{is-m} = \]
\[
(\langle \text{s-display}:\text{is-named-message-list} \rangle, \langle \text{s-reply}:\text{is-named-message-list} \rangle, \langle \text{s-comment}:\text{is-comment-list} \rangle)
\]

Note: All three components are incremented on their tails only.

(108) \[ \text{is-named-message} = \]
\[
(\langle \text{s-name}:\text{is-n} \rangle, \langle \text{s-message}:\text{is-char-val-list} \rangle)
\]
\[
\text{Ref.: is-char-val 9-3(6)}
\]

Note: The name is used to relate a reply message uniquely with a display message.

(109) \[ \text{is-comment} = \]
\[
\text{Note: This predicate is implementation defined (see chapter 10).} \]
3.7 The Computation of the PL/I Machine

The PL/I machine describes the interpretation of a PL/I program \( t \) by defining the set of possible computations resulting from the program. A computation is a sequence of states

\[ \xi(0), \xi(1), \xi(2), \ldots \]

satisfying the following two conditions:

1) \( \xi(0) \) is an initial state corresponding to \( t \), as given by the function initial-state (cf. 4.1):

\[ (\exists \text{stg,es,time,dto,etv,cell,gen}) \quad (\xi(0) = \text{initial-state(stg,es,time,dto,etv,cell,gen}) \]

2) Any two adjacent states \( \xi(i), \xi(i+1) \) in the computation must represent a valid step. The validity of a step is formally defined by the predicate is-step, i.e., any two steps \( \xi(i), \xi(i+1) \) in the computation must satisfy the condition

\[ \text{is-step}(\xi(i),\xi(i+1)) \]

A computation is "successful" if it is finite:

\[ \xi(0), \xi(1), \ldots, \xi(n) \]

and if its end state \( \xi(n) \) satisfies the condition

\[ \text{is-non-pa}(\xi(n)) \]

A step represented by a pair of states \( \xi(i) \) and \( \xi(i+1) \) is described by considering two intermediate states and, correspondingly, three intermediate steps. The first intermediate step, the computation step, corresponds to the computation of the machine as it is controlled by the instruction selected for execution. The second intermediate step, the environment step, allows for changes in the state effected from outside the machine, i.e., by replies, attentions, changes in the external storage not caused by the PL/I program being executed, etc. The third intermediate step (if it is not the null step) comprises actions which interrupt the normal program execution. These interrupts are caused by changes in the state that occurred in the environment step.

Metavariables

\begin{align*}
\text{an} & \quad \text{is-an} & \text{attention directory} \\
\text{ds-dir} & \quad \text{is-ds-dir} & \text{data set directory} \\
\text{ds-n} & \quad \text{is-ds-n} & \text{data set name} \\
\text{ident} & \quad \text{is-ident} & \text{attention identification} \\
\text{infol} & \quad \text{is-infolist} & \text{information stack} \\
\text{p} & \quad \text{is-ptr-val} & \text{pointer value} \\
\text{reply} & \quad \text{is-named-message} & \text{reply message} \\
\text{s} & \quad \text{is-vr} & \text{storage} \\
\text{time} & \quad \text{is-intg-val} & \text{value of time part} \\
\text{tn} & \quad \text{is-tn} & \text{task name}
\end{align*}
3.7.1 THE COMPUTATION STEP

A machine state in its parallel actions part \( PA \) contains under individual selectors the states of the individual parallel tasks or I/O events (cf. 3.1.1). Based on the priorities of the different parallel actions one of the active parallel actions is selected for execution, i.e., one which is not in the wait state. The selection is described by the implementation defined function \( pr\text{-}sched \), which returns the selector of this parallel action. This selector is entered in the \( s\text{-}tn \) component of the state.

An instruction located at one of the terminal nodes of the control part of the selected parallel action is selected for execution. The set of instructions at terminal nodes of a control part is defined by the function \( term\text{-}node \), which returns the set of selectors leading to these instructions. The function \( compute \), applied to the state and the selector identifying the instruction to be executed, gives the state which is the result of the computation step.

The function \( compute \) is defined in 1.5.3.

\[
\begin{align*}
(110) \quad &is\text{-}step(\xi-1,\xi-2) = \\
&(3\xi-3,\xi-4)(is\text{-}comp\text{-}step(\xi-1,\xi-3) \land is\text{-}env\text{-}step(\xi-3,\xi-4) \land \\
&\xi-2 = prep\text{-}interrupt(\xi-3,\xi-4))
\end{align*}
\]

3.3.1 STATE COMPONENTS AND COMPUTATION OF THE PL/I MACHINE

\[
\begin{align*}
(111) \quad &is\text{-}comp\text{-}step(\xi-1,\xi-2) = \\
&(3\xi) (is\text{-}active\text{-}pa(\xi-1)) \rightarrow \\
&(3\xi) (s \in term\text{-}nodes\text{-}ctn\text{-}pri\text{-}s-pa(\xi-1) \land \xi-2 = compute(\mu(\xi-1; \\
&s\text{-}tn\text{-}ctn\text{-}pri\text{-}s-pa)))
\end{align*}
\]

where:
\[
\begin{align*}
\text{tn\text{-}pri}_4 &= pr\text{-}sched(\xi-1)
\end{align*}
\]

\[
(112) \quad pri\text{-}sched(\xi) =
\]

Note: This implementation defined function returns the selector of an active task or I/O event.

\[
(113) \quad is\text{-}active\text{-}pa(pri\text{-}sched(\xi),\xi)
\]

\[
(114) \quad is\text{-}active\text{-}pa(\text{tn},\xi) =
\]

\[
\begin{align*}
is\text{-}tn(\text{tn}) &\land -is\text{-}0\text{-}s\text{-}ctn\text{-}s\text{-}pa(\xi) \land is\text{-}0\text{-}s\text{-}wait\text{-}s\text{-}ctn\text{-}s\text{-}pa(\xi)
\end{align*}
\]

Ref.: is-tn 3-8(6)
3.7.2 THE ENVIRONMENT STEP

The predicate \( \text{is-env-step}(\xi_1, \xi_2) \) tests whether \( \xi_2 \) is an admissible modification of \( \xi_1 \). Modifications by the environment may concern the internal storage, the external storage, the time part, the message part, and the attention directory.

\[ \text{is-env-step}(\xi_1, \xi_2) = \]
\[ \xi_2 = \mu(\xi_1; <s-s: s-e(s(\xi_2))>, <s-ds: ds-e(s(\xi_2))>, <s-tm: tm-e(s(\xi_2))>, <s-rep: rep-e(s(\xi_2))>, <s-an: an-e(s(\xi_2))>) \& \text{is-step}(s-s(\xi_1), s-s(\xi_2)) \&
\]
\[ \text{is-ds-dir-step}(s-ds-dir(s(\xi_1)), s-ds-dir(s(\xi_2))) \& \text{is-time-step}(s-tm(s(\xi_1)), s-tm(s(\xi_2))) \&
\]
\[ \text{is-reply-step}(s-rep(s(\xi_1)), s-rep(s(\xi_2))) \& \text{is-an-step}(s-an(s(\xi_1)), s-an(s(\xi_2))) \]

\[ \text{(115)} \]

\[ \text{is-s-step}(s_1, s_2) = \]
\[ \text{is-vr}(s_2) \& \text{size}(s_1) = \text{size}(s_2) \&
\]
\[ (\forall p) (p \in \text{alloc-state}(s_1) \equiv p \in \text{alloc-state}(s_2) \& p(s_1) = p(s_2)) \]

Ref.: is-vr 3-15(32)
size 3-15(33)
alloc-state 3-16(53)

Note: These parts of the internal storage which are identified in the allocation state must not be modified by the environment.

\[ \text{(116)} \]

\[ \text{is-ds-dir-step}(ds-dir_1, ds-dir_2) = \]
\[ \text{is-ds-dir}(ds-dir_2) \&
\]
\[ (\forall d-n) (d-n \in ds-n(ds-dir_1) \equiv d-n \in ds-n(ds-dir_2)) \&
\]
\[ \text{is-tm}(s-tm(ds-n(ds-dir_1)) \equiv \text{is-tm}(s-tm(ds-n(ds-dir_2))) \]

Ref.: is-ds-dir 3-27(101)

Note: Data sets and transmission error flags must not disappear. However, existing data set members may be replaced and/or additional members may be entered. A setting of the transmission flag to TMT will be interpreted as a transmission error through the next following data transmission or from the flagged data set.

\[ \text{(117)} \]

\[ \text{is-time-step}(time_1, time_2) = \]
\[ \text{is-intg-val}(time_2) \& time_2 \geq time_1 \]

Ref.: is-intg-val 9-3(5)

Note: The value of the time component is required not to decrease in the course of the computation.
(119) \textit{is-reply-step}(\textit{reply-1}, \textit{reply-2}) =
\begin{align*}
\textit{is-named-message-list}(\textit{reply-2}) & \land \\
(\exists \textit{reply})(\textit{is-list}(\textit{reply}) & \land \textit{reply-2} = \textit{reply-1''reply})
\end{align*}
\textbf{Ref.:} \textit{is-named-message} 3-26(108)

\textbf{Note:} Reply messages may be received from the environment. They are interpreted by the display statement (cf. 11.8).

(120) \textit{is-an-step}(\textit{an-1}, \textit{an-2}) =
\begin{align*}
\textit{is-an}(\textit{an-2}) & \land \mathbf{(Ident)} \neg\mathbf{(Ident)}(\textit{an-1}) \Rightarrow \mathbf{(Ident)}(\textit{an-2} = \texttt{attn}_1)
\end{align*}
\textbf{where:}
\begin{align*}
\texttt{attn}_1 = \mu(\text{Ident}(\textit{an-1});<\texttt{s-info};\text{Ident}(\textit{an-1});\texttt{s-info}>)
\end{align*}
\textbf{Ref.:} \textit{is-an} 3-18(66)

\textbf{Note:} Attentions may be received from the environment (cf. 10.3).

\subsection*{3.7.3 THE INTERRUPT STEP}

In the interrupt step, tasks may be shifted into the active state (cf. 5.3.1), and the interruption of the normal execution in tasks which received an asynchronous attention is prepared (cf. 10.3.1). The predicate \textit{test-activate} determines whether the activation of tasks is required.

(121) \textit{prep-interrupt}(\textit{t-1}, \textit{t-2}) =
\textit{prep-activate}(\textit{t-1}, \textit{prep-attn}(\textit{t-1}, \textit{t-2}))

(122) \textit{prep-attn}(\textit{t-1}, \textit{t-2}) =
\begin{align*}
\mu(\textit{t-2};<\texttt{tn} \texttt{s-pa}; \textit{prep-attn-1}(\texttt{tn} \texttt{s-pa}(\textit{t-2}), \mu(\texttt<s-ident};\text{ident})>) \\
\texttt{tn} = \texttt{s-task};\texttt{Ident};\texttt{s-an}(\textit{t-2}) & \land \texttt{s-info};\texttt{Ident};\texttt{s-an}(\textit{t-1}) \land \texttt{s-info};\texttt{Ident};\texttt{s-an}(\textit{t-2}))
\end{align*}
\textbf{Ref.:} \textit{prep-attn-1} 10-14(39)

(123) \textit{prep-activate}(\textit{t-1}, \textit{t-2}) =
\textit{test-activate}(\textit{t-1}, \textit{t-2}) \leftarrow
\begin{align*}
\mu(\textit{t-2};<\texttt{s-pa}; \textit{activate-tasks};\texttt{s-pa}(\textit{t-2}), <\texttt{s-delay}; \texttt{s-td};\texttt{delay}_1>) \\
\texttt{T} \leftarrow \textit{t-2}
\end{align*}
\textbf{where:}
\begin{align*}
\texttt{delay}_1 = \{\text{time} \mid \text{time} < \texttt{s-delay};\texttt{s-td}(\textit{t-2}) \land \text{time} > \texttt{s-time};\texttt{s-td}(\textit{t-2})\}
\end{align*}
\textbf{Ref.:} \textit{activate-tasks} 5-12(42)
\[(124) \quad \text{test-activate}(\xi-1, \xi-2) = \]
\[
\begin{align*}
& \text{s-reply} \cdot \text{s-m}(\xi-1) \neq \text{s-reply} \cdot \text{s-m}(\xi-2) \lor \\
& (\text{3ds-n})(\text{is-ds} \cdot \text{ds} \cdot \text{ds-n} \cdot \text{ds-dir} \cdot \text{es}(\xi-1) \land \\
& \text{s-ds} \cdot \text{ds-n} \cdot \text{ds-dir} \cdot \text{es}(\xi-1) \neq \text{s-ds} \cdot \text{ds-n} \cdot \text{ds-dir} \cdot \text{es}(\xi-2)) \lor \\
& (\text{ident})(\text{is-ASYN} \cdot \text{spec} \cdot \text{ident} \cdot \text{s-an}(\xi-1) \land \\
& \text{s-info} \cdot \text{ident} \cdot \text{s-an}(\xi-1) \neq \text{s-info} \cdot \text{ident} \cdot \text{s-an}(\xi-2)) \lor \\
& (\text{time})(\text{time} \cdot \text{s-delay} \cdot \text{td}(\xi-1) \land \text{time} \neq \text{s-time} \cdot \text{td}(\xi-2)) \\
\end{align*}
\]

Ref.: is-ds 3-27(103)

Note: Tasks are activated if reply messages, input to transient files, or asynchronous attentions have occurred, or if the time condition for a delay statement is satisfied.

3.7.4 OPTIMIZATION

The above predicates define computations of the PL/I machine disregarding optimization attributes. These computations are called strict computations. The set of strict computations is modified by optimization rules in two respects. First, additional valid computations may be derived from strict computations using the rules for commoning of expressions. Second, there are strict computations which are invalid because of the wrong use of optimization attributes in the interpreted text. Both the rules for commoning of expressions and the additional constraints for computations imposed by optimization are presented in chapter 13.
ABSTRACT SYNTAX AND INTERPRETATION OF PL/I

4. PROGRAM INITIALIZATION

This chapter describes the initial state of the PL/I machine and its first actions up to the regular run of the program. Section 3.1 defines the initial state of the machine, section 2.2 the initial instruction int-program contained in the control of this initial state and the actions started by this instruction, and section 3.3 the so-called "prepass" of the program.

4.1 THE INITIAL STATE

The interpretation of a PL/I program starts with an initial state $E_0$ of the PL/I machine which is essentially a cleared machine. It contains one active task with the task-event name s-main. Its control consists of the initial instruction int-program. The initial state depends on the following nine parameters:

- $stg \text{ is-vr}$: The initial storage of the machine. It should be noted that the effect of the interpretation of a program may well depend on the initial storage $stg$, though none of these dependencies can be deduced from this document since they are implementation defined. It is the responsibility of the programmer not to use storage parts which have not been explicitly initialized by his program.

- $es \text{ is-es}$: The initial external storage. It contains in particular the input data for the program interpretation.

- $time \text{ is-intg-val}$: An integer value denoting the initial time (in milliseconds).

- $date \text{ is-intg-val}$: An integer value denoting the date.

- $tv \text{ is-gen}$: The generation of the task variable associated with the main task. Its storage part contains the priority value of the main task.

- $ev \text{ is-gen}$: The generation of the event variable associated with the main task.

- $t \text{ is-proper-program}$: The program text to be interpreted. It satisfies the rules of the abstract syntax given in chapter 2.

- $call \text{ is-call v is-ref}$: The call statement or function reference which initiates the interpretation of the program. In order to start successfully, it has to call one of the external entry constants declared in the declaration part of the program.

- $gen \text{ is-gen v is-0}$: A scalar generation, the storage part of which is determined finally to receive and return the function value if the initial call is specified by a function reference. It is 0 if the initial call is specified by a call statement.
4. INITIATING AND TERMINATING ACTIONS OF PROGRAM INTERPRETATION

The control of the initial state consists of the instruction int-program which starts the interpretation of the program. It has three arguments: the program text t, the initial call statement or function reference call and the target generation gen for the function value if applicable.
The program interpretation consists of: the so-called prepass, which performs certain actions for some classes of declarations as described in section 4.3, the program prologue, which performs the same actions as a block prologue (cf. 6.1) with respect to the declaration part of the program, the regular run of the program initiated by the initial call statement or function reference, and finally the program epilogue which performs exactly the same actions as a normal task epilogue (cf. 5.2).

Metavariables

\[ t \quad \text{is-program} \quad \text{the program text to be interpreted and its different modified forms} \]

\[ \text{call} \quad \text{is-call-st} \lor \text{is-ref} \quad \text{the initial call statement or function reference} \]

\[ \text{gen} \quad \text{is-gen} \lor \text{is-0} \quad \text{the target generation for the returned function value or 0} \]

\[ \text{id} \quad \text{is-id} \quad \text{identifier of a declaration} \]

\[ e \quad ([<\text{id}\text{:is-n}> \mid \text{is-id(id)}]) \quad \text{a local environment (cf. 6.1.1)} \]

\[ e \quad \text{is-selector} \quad \text{a selector selecting a declaration cut of the program text } t \]

\[(4) \quad \text{is-program}(t, \text{call}, \text{gen}) = \]

\[ \text{task-epilogue}(\text{NORMAL}); \]

\[ \text{call-program}(\text{call-1}, \text{gen}); \]

\[ \text{call-1}\text{-qualify-names}(\text{call}, e); \]

\[ \text{program-prologue}(t-3, e); \]

\[ t-3\text{-qualify-names}(t-2, e); \]

\[ t-2\text{-prep-text}(t-1, e); \]

\[ e = \text{mk-ext-e}(t), \]

\[ t-1\text{-qualify-names}(t, E_0) \]

\[ \text{where:} \]

\[ E_0 = \mu_0([<\text{id}\text{:is-built-in}> \mid \text{is-built-in(id)}]) \]

Ref.: \quad \text{task-epilogue 5-7(23)}

\[ \text{qualify-names 6-3(5)} \]

\[ \text{prep-text 4-6(10)} \]

\[ \text{is-built-in 12-13(31)} \]

Note: The first qualification with the local environment \( E_0 \) and the attribute directory \( E_0 \) in the initial state (cf. 4.1) ensure that all builtin functions are available for all expression evaluations during the prepass.
(5) \texttt{sk-ext-e(t) =}
\begin{verbatim}
  pass(e); \texttt{un-name | (exists \texttt{is-ext-decl-id-e(t)})}
\end{verbatim}

Ref.: \texttt{un-name 3-10(20)}

(6) \texttt{is-ext-decl(decl) =}
\begin{verbatim}
  is-EXT*scope(decl) \lor is-BUILTIN(decl) \lor is-COMD(decl) \lor is-attn(decl)
\end{verbatim}

for: \texttt{is-decl(decl)}

(7) \texttt{program-prologue(t,e) =}
\begin{verbatim}
  (Vid) (\texttt{-is-G-id(dp_1) \lor is-entry-const-id(dp_1) \land is-EXT*scope-id(dp_1)}) \rightarrow
  upd-dn(e_1,s-body-part(t));
  upd-st(dp_1, e_1)
\end{verbatim}

\texttt{T -> error}

where:
\begin{align*}
  dp_1 &= s-decl-part(t) \\
  e_1 &= \mu_0(\langle\texttt{id:id(e)}\rangle \land \texttt{-is-G-id(dp_1)})
\end{align*}

Ref.: \texttt{upd-dn 6-5(11)}
\texttt{upd-st 6-4(10)}

(8) \texttt{call-program(call,gen) =}
\begin{verbatim}
  is-call-st(call) \land is-Q(gen) \rightarrow \texttt{int-call-st(call)}
\end{verbatim}
\begin{verbatim}
  is-ref(call) \land is-gen(gen) \rightarrow
  test-assign(gen,op);
  cp:eval-ref(call)
\end{verbatim}

\texttt{T -> error}

Ref.: \texttt{int-call-st 6-13(34)}
\texttt{is-gen 3-14(30)}
\texttt{eval-ref 8-20(54)}

(9) \texttt{test-assign(gen,op) =}
\begin{verbatim}
  s-eva(gen) = s-eva(op) \rightarrow \texttt{assign(gen,op)}
\end{verbatim}

\texttt{T -> error}

for: \texttt{is-op(cp)}

Ref.: \texttt{assign 8-5(23)}
\texttt{is-op 9-9(34)}

4 4. PROGRAM INITIALIZATION
4.3 THE PREPASS

The preprocess modifies the original program text by inserting a unique name as a selector component into each declaration of a static or controlled variable, an external entry constant, a file constant or an attention. These unique names will later be used as denotations of their declarations. All internal declarations get individual unique names, while all external declarations of the same identifier get a common one.

Furthermore, for these declarations certain entries are made under their unique names into different directories, namely: for static variables allocation and initialization of storage and the corresponding entry in the aggregate directory AG; for controlled variables an initial dummy entry (which is possibly used for testing on parameter passing, cf. 6.2.3.4) into the generation list in the aggregate directory AG; for file constants an entry into the file directory FD; and for attentions the evaluated environment attribute into the attention environment directory AE.

Finally, the preprocess tests, that all external declarations of the same identifier match. This is done by producing (by the instruction prep-decl) normal forms for all external declarations, so-called "evaluated declarations" (evdecl). The instruction prep-decls-1 ensures that all declarations with the same inserted unique name have the same evaluated declaration, before it performs the above described entries into the different directories.

Metavariables

| t     | is-program | modified form of the program text |
| s     | is-selector| a selector selecting a declaration cut of t |
| id    | is-id      | identifier of a declaration |
| decl  | is-decl    | a declaration |
| b     | is-n       | a unique name associated as denotation with a declaration |
| t-den |           | an auxiliary object having the unique names b at those selectors s where in the program t the corresponding declarations decl are located |
| evdecl|           | an "evaluated declaration" (see above) |
| t-evdecl|         | an auxiliary object having the evaluated declarations evdecl at those selectors s where in the program t the corresponding declarations decl are located |
| aggr  | is-aggr    | aggregate attribute of a static variable |
| eva   | is-eva     | an object having a similar structure as an evaluated aggregate attribute (is-eva), but having additionally lists of operands and asterisks as s-init components of the scalar components |
| init  | is-init    | initial of a static declaration |
| i,k   | is-inte-val| integer values |
| fa    | is-file-attr| a file attribute |
ABSTRACT SYNTAX AND INTERPRETATION OF PL/I

fas is-file-attr-set a set of file attributes

ea is-ea an evaluated environment attribute

(10) prep-text(t,e) =
    prep-text-1(t,t-den);
    prep-decls(t,t-den);
    \[ (\text{un-name} \subseteq \text{INT}\times\text{scope}\times (t) \& ((\text{is-STATIC} \lor \text{is-CTL}) \lor \text{is-stg-cl} \times (t) \lor \text{is-file-const} \times (t))) \& (\text{un-name} \subseteq \text{ext-decl} \times (t) \& (\exists t)(e = \text{id-}t)) \]

Ref.: un-name 3-10 (20)
      is-ext-decl 4-4 (6)

Note: The argument e is a local environment having a unique name for each identifier of an external declaration in the program t.

(11) prep-text-1(t,t-den) =
    PASS: \[ (\text{den} \subseteq \text{e} \times (t-den) \lor (\text{is-prop-var} \lor \text{is-entry-const} \lor \text{is-file-const} \lor \text{is-attn}) \times (t)) \]

(12) prep-decls(t,t-den) =
    prep-decls-1(t-evdecl,t-den);
    \[ (\text{e} \times (t-evdecl) \subseteq \text{prep-decl} \times (t) \& \text{is-}n \times (t-den)) \]

6 4. PROGRAM INITIALIZATION
(13) \( \text{prep-decl}(\sigma, t) = \)
\[
\text{is-correct-static}(\sigma, t) \rightarrow
\]
\[
\text{pass}(\text{evdecl});
\]
\[
\text{s-stg-cl}(\text{evdecl}) \equiv \text{pass}(\text{STATIC}),
\]
\[
\text{e-s-e}(\text{evdecl}) \equiv \text{eval-static-aggr}(\text{s-aggr}\sigma(t), 1)
\]
\[
\text{is-CTL} \ast \text{s-stg-cl}\sigma(t) \rightarrow \text{PASS}\mu_0(\langle\text{s-stg-cl}\sigma(\text{CTL}), \langle\text{e-s-e}\text{pure-aggr}\text{s-aggr}\sigma(t)\rangle\rangle)
\]
\[
\text{is-file-const}\sigma(t) \land \text{is-correct-env-attr}(\text{s-env-attr}\sigma(t), t) \rightarrow
\]
\[
\text{prep-file}(\text{is-env-fas}\text{file-attr}\sigma(t), \text{ea}, \text{id}_0);
\]
\[
\text{e-s-e}\text{eval-env-attr}(\text{s-env-attr}\sigma(t))
\]
\[
\text{is-attr}\sigma(t) \land \text{is-correct-env-attr}(\text{s-env-attr}\sigma(t), t) \rightarrow
\]
\[
\text{eval-env-attr}(\text{s-env-attr}\sigma(t))
\]
\[
\text{is-entry-const}\sigma(t) \rightarrow \text{PASS}:\text{ENTRY}
\]
\[
(\text{is-BUILTIN} \lor \text{is-COND})\sigma(t) \rightarrow \text{PASS}:\sigma(t)
\]
\[
T \rightarrow \text{error}
\]

where:
\[
\text{id}_0 = (\text{lid}) (\exists \sigma)(\sigma = \text{id} = \sigma - 1)
\]

Ref.: pure-aggr 6-20 (52)
\[
eval-env-attr 11-21 (57)
\]

Note: Beside incorrect static variable, file constant and attention declarations this instruction rejects also external automatic variable declarations.

(14) \( \text{is-correct-static}(\sigma, t) \rightarrow \)
\[
\text{is-STATIC} \ast \text{s-stg-cl}\sigma(t) &
\]
\[
(\text{is-ref}(\text{is-decl-1}(\text{s-aggr}\sigma(t), \text{ref}) \rightarrow
\]
\[
\text{is-BUILTIN}\text{matching-decl}(\text{head}\ast \text{id-list}(\text{ref}), \sigma, t))
\]
for: \( \text{is-ref}(\text{ref}) \)

Ref.: is-dep-1 6-6 (17)

Note: This predicate tests that the static variable declaration \( \sigma(t) \) is allowed in the context \( \sigma \) of the program \( t \). This is the case if all identifiers occurring in expressions to be evaluated during the prepass are in the scope of a builtin function declaration.

(15) \( \text{matching-decl}(\text{id}, \sigma, t) = \)
\[
(\text{decl})
\]
\[
(3\sigma - 1, \sigma - 2) (\sigma = \sigma - 2 \ast \sigma - 1 \land \text{decl} = \text{id} \ast \text{decl-part}\sigma - 1(t) \land \text{is-decl}(\text{decl}) \land
\]
\[
\neg (3\sigma - 3, \sigma - 4) (\sigma - 3 \neq 1 \land \sigma = \sigma - 4 \ast \sigma - 3 \ast \sigma - 1 \land
\]
\[
\text{decl} = \text{id} \ast \text{decl-part}\sigma - 3 \ast \sigma - 1(t))
\]

4. PROGRAM INITIALIZATION
(16) \[ \text{eval-static-aggr}(\text{aggr},k) = \]
\[ \text{is-array}(\text{aggr}) \rightarrow \]
\[ \text{pass}(\text{eva}) ; \]
\[ \text{s-elem}(\text{eva}) : \text{eval-static-aggr}(\text{s-elem}(\text{aggr}),k-1) ; \]
\[ k-1 : \text{nmbr-of-comp}(\text{eva},k) ; \]
\[ \text{s-lbd}(\text{eva}) : \text{eval-intg-expr}(\text{s-lbd}(\text{aggr})) , \]
\[ \text{s-ubd}(\text{eva}) : \text{eval-intg-expr}(\text{s-ubd}(\text{aggr})) \]
\[ \]
\[ \text{is-struct}(\text{aggr}) \rightarrow \]
\[ \text{pass}(\text{eva}) ; \]
\[ \{ \text{s-aggr} \cdot \text{elem}(i) \} (\text{eva}) : \text{eval-static-aggr}(\text{s-aggr} \cdot \text{elem}(i,\text{aggr}),k) \mid \]
\[ 1 \leq i \leq \text{length}(\text{aggr}) \]
\[ \]
\[ \text{is-scalar}(\text{aggr}) \rightarrow \]
\[ \text{pass}(\text{eva}) ; \]
\[ \text{s-init}(\text{eva}) : \text{eval-static-init}(\text{s-init}(\text{aggr}),\text{eva},k) ; \]
\[ \text{s-da}(\text{eva}) : \text{eval-da}(\text{s-da}(\text{aggr})) , \]
\[ \text{s-dens}(\text{eva}) : \text{pass}(\text{s-dens}(\text{eva})) \]

Ref.: \[ \text{eval-intg-expr} 8-22(60) \]
\[ \text{eval-da} 6-9(24) \]

(17) \[ \text{nmbr-of-comp}(\text{eva},k) = \]
\[ \text{is-intg-val} \cdot \text{s-lbd}(\text{eva}) \& \text{is-intg-val} \cdot \text{s-ubd}(\text{eva}) \& \text{s-lbd}(\text{eva}) \leq \text{s-ubd}(\text{eva}) \rightarrow \]
\[ \text{PASS}: k - (\text{s-ubd}(\text{eva}) - \text{s-lbd}(\text{eva}) + 1) \]
\[ \text{T} \rightarrow \text{error} \]

Ref.: \[ \text{is-intg-val} 9-3(5) \]

(18) \[ \text{eval-static-init}(\text{init},\text{eva},k) = \]
\[ \sim (\text{is-init-elem-list}(\text{init}) \& \text{is-correct-eva}(\text{eva})) \rightarrow \text{error} \]
\[ k = 0 \rightarrow \text{PASS}:<> \]
\[ \text{is-<}(\text{init}) \rightarrow \text{is-*head}(\text{init}) \rightarrow \]
\[ \text{mk-list}(*,\text{opr}) ; \]
\[ \text{opr} : \text{eval-static-init}(\text{tail-1}(\text{init}),\text{eva},k-1) \]
\[ \text{is-expr-head}(\text{init}) \rightarrow \]
\[ \text{mk-list}(\text{op},\text{opr}) ; \]
\[ \text{opr} : \text{eval-static-init}(\text{tail}(\text{init}),\text{eva},k-1) ; \]
\[ \text{op} : \text{convert}(\text{eva},\text{op-1},2,2) ; \]
\[ \text{op-1} : \text{eval-expr}(\text{head}(\text{init}),\text{s-da}(\text{eva})) \]
\[ \text{is-init-iter-head}(\text{init}) \rightarrow \]
\[ \text{eval-static-init}(\text{init-1},\text{eva},k) ; \]
\[ \text{init-1} : \text{pass-rep-conc}(\text{i,init-head}(\text{init}),\text{tail}(\text{init})) ; \]
\[ \text{i} : \text{eval-intg-expr}(\text{s-rep-head}(\text{init})) \]

cont'd

8. PROGRAM INITIALIZATION
Ref.:  
- is-correct-eva 7-5(9)  
- convert 8-8(17)  
- eval-expr 8-10(24)  
- rep-conc 7-17(54)  
- eval-intg-expr 8-22(60)

(19) \text{is-correct-env-attr}(\sigma, t) =

Note: This predicate is implementation defined. It tests analogously as the predicate is-correct-static that the environment attribute \( \sigma(t) \) is allowed in the context \( \sigma \) of the program \( t \).

(20) prep-file(fas, ea, id) =
    PASS: \mu_0(<s-attr:fas>, <s-ea:ea>, <s-id:id>)

(21) impl-fas(fas) =
    fas \cup \bigcup_{fa \in fas} \text{impl-fas-1}(fa)

(22) impl-fas-1(fa) =
    \begin{align*}
    &fa \in \{\text{CST, BST, REC, INF, OUT}\} \rightarrow \{\}\quad (22a) \\
    &fa \in \{\text{UPD, SEQ, TRA, BUF, UNB, KEY}\} \rightarrow \{\text{REC}\} \\
    &\text{is-DIR}(fa) \rightarrow \{\text{REC, KEY}\} \\
    &\text{is-FBT}(fa) \rightarrow \{\text{CST, OUT}\} \\
    &\text{is-BAC}(fa) \rightarrow \{\text{REC, INF, SEQ}\} \\
    &\text{is-EXC}(fa) \rightarrow \{\text{REC, UPD, DIR, KEY}\}
    \end{align*}

(23) prep-decls-1(t-evdecl, t-den) =
    \begin{align*}
    \textbf{null}; \\
    \{\text{prep-decl-1}(b, evdecl) \mid \text{is-b}(b) \& (3a)(b = \sigma(t-den)) \& \\
    \text{evdecl} = (\text{levdecl}) (3b)(b = \sigma(t-den) \& \text{evdecl} = \sigma(t-evdecl))\}
    \end{align*}
(24) \[ \text{prep-decl-1}(b, evdecl) = \]
\[ \text{is-STATIC}\Rightarrow \text{is-stg-cl}(evdecl) \Rightarrow \]
\[ \text{initialize}(\text{gen}, \text{is-ev}(evdecl)); \]
\[ \text{gen}: \text{allocate}(b, \text{is-ev}(1) \text{STATIC}) \]
\[ \text{is-CTL}\Rightarrow \text{is-stg-cl}(evdecl) \Rightarrow \text{s-ev}(AG; b; evdecl)) \]
\[ \text{is-file-attr-set}\Rightarrow \text{is-attr}(evdecl) \Rightarrow \text{s-fd}(FF; b; evdecl) \]
\[ \text{is-} \text{ea}(evdecl) \Rightarrow \text{s-ev}(FF; b; evdecl) \]
\[ \text{(is-ENTRY } \lor \text{is-BUILTIN } \lor \text{is-COND}(evdecl)) \Rightarrow \text{null} \]

where:
\[ \text{ev}_{1} = \delta(s-ev(evdecl); \{s-init; \land \text{is-}0\text{-s-init}; s-ev(evdecl))} \]

Ref.:
\[ \text{initialize} \ 7-15(49) \]
\[ \text{allocate} \ 7-11(25) \]
\[ \text{is-ev} \ 11-21(58) \]
This chapter describes the attaching, terminating and synchronizing of tasks and input and output events. The parallel actions of several tasks and input and output events are described by a sequentialized model performing one action of one task after another action of another task. By this model, the various well-defined relations between tasks are specified. However, the model does not reflect the undefinedness which would result from access to storage or external storage by truly parallel, i.e. simultaneous, execution of tasks. Moreover, the execution of one instruction of one task in the model (one uninterruptable action of the model) may be represented in a concrete implementation by a series of uninterruptable actions which may be performed in parallel with, or intermixed with, actions of other tasks. Thus it is an unfortunate consequence of the model that it does in fact specify more about the execution of a PL/I program than is guaranteed by the language. However, since the language guarantees well-defined results for the actions performing the synchronization of tasks, certain instructions of the model are to be considered as principally uninterruptable, namely those assigning and accessing event variables or modifying and accessing state components of other tasks.

Section 5.1 describes the attaching, section 5.2 the termination of tasks. Section 5.3 describes the synchronization of tasks and input and output events by the wait statement. This includes the completion of "semi-complete" input and output events (i.e. condition raising and termination of input and output events whose data transmission had already been completed). Section 5.4 describes the delay statement which synchronizes tasks governed by the real time. The attaching of input and output events is described in chapter 7.

5.1 ATTACHING OF TASKS

A new task is attached by a call statement specifying a parallel action option pa-opt. The general actions of a call statement are described in section 6.2.1. The specific actions of a task call are initiated by the instruction call-task, described here, which is activated by the instruction int-call-1 in section 6.2.1.

Before the proper attaching of the new task its task-event specification is prepared, particularly the generations of the associated task and events variables are established either by evaluating the generations of the specified variables or by allocating dummy variables. Furthermore the priority of the new task is determined from the priority expression specified in the call statement, the value of the specified task variable and the priority of the current task.

The proper attaching of the new task is performed by the instruction attach-task which enters the initialized components of the new task under a unique task-event name tn into the parallel action part P4 of the machine. The control of this new task starts with the instruction int-task which performs a regular procedure call and after return from the called procedure terminates the task by the instruction task-epilogue defined in section 5.2.

Metavariables

den is-entry-den denotation of the called entry
argl is-pa-opt list of (evaluated) arguments passed to the called entry (cf. 6.2.3).

pa-opt is-pa-opt task options specified in the call statement
tn is-n unique task name of the attached task

5. TASK
te is-te task-event specification of the attached task
pa is-pa the parallel action part PA
ref is-ref ∨ is-* optional reference specifying the task or event variable of the attached task
tv is-gen generation of the task variable
expr is-opt-expr optional expression specifying the priority of the attached task
cp is-op ∨ is-0 optional priority operand
eva is-scalar-eva scalar aggregate attribute for task or priority operand
gen is-gen a generation

(1) call-task(den, argl, pa-opt) =
  attach-task(den, argl, tn, te, cp);
  tn=tn-name,
  op=eval-pri(op-1, tv, s-task(pa-opt));
  op-1=eval-pri(s-ref, s-tv(pa-opt)),
  tv=pass-s-tv(te);
  te=mk-te(pa-opt)
Ref.: un-name 3-10 (20)

(2) mk-te(pa-opt) =
  pass(te):
  s-trv(te):eval-te-gen(s-task(pa-opt), TASK),
  s-ev(te):eval-te-gen(s-event(pa-opt), EVENT),
  s-io-ev(te):pass({}),
  s-free-set(te):pass({})

(3) eval-te-gen(ref, eda) =
  is-var-ref(ref, AT) ∨ is-aggr-ref(ref, AT) = eda → eval-ref-aggr(ref)
  is-*[ref] →
  allocate-auto(b, aggr-scalar(eda));
  b:un-name
  ? → error

for: (is-TASK ∨ is-EVENT) (eda)

Ref.: is-var-ref 8-9 (21)
      aggr-ref 8-25 (71)
      eval-ref-gen 8-28 (82)
      allocate-auto 7-10 (24)
      aggr-scalar 8-23 (58)
      un-name 3-10 (20)
Note: If no task or event variable is specified, a dummy variable is allocated which is automatically freed like automatic variables at the end of the calling block.

(4) \[ \text{eval-pri-1}(\text{expr}) = \]
\[ \neg \text{is-0}(\text{expr}) \rightarrow \]
\[ \text{convert-1}(\text{PRI-EVA}, \text{op}); \]
\[ \text{op} : \text{eval-expr}(\text{expr}, 0) \]
\[ \text{T} \rightarrow \text{null} \]

Ref.: \text{convert-1} 9-29(119)
\text{eval-expr} 8-10(24)

(5) \[ \text{PRI-EVA} = \]
\[ \text{aggr-scalar-bintg-edat}(\text{PRI-PREC}) \]

Ref.: \text{aggr-scalar} 8-23(64)
\text{bintg-edat} 9-11(48)

(6) \[ \text{PRI-PREC} = \]

Note: \text{PRI-PREC} is an implementation defined positive integer. It is the default precision for priority values.

(7) \[ \text{eval-pri}(\text{op}, \text{tv}, \text{ref}) = \]
\[ \neg \text{is-0}(\text{op}) \rightarrow \text{eval-pri-2}(\text{is-eva}(\text{tv}), \text{op}) \]
\[ \neg \text{is-0}(\text{ref}) \rightarrow \text{gen-op}(\text{tv}) \]
\[ \text{T} \rightarrow \]
\[ \text{convert-pri}(\text{is-eva}(\text{tv}), \text{op-1}); \]
\[ \text{op-1} : \text{gen-op}(\text{is-tv}(\text{T})) \]

Ref.: \text{gen-op} 8-22(59)

(8) \[ \text{eval-pri-2}(\text{eva}, \text{op}) = \]
\[ \text{convert-pri}(\text{eva}, \text{op-4}); \]
\[ \text{op-4} : \text{convert-1}(\text{PRI-EVA}, \text{op-3}); \]
\[ \text{op-3} : \text{eval-infix-expr}(\text{op}, \text{op-2}, \text{ADD}); \]
\[ \text{op-2} : \text{convert-pri}(\text{PRI-EVA}, \text{op-1}); \]
\[ \text{op-1} : \text{gen-op}(\text{is-tv}(\text{T})) \]

Ref.: \text{convert-1} 9-29(119)
\text{eval-infix-expr} 9-13(64)
\text{gen-op} 8-22(59)
(9) \textit{convert-pri}(eva, op) = \\
\text{PASS:} \mu_0(\langle s-eva:eva>, \langle s-vr:rep(eva, op-val(op)) \rangle) \\
\text{Ref.:} \text{rep 9-6(19)} \\
op-val 9-9(36) \\
\text{Note:} This instruction transforms priority operands (i.e. binary integer values) into task operands with the same value and vice versa. Both operands, though having the same values, have possibly different value representations; the former are necessary for arithmetic operations, the latter for assignment to task variables.

(10) \textit{attach-task}(den, argl, tn, te, op) = \\
is-active-event(s-ev(te), PA) \rightarrow \text{call-ccnd}(\text{ERROR}) \\
is-active(s-tv(te), PA) \lor \text{is-active}(s-ev(te), PA) \lor \text{~is-indep}(s-pp*evv(te), s-pp*ev(te)) \rightarrow \text{error} \\
T \rightarrow \\
\mu(\langle s-te:te>, \langle s-ag:ag0>, \langle s-en:EN_1>, \langle s-cs:CS>, \langle s-d:fd \rangle, \langle s-c:int-task(den, argl) \rangle) \\
\text{where:} \\
\text{stg}_0 = \text{el-ass}(s-vr(op), s-pp*tv(te), \text{el-ass}(s-evv, s-pp*evv(te), s)) \\
ev-vr_0 = \text{rep}(s-eva*evv(te), \mu_0(\langle s-compl:0-BIT>, \langle s-status:0 \rangle)) \\
ag_0 = \delta(\text{AG}; \text{dummy-set}_4) \\
ei_0 = \mu(\text{EI}; \langle s-task-set; s-task-set(\text{EI}) \rangle \cup \{tn\}) \\
an_1 = \mu_0(\langle s-den; elem(i, argl) | 1 \leq i \leq \text{length(argl)} \rangle \land \text{is-DUMMY}(\text{elem}(i, argl)) \\
\text{dummy-set}_4 = \{s-den*elem(i, argl) | 1 \leq i \leq \text{length(argl)} \land \text{is-DUMMY}(\text{elem}(i, argl))\} \\
fd_1 = \mu(\text{FD}; \\
\text{\{s-fd-stf; } s-fd-stf(\text{FD}); \text{own}, s-tst) \rangle \land \\
\text{~is-start(\text{FD}) \land (is-n \lor is-s-st-prt)}(\text{fd})\}) \\
EN_1 = \mu_0(\langle s-enab-list; \rangle, \langle s-assoc-list; \rangle, \langle s-wait-list; \rangle) \\
\text{Ref.:} \text{call-ccnd 10-18(54)} \\
is-indep 3-16(43) \\
el-ass 3-16(48) \\
rep 9-6(19) \\
is-s-st-prt 3-24(90) \\
\text{Note:} This instruction performs the proper attaching of the new task. At the same time it assigns the priority value to the task variable and the incomplete-normal value to the event variable and enters the task name \text{tn} of the new task into the epilogue information \text{EI} of the calling block, thereby causing that the new task is terminated at block end if it does not terminate earlier.

The task-event specification \text{TE} of the new task is as prepared before by \text{mk-te}, the dump \text{D} and the condition status \text{CS} are inherited from the current block activation of the calling task. The aggregate directory of the called task consists of the last generations of all variables in the aggregate directory \text{AG} of the calling task (which can not be freed in the
called task). Only for the passed dummy arguments the complete entries (consisting of their proper generation and an asterisk) are taken over in order to free them at task end (these entries are removed from the aggregate directory of the calling task in order to avoid another freeing). The file directory $F$ of the called task is essentially the same as that of the calling task with removed $s$-own components in order to avoid closing of not owned files by the called task. All other task local state components of the called task (except its control) are left empty like in the initial state of the computation (cf. 6.1).

(11) \[ \text{is-active-event}(\text{gen}, \text{pa}) = \]

\[ (\exists \text{tn})(\text{gen} = s\text{-evs}\text{-te\text{-tn}}(\text{pa})) \]

Note: This predicate tests whether the generation gen is an event variable generation currently associated with an active task or event. Any attempt to use gen another time as event variable of a task or event or to assign or free it raises the ERROR condition.

(12) \[ \text{is-active}(\text{gen}, \text{pa}) = \]

\[ (\exists \text{tn})(\text{is-gen}(	ext{tv}_1) \land \neg \text{is-indep}(s\text{-pp}(\text{gen}), s\text{-pp}(	ext{tv}_1)) \lor \text{is-gen}(	ext{ev}_1) \land \neg \text{is-indep}(s\text{-pp}(\text{gen}), s\text{-pp}(	ext{ev}_1))) \]

where:
\[ \text{tv}_1 = s\text{-tv\text{-te\text{-tn}}}(\text{pa}) \]
\[ \text{ev}_1 = s\text{-ev\text{-te\text{-tn}}}(\text{pa}) \]

Ref.: \[ \text{is-gen} 3-14(30) \]
\[ \text{is-indep} 3-16(43) \]

Note: This predicate tests whether the generation gen has overlapping storage with a task or event variable currently associated with an active task or event. Any attempt to use such a generation as task or event variable of a new task or to assign or free it, could make the priority or status and completion value of the associated task or event undefined and is therefore erroneous.

(13) \[ \text{int-task}(\text{den}, \text{argl}) = \]

\[ \text{task-epilogue}(\text{NORMAL}); \]
\[ \text{call-proc}(\text{den}, \text{argl}, 0) \]

Ref.: \[ \text{task-epilogue} 5-7(23) \]
\[ \text{call-proc} 5-16(43) \]

5.2 TERMINATION OF TASKS

Normally a task is terminated by return from the called entry. In this case the instruction task-epilogue is executed with the argument NORMAL.

An abnormal termination of a task is caused by a stop or exit statement or by the epilogue of the block activation of the calling task during which the task was attached (cf. 6.1.3). In this case all block activations are terminated one by one by the instruction task-exit, thereby again terminating all tasks attached during these block activations by the instruction task-exits. When all block activations are terminated the instruction task-epilogue is executed with the argument ABNL. By this algorithm it is guaranteed that each task, before it
finally terminates, has terminated all its dynamically descendant tasks. This is used by the stop statement which forces the main task, and thereby all tasks, to terminate.

The task epilogue itself disables all attentions, terminates all input and output events, unlocks all locked records, closes all files and frees all storage belonging to the task. Finally all state components belonging to this task are removed from the state of the PL/I machine and the completion value of the associated event variable is set to complete and its status value is set according to the status of completion (normal or abnormal).

**Restatements**

\( \text{tn} \) is-tn

the unique task name of the terminated task

\( \text{status} \) is-NORMAL \( \lor \) is-ABN

status of terminated task

(14) \( \text{int-stop-st} = \)

\[ \text{program-stop;} \]

\[ \text{call-cond(FINISH)} \]

Ref.: \text{call-cond 10-18(54)}

(15) \( \text{program-stop} = \)

\[ \text{s-pa:} \mu (PA; \langle s-c\#s-main:task-exit \rangle) \]

(16) \( \text{int-exit-st} = \)

\[ \text{TN = s-main } \rightarrow \]

\[ \text{task-exit;} \]

\[ \text{call-cond(FINISH)} \]

\[ T \rightarrow \text{task-exit} \]

Ref.: \text{call-cond 10-18(54)}

(17) \( \text{term-tasks} = \)

\[ \text{term-tasks-wait;} \]

\[ \text{term-tasks-1} \]

(18) \( \text{term-tasks-1} = \)

\[ \text{s-pa:} \mu (PA; \langle s-c\#tn:task-exit \rangle \mid \text{tn} \in \text{active-descendants(PL/PA)}) \]

6 5. TASK
ABSTRACT SYNTAX AND INTERPRETATION OF PL/I

(19) \texttt{term-tasks-wait} =
\begin{align*}
\text{is-} & \{\text{active-descendants}(\text{Ei, PA}) \rightarrow \text{null} \\
\text{T} & \rightarrow \\
\text{term-tasks-wait:} \\
\text{set-wait-State-1} \\
\end{align*}

(20) \texttt{set-wait-State-1} =
\begin{align*}
\text{is-} & \{\text{active-descendants}(\text{Ei, PA}) \rightarrow \text{null} \\
\text{T} & \rightarrow \text{s-te:}μ(\text{T}:<\text{wait:"}>) \\
\end{align*}
Note: Compare the instruction \texttt{set-wait-State} in section 5.3.1.

(21) \texttt{active-descendants(Ei, PA)} =
\begin{align*}
\{\text{tn} \mid \text{tn} \in \text{s-task-set}(\text{Ei}) \& \text{is-} & \text{tn}(\text{PA}) \} \\
\text{for:} & \text{is-}\text{ei}(\text{Ei}), \text{is-}\text{pa}(\text{PA}) \\
\text{Ref.:} & \text{is-}\text{ei} 3-8(16) \\
& \text{is-}\text{pa} 3-3(2) \\
\text{Note:} & \text{This function collects the names of all tasks attached by the current block activation which are not yet terminated.} \\
\end{align*}

(22) \texttt{task-exit} =
\begin{align*}
\text{is-} & \text{-}0(\text{PA}) \rightarrow \text{task-epilogue}(\text{ABNL}) \\
\text{T} & \rightarrow \\
\text{s-d:}μ(\text{P};<\text{c-task-exit}>) \\
\text{e-c:} & \text{epilogue} \\
\text{Ref.:} & \text{epilogue 6-11(31) } \\
\end{align*}

(23) \texttt{task-epilogue}(\text{status}) =
\begin{align*}
\text{delete-task-event}(\text{TE}, \text{status}); \\
\text{free-taskend}; \\
\text{int-disable-1}(\text{s-enab-list}(\text{EN}) \& \text{s-assoc-list}(\text{EN})); \\
\text{term-events}; \\
\text{unlock-taskend}; \\
\text{task-fd-close} \\
\text{Ref.:} & \text{int-disable-1 10-0(23) } \\
\end{align*}

(24) \texttt{term-events} =
\begin{align*}
\text{null;} \\
\{\text{delete-task-event}(\text{en}, \text{ABNL}) \mid \text{en} \in \text{s-io-ev}(\text{TF})\} \\
\end{align*}
(25) \textit{unlock-taskend} =
\begin{align*}
\text{s-fu:} & (\text{FD;TMn | is-n(u) & is-0\cdot u(FD)}) \\
\text{s-pa:} & \text{activate-tasks(FA)}
\end{align*}

Ref.: activate-tasks 5-12(42)

(26) \textit{task-fd-close} =
\begin{align*}
\text{null; for:} & (\text{is-n} \lor \text{is-st-prt}) (f) \\
\text{Ref.:} & \text{close-1 11-29(68)} \\
\text{is-st-prt 3-24(90)}
\end{align*}

(27) \textit{free-taskend} =
\begin{align*}
\text{s-e:} & \text{el-free(free-set}_A, s) \\
\text{s-te:} & \mu(T;\langle s\text{-free-set:} e \rangle) \\
\text{s-ae:} & \emptyset
\end{align*}

where:
\begin{align*}
\text{free-set}_A = \text{s-free-set}(T; s \text{-pp\cdot elem}(i,b(AG)) | \\
\text{is-0\cdot b(AG)} \land 1 \leq i < \text{length\cdot b(AG)})
\end{align*}

for: is-n(b)

Ref.: el-free 3-17(62)

(28) \textit{delete-task-event}(tn, status) =
\begin{align*}
\text{is-2\cdot tn}(PA) \rightarrow \text{null} \\
T \rightarrow \\
\text{s-g:} & \text{el-ass}(\text{ev-vr}_1, s\cdot pp(\text{ev}_1), s) \\
\text{s-set:} & \langle \text{ev}_1 \rangle \\
\text{s-pa:} & \text{activate-tasks}(6(PA;tn))
\end{align*}

where:
\begin{align*}
\text{ev}_1 = s\cdot ev\cdot te\cdot tn(FA) \\
\text{status}_0 = s\cdot status\cdot val(s\cdot eva(\text{ev}_1), s\cdot pp(\text{ev}_1)(s)) \\
\text{status}_A = (\text{is-ABNL}(\text{status}) \land \text{status}_0 = 0 \rightarrow 1, \\
T \rightarrow \text{status}_0) \\
\text{ev-vr}_1 = \text{rep}(s\cdot eva(\text{ev}_1), \mu_0(\langle s\cdot compl:1\cdot BIT>, s\cdot status; \text{status}_A))
\end{align*}

Ref.: el-ass 3-16(48) \\
activate-tasks 5-12(42) \\
val 9-6(18) \\
rep 9-6(19)

Note: Applied to the current task name \textit{TN} this instruction causes a task to terminate itself. Applied to the unique name \textit{ex} of an input/output or
attention event by this instruction a task terminates this event. The first alternative avoids that the PL/I machine runs into error trying to terminate an input/output event which had already been terminated before.

5.3 THE WAIT STATEMENT

The wait statement is a means to synchronize the actions of different tasks or events. It specifies that the current task shall not continue its actions before a certain number of given event variable generations have been set complete, usually by other tasks, either by explicit assignment or automatically by termination of tasks or events.

If an event variable generation is associated with an input or output event, started by the current task, then the wait statement does not test the completion value of this one wait state flag but the actual completion of the event performed by the event. If the data transmission has been recognized as completed (the event is called "semi-complete" in this case), the wait statement itself performs the outstanding actions of the corresponding input or output statement, including the complete setting of the associated event variable generation (cf. section 5.3.2).

5.3.1 THE WAIT MECHANISM

After evaluating the event variable generations and the number of events to be waited for, the wait statement tests whether some of its event variable generations are complete or semi-complete. Each generation recognized as complete or semi-complete is removed from the list and the count number decreased accordingly, thereby the input and output events recognized as semi-complete are fully completed (cf. section 5.3.2). If none of the remaining generations is complete, the instruction wait sets the current task into the wait state until another generation is complete or semi-complete.

Each task and input and output event has in its task-event specification a "wait state flag" s-wait(TE) which is either 0 ("active") or 1 ("waiting"). The priority scheduler selects only those tasks or events for execution of an instruction whose wait state flag is 0. Each waiting instruction, e.g. wait, tests whether the condition it is waiting for is satisfied. If so, it continues, if not, it inserts itself back into the control and sets the current task into the wait state. Each action which possibly may cause the condition of any waiting instruction of any task to be satisfied (e.g. complete setting of an event variable), resets the wait state flags of all tasks and events to 0 (replacing the parallel action part PA of the machine by activate-tasks(2A)). Then, governed by priority selection, each waiting instruction can test again whether its individual condition is satisfied and depending on this it can either continue or go back into the wait state.

To enable the wait statement to consider not only those event variables which are complete at the time when inspected by the instruction test-wait but also those which are set complete and remain to incomplete during the wait statement, saves the event trace ET. It lists in time order all complete settings of event variable generations and all starts of wait statements. Each wait statement is identified by a unique name w which is inserted into the event trace ET when it starts. Each event variable generation is inserted into ET when it is set complete by any means. So it can be tested whether a generation has been set complete later than the start of a specific wait statement even if it has been set incomplete in the meantime (this is done by the function is-corpl).

**Metavaiables**

- **t** is-wait-st the interpreted wait statement
- **ref-list** is-ref-list list of references specifying the event
variables to be waited for

genl is-gen-list list of generations of event variables to be waited for
gen is-gen generation of an event variable
expr is-opt-expr optional expression specifying the number of events to be waited for.
count is-int-val number of events waited for
w is-n unique name identifying the wait statement in FT
stg is-vr the storage S
pa is-pa the parallel action part PA
te is-te the task-event specification TE
et is-et the event trace ET
en is-n a unique event name
i,j,k is-intg-val integer values

(29) \[ \text{int-wait-st}(t) = \]
\[ \text{int-wait-st-1}(s\text{-event-list}(t), s\text{-event-number}(t), w); \]
\[ \text{trace-wait-st}(w); \]
\[ w: \text{un-name} \]

Ref.: \text{up-name 3-10(20)}

(30) \[ \text{trace-wait-st}(w) = \]
\[ s\text{-st}\text{-ST}<w> \]

(31) \[ \text{int-wait-st-1}(\text{ref-list}, expr, w) = \]
\[ (\forall i (1 \leq i \leq \text{length}(\text{ref-list}) \Rightarrow \text{is-var-ref}(\text{elem}(i, \text{ref-list}), \text{AT})) \rightarrow \]
\[ \text{test-wait}(\text{genl}, \text{count}, w); \]
\[ \text{count}\text{-wait-count}(\text{genl}, \text{count}-1); \]
\[ \text{count}-1: \text{eval-intg-expr}(\text{expr}); \]
\[ \text{genl}\text{-conc-event-list}(\text{genl}-1); \]
\[ \{ \text{elem}(i) (\text{genl}-1) : \text{eval-ref-gen}(\text{elem}(i, \text{ref-list})) \} \]
\[ 1 \leq i \leq \text{length}(\text{ref-list}) \}

T \rightarrow \text{error}

Ref.: \text{is-var-ref 8-9(21)}
\text{eval-intg-expr 8-22(60)}
\text{eval-ref-gen 8-28(82)}
(32) \[ \text{conc-event-list} (\text{genl}) = \]
\[ (\forall i : 1 \leq i \leq \text{length} (\text{concl}) \rightarrow \text{is-EVENT} \rightarrow \text{da} \rightarrow \text{eva} \text{elem}(i, \text{concl}) \rightarrow \]
\[ \text{PASS: concl} \]
\[ T \rightarrow \text{error} \]

where:
\[ \text{length} (\text{genl}) \]
\[ \text{concl} = \text{CONC} \sum_{i=1}^{\text{length}(\text{genl})} \text{elem}(i, \text{genl}) \]

(33) \[ \text{gen-list} (\text{gen}) = \]
\[ \text{is-scalar} \rightarrow \text{eva} (\text{gen}) \rightarrow <\text{gen}> \]
\[ T \rightarrow \text{CONC} \sum_{i=1}^{\text{ubd}(\text{gen})} \text{gen-list} \text{sub-gen} (\text{gen}, <i>) \]

Ref.: sub-gen 8-36(115)

(34) \[ \text{wait-count} (\text{genl}, \text{count}) = \]
\[ \text{is-2} (\text{count}) \rightarrow \text{PASS: length} (\text{genl}) \]
\[ T \rightarrow \text{PASS: count} \]

for: (is-intg-val \lor is-0) (\text{count})

Ref.: is-intg-val 9-3(5)

(35) \[ \text{test-wait} (\text{genl}, \text{count}, \text{w}) = \]
\[ \text{is-<>} (\text{genl}) \lor \text{count} \leq 0 \rightarrow \text{null} \]
\[ (3i) \rightarrow \text{is-compl} (\text{genl}, \text{genl}, \text{w}, \text{ET}, \text{PA}, \text{TE}) \rightarrow \]
\[ \text{null:} \]
\[ \text{test-wait} (\text{genl}, \text{count}, \text{w}, \text{i}) \lor \text{is-compl} (\text{genl}, \text{genl}, \text{w}, \text{ET}, \text{PA}, \text{TE}) \]
\[ T \rightarrow \text{wait} (\text{genl}, \text{count}, \text{w}) \]

(36) \[ \text{is-compl} (i, \text{genl}, \text{stg}, \text{w}, \text{et}, \text{pa}, \text{te}) = \]
\[ 1 \leq i \leq \text{length}(\text{genl}) \land \]
\[ \text{is-1-BITs-compl-val} (\text{s-eva} (\text{genl}), \text{s-pp} (\text{genl})) \land \]
\[ (3j,k) (1 \leq j < k \leq \text{length}(\text{et}) \land \text{w} = \text{elem}(j, \text{et}) \land \text{genl} = \text{elem}(k, \text{et}) \land \]
\[ \text{is-semi-compl} (\text{genl}, \text{pa}, \text{te}) \]

where:
\[ \text{genl} = \text{elem}(i, \text{genl}) \]

Ref.: val 9-6(18)
Note: This predicate tests whether the i-th generation in the list genl is either complete (its completion value is 1-BIT), or has been set complete after the start of the wait statement, or is semi-complete (i.e., it is associated with an active input or output event started by the current task whose data transmission has been finished).

(37) $\text{is-semi-compl}(\text{gen}, \text{pa}, \text{te})$
    \[= (\exists \text{en}) (\text{gen} = \text{s-ev-s-te-en}(\text{pa}) \& \text{is-Qs-c-en}(\text{pa}) \& \text{en} \in \text{s-io-ev}(\text{te})) \]

(38) $\text{compl-wait}(\text{genl}, \text{count}, \text{w}, \text{i}) =$
    \[
    \text{c-int-next-st}:
    \text{test-wait}(\text{remove}(\text{genl}, \text{i}), \text{count} - 1, \text{w});
    \text{compl-wait-1}(\text{elem}(\text{i}, \text{genl}))
    \]

Ref.: $\text{int-next-st} 6-39(111)$
\text{remove 7-3(6)}

Note: This instruction overwrites the control of the current task in order to complete the i-th event of the list genl fully before any actions for other events are performed. I.e. the instruction \text{test-wait} enables all complete and semi-complete events to be handled equally. But the first instruction \text{compl-wait} selected for execution prevents all others from being executed before itself has fully finished its actions.

(39) $\text{compl-wait-1}(\text{gen}) =$
    $\text{is-semi-compl}(\text{gen}, \text{P}_\text{A}, \text{T}_\text{E}) \rightarrow \text{compl-event}(\text{en}_1)$
    $\text{T} \rightarrow \text{null}$

where:
$\text{en}_1 = (\exists \text{en}) (\text{gen} = \text{s-ev-s-te-en}(\text{P}_\text{A}))$

(40) $\text{wait}(\text{genl}, \text{count}, \text{w}) =$
    $\exists \text{i} (\text{is-compl}(\text{i}, \text{genl}, \text{S}, \text{w}, \text{IT}, \text{PA}, \text{TP}) \rightarrow \text{test-wait}(\text{genl}, \text{count}, \text{w})$
    $\text{T} \rightarrow$
    $\text{wait}(\text{genl}, \text{count}, \text{w});$
    $\text{set-wait-state}(\text{genl}, \text{w})$

(41) $\text{set-wait-state}(\text{genl}, \text{w}) =$
    $\exists \text{i} (\text{is-compl}(\text{i}, \text{genl}, \text{S}, \text{w}, \text{IT}, \text{PA}, \text{TP}) \rightarrow \text{null}$
    $\text{T} \rightarrow \text{s-te-p}(\text{P}_\text{E}; <\text{s-wait:*})$

Note: This instruction generally sets the current task actually into the wait state. The test, already performed by the instruction \text{wait}, is necessary since in the meantime another task might have satisfied the waiting condition (in which case, of course, the task should not go into the wait state).

12 5. TASK
(42) \[ \text{activate-tasks}(\text{pa}) = \delta(\text{pa}; \{ \text{s-wait} = \text{s-trn} | \text{is-trn}(\text{pa}) \}) \]

\textbf{Note:} Each instruction for which another task might wait replaces \text{pa} by \text{activate-tasks}(\text{PA}) thereby reactivating all waiting tasks.

5.3.2 COMPLETION OF INPUT AND OUTPUT EVENTS

A semi-complete input or output event, i.e., one whose data transmission has been finished but which has not yet been deleted from \text{PA}, is completed during the corresponding wait statement by the instruction \text{comp!-event}. This instruction performs raising of input-output conditions accumulated during data transmission, raising of check condition in case of input, unlocking of the transmitted record, data set label handling, complete setting of the associated event variable generation and finally removing of the event from \text{PA}.

\textbf{Metavariables}

- \text{en} \quad \text{is-n} \quad \text{unique event name of the completed event}
- \text{cond-list} \quad \text{is-f-cond-list} \quad \text{list of input/output conditions to be raised}
- \text{unlock} \quad \text{the s-unlock component of the task-event specification of the completed event}
- \text{ecv-bcv} \quad \text{the s-eov-bcv component of the task-event specification of the completed event}

(43) \[ \text{comp!-event}(\text{en}) = \]
- \text{call-check-cond(}\text{s-check}(\text{te}_1))
- \text{test-ecv-bcv(}\text{s-eov-bcv}(\text{te}_1))
- \text{delete-task-event(}\text{en}, \text{NORMAL})
- \text{test-unlock(}\text{s-unlock}(\text{te}_1))
- \text{io-cond-iterate(}\text{en})

\textbf{where:}

- \text{te}_1 = \text{s-trn}(\text{PA})

\textbf{Ref.:}
- \text{call-check-cond 10-21(6q)}
- \text{delete-task-event 5-8(28)}

5. TASK 13
(44) \( \texttt{io-cond-iterate}(en) = \)
\( \texttt{is-\langle e\rangle s\texttt{-cond}(te_1) \rightarrow null} \)
\( T \rightarrow \texttt{io-cond-iterate-1}(s\texttt{-cond}(te_1),en) ; \)
\( \texttt{assign-status}(s\texttt{-ev}(te_1),\texttt{hintg-op}(1)) \)

where:
\( te_1 = s\texttt{-te}\texttt{en}(PA) \)

Ref.: \( \texttt{assign-status} \ 12-73(187) \)
\( \texttt{hintg-op} \ 9-10(40) \)

Note: If no conditions are to be raised, the status value of the event variable generation is left unchanged (normal completion); otherwise the status value 1 is assigned before condition raising (default value for abnormal completion, if no other status value is assigned explicitly by the raised on-units).

(45) \( \texttt{io-cond-iterate-1}(\texttt{cond-list},en) = \)
\( \texttt{is-\langle e\rangle (\texttt{cond-list}) \rightarrow null} \)
\( T \rightarrow \texttt{io-cond-iterate-1}(\texttt{tail(\texttt{cond-list})},en) ; \)
\( \texttt{call-ic-\texttt{-cond-1}(\texttt{cond_4})} \)

where:
\( \texttt{cond_4} = \mu(\texttt{head(\texttt{cond-list})};<\texttt{chif};\mu_0(<\texttt{s-oncount:length(\texttt{cond-list})}>),<\texttt{s-en*c-abn-ret:en}>)) \)

Ref.: \( \texttt{call-ic-cond-1} \ 10-28(72) \)

(46) \( \texttt{test-unlock}(unlock) = \)
\( \texttt{is-\langle e\rangle (unlock) \rightarrow null} \)
\( T \rightarrow \texttt{int-unlock-1}(unlock) \)

Ref.: \( \texttt{int-unlock-1} \ 11-46(126) \)

(47) \( \texttt{test-eov-bov}(eov-bov) = \)
\( \texttt{is-\langle e\rangle (eov-bov) \rightarrow null} \)
\( T \rightarrow \texttt{io-cond-return}(0,6(eov-bov;s-event)) \)

Ref.: \( \texttt{io-cond-return} \ 11-44(120) \)

14. TASK
5.0 THE DELAY STATEMENT

The execution of a delay statement results in a time delay for continuation of execution of the current task. The time to be waited for is noted in the s-delay component of the time and date part TD of the state, and the current task is set into the wait state. When this time is reached automatically all waiting tasks are reactivated.

Metavariables

\[ t \quad \text{is-delay-st} \quad \text{the interpreted delay statement} \]
\[ \text{time} \quad \text{is-intg-val} \quad \text{the time to be waited for} \]

(48) \[ \text{int-delay-st}(t) = \]
\[ \text{wait-delay(time-1)}; \]
\[ \text{time-1:note-delay(time)}; \]
\[ \text{time:eval-intg-exp}(s\text{-time}(t)) \]

Ref.: \[ \text{eval-intg-exp} \text{ 8-22(60)} \]

(49) \[ \text{note-delay(time)} = \]
\[ \text{PASS:time} \]
\[ \text{s-td}=\mu(\text{TD};\langle s\text{-delay:s\text{-delay(TD)} = \{time} \rangle) \]

where:
\[ \text{time} = s\text{-time(TD) + time} \]

(50) \[ \text{wait-delay(time)} = \]
\[ s\text{-time(TD)} \geq \text{time} \rightarrow \text{null} \]
\[ T \rightarrow \]
\[ \text{wait-delay(time)}; \]
\[ \text{set-wait-state-2(time)} \]

(51) \[ \text{set-wait-state-2(time)} = \]
\[ s\text{-time(TD)} \geq \text{time} \rightarrow \text{null} \]
\[ T \rightarrow s\text{-te}:\mu(\text{TD};\langle s\text{-wait:*} \rangle) \]

Note: Compare the instruction \text{set-wait-state} in section 5.3.1.
This chapter defines all those parts of PL/I which describe the flow of control of the PL/I machine between the single simple statements of a program within a single task. Excepted are only the activation and termination of on-units which interrupt the normal flow of control on occurrence of exceptional conditions and the asynchronous occurrence of attention interrupts (cf. chapter 10). Included are all those actions which, though not directly concerned with the flow of control itself, are performed by the PL/I statements described here, especially the handling of declarations at block entry and the passing of arguments at procedure call.

The first two sections define the mechanism of block activations: Section 6.1 the activation and termination of begin blocks, i.e. essentially those actions which are common to all kinds of block activations, and section 6.2 all those actions which are specific for procedure calls.

Section 6.3 defines the sequential flow of control through the nested structure of statement lists and if-statements within one block activation. It ends up with the case distinction of proper statements leading to the definitions of the individual types of statements in the different chapters of this document.

Section 6.4 defines the goto statement which interrupts the normal flow of control described in 6.1 and 6.3 continuing at some specified point of the program. Section 6.5 defines the iteration of statement lists by means of PL/I groups, and section 6.6 the inclusion of non-PL/I text.

6.1 BLOCK ACTIVATION

This section defines the interpretation of a begin block, i.e. the actions normally performed on establishing and terminating a block activation in the state of the PL/I machine. The special actions performed for a procedure call are defined in section 6.2, those for raising an on-unit in chapter 10, those for creating a task in chapter 5.

The interpretation of a begin block consists of the following five actions:
First initializing or partially inheriting the five local state components BA, FI, CS, CI, C and stacking the old ones into the dump D; second, unique qualification of names in the text of the block (cf. 6.1.1); third, the block prologue (cf. 6.1.2) which essentially consists of evaluating the declarations of the local declarations; fourth, the interpretation of the statement list of the block (cf. 6.3); and fifth, the block epilogue which reestablishes the former block activation (cf. 6.1.3).

Metavariables

<table>
<thead>
<tr>
<th>Metavariable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>t</td>
<td>the text of a begin block, procedure body or of the complete program</td>
</tr>
<tr>
<td>e</td>
<td>the local environment of a block activation (cf. 6.1.1)</td>
</tr>
<tr>
<td>dp</td>
<td>the declaration part of a block</td>
</tr>
<tr>
<td>bp</td>
<td>the body part of a block</td>
</tr>
<tr>
<td>cp</td>
<td>the condition part of a block</td>
</tr>
</tbody>
</table>
The block activation name BA
the epilogue information EI
the condition status CS
the dump D
the control information CI
the control C
the attribute directory AT
the denotation directory DN
identifier of a declaration
a declaration
unique name associated with a declaration
aggregate attribute of a variable declaration
scalar data attribute
denotation for a declaration
unique aggregate name of a proper variable
identifier list serving as qualified name
an expression
integer values
a selector

(1) \[
\text{int-block}(t) = \\
\begin{cases} \\
\text{s-ba} : & \mu(\text{EI} : \langle \text{s-block-act: BLOCK} \rangle, \langle \text{s-free-set: {}} \rangle, \langle \text{s-task-set: {}} \rangle) \\
\text{s-ei} : & \mu(\text{EI} : \langle \text{s-block-act: BLOCK} \rangle, \langle \text{s-free-set: {}} \rangle, \langle \text{s-task-set: {}} \rangle) \\
\text{s-d} : & \text{stack}(\langle \text{BA}, \text{EI}, \text{CS}, \text{D}, \text{CI}, \text{C} \rangle) \\
\text{s-ci} : & 0 \\
\text{s-c} : & \text{int-block-1}(t-1, e) \\
\text{t-1} : & \text{qualify-names}(t, e) \\
\text{e} : & \text{sh-e}(\text{s-decl-part}(t)) \\
\end{cases}
\]
for: is-block(t)

(2) \[
\text{stack}(\text{ba}, \text{ei}, \text{cs}, \text{d}, \text{ci}, \text{c}) = \\
\mu_0(\langle \text{ba} : \text{ba} \rangle, \langle \text{ei} : \text{ei} \rangle, \langle \text{cs} : \text{cs} \rangle, \langle \text{d} : \text{d} \rangle, \langle \text{ci} : \text{ci} \rangle, \langle \text{c} : \text{c} \rangle)
\]

2 6. FLOW OF CONTROL WITHIN A SINGLE TASK
6.1.1 UNIQUE QUALIFICATION OF NAMES

This section solves the scope rules of PL/I for identifiers and qualified names (identifier lists identifying components of structures). Whenever the text of a block (begin block, procedure body or program) is activated each of its local declarations gives a unique meaning to the declared identifier (and qualified names) wherever it occurs in that text, except in inner blocks if it is redeclared. This meaning of an identifier is to be fixed for all interpretations of parts of the text, even if the interpretation is performed during other nested block activations.

This scope rule of PL/I is realized in the formal definition by the following mechanism: When activating a block, each of its declarations is associated with a unique name n. From these unique names an auxiliary object, the so-called "local environment" e is constructed, which has the structure \( \{ \text{id:is-n} \mid \text{is-id(id)} \} \); it maps the locally declared identifiers to the unique names associated with their declarations. Subsequently, these unique names n are inserted into the text as s-n components wherever the corresponding identifiers or qualified names occur as s-id or s-id-list components. These inserted unique names are used on interpretation of the text to find the information about the meaning of names in the different state components of the PL/I machine. These unique names are left unchanged in the text of the block and its parts wherever it is copied and used; the only exception is that they are overwritten (and change thereby the meaning of names) if an identifier is redeclared in a nested block activation.
ABSTRACT SYNTAX AND INTERPRETATION OF PL/I

(7) collect-idl(idl,aggr) =
    is-array(aggr) → collect-idl(idl,s-elem(aggr))
    is-struct(aggr) →
    length(aggr)
    {idl} w.  collect-idl(idl<s-qual-elem(i,aggr)>,s-agg-elems(i,aggr))
    T → {idl}

Note: This function collects all qualified names of a structure declaration. For any other declaration it yields only the list consisting of the declared identifier itself.

6.1.2 BLOCK PROLOGUE

The block prologue establishes a unique block activation name BA identifying the current block activation, enters the text of all local declarations under their associated unique names into the attribute directory AT, merges the condition part of the block into the block prefix part of the condition status CS and, as the main action, evaluates the denotations of the local declarations and enters them under the associated unique names into the denotation directory DH.

Since the evaluation of the denotations of automatic and defined declarations requires expression evaluation and these expressions may contain locally declared identifiers, it is necessary to evaluate the denotation of an automatic or defined declaration not before all denotations of those declarations have been evaluated on which its own evaluation depends. These dependencies are defined in section 6.1.2.1, resulting in a predicate is-ready-for-exec.

The denotations of the declarations are evaluated one after the other, not simultaneously but - apart from the above mentioned dependencies - in arbitrary order. This is realized by entering the instruction upd-decl-dn for all declarations which are ready for execution into the control; the first of these instructions which is selected for execution overwrites the control, (thereby prohibiting all others from execution), evaluates its declaration completely and finally enters again upd-decl-dn into the control for all declarations which then are ready for execution. This ends up when either all declarations are evaluated, or there is no more an independent declaration (which is an erroneous case).

(8) prologue(dp,bp,cp,e) =
    upd-dn(e,bp);
    upd-cs(s-cs(cp),s-no(cp));
    upd-at(dp,e);
    upd-ba(ba);
    ba:un-name

Ref.: upd-cs 10-2(1)
      un-name 3-10(20)

(9) upd-ba(ba) =
    s-ba:ba

4 6. FLOW OF CONTROL WITHIN A SINGLE TASK
30 April 1969

ABSTRACT SYNTAX AND INTERPRETATION OF PL/I

6. FLOW OF CONTROL WITHIN A SINGLE TASK

(10) $\text{upd-at}(\text{dp}, e) =$

$$s\text{-at}:\mu(\text{AT} : \langle \text{id}(e) : \text{id}(\text{dp}) \rangle | -\text{is-}\text{Q}\cdot\text{id}(e))$$

(11) $\text{upd-dn}(e, bp) =$

$$\text{is-BLOCK}\cdot s\text{-block-act}(AT) & (\exists \text{id})(-\text{is-}\text{Q}\cdot\text{id}(e) & \text{is-PARAMs-scope}(\text{attr}_0)) \rightarrow$$

error

$T \rightarrow$

$$s\text{-ci}:\nu(<s\cdot C>)$$

$s\cdot \text{upd-dn-1}(e_1, bp)$

where:

$\text{attr}_0 = \text{id}(e)(\text{AT})$

$e_1 = \delta(e;\text{id} | \text{is-PARAMs-scope}(\text{attr}_0))$

Note: The control $C$ is stacked into $CI$ intermediately in order to avoid the loss of its information during the overwriting mechanism by the instruction $\text{upd-decl-dn}$ below. It is restored by the instruction $\text{upd-dn-1}$ when all denotations have been evaluated.

(12) $\text{upd-dn-1}(e, bp) =$

$$\text{is-}\text{Q}(e) \rightarrow$$

$$s\cdot\text{ci}:\nu$$

$$s\cdot\text{ci}:s\cdot\text{c}(\text{CI})$$

$(\exists \text{id})(\text{is-ready-for-exec}(\text{id}, e, \text{AT}, \text{DN})) \rightarrow$

null;

{$\text{upd-decl-dn}(id, e, bp) | \text{is-ready-for-exec}(id, e, \text{AT}, \text{DN})$}

$T \rightarrow$ error

(13) $\text{upd-decl-dn}(id, e, bp) =$

$$s\cdot\text{ci}:\text{upd-dn-1}(\delta(e;\text{id}), bp);$$

$\text{upd-decl-dn-1}(id(e), \text{den});$

$\text{den}:\text{eval-den}(id, \text{attr}_0, \text{body}_0)$

where:

$\text{attr}_0 = \text{id}(e)(\text{AT})$

$\text{body}_0 = (\text{is-}\text{Q}\cdot\text{body}(\text{attr}_0) \rightarrow q,\text{id})$

$\text{is-}\text{Q}(\text{body}(\text{attr}_0)(bp)) \rightarrow \text{error},$

$T \rightarrow \text{body}(\text{attr}_0)(bp))$

Note: By overwriting the control this instruction prevents all parallel ones from being executed before it has finished its own actions.

(14) $\text{upd-decl-dn-1}(n, \text{den}) =$

$$s\cdot\text{dg}:\mu(\text{DP} : \langle n : \text{den} \rangle)$$
6.1.2.1 Order of evaluating declarations

An automatic or defined declaration (requiring expression evaluation) is ready for execution when all those declarations have been evaluated (and their denotations been entered into the denotation directory), on which its own evaluation depends. This includes first the declarations of all identifiers occurring in extents and initials of its aggregate attribute. Second, it includes the declarations of implied pointers of based references, of bases of defined references and of alternatives of generic references occurring in expressions to be evaluated. Third, it includes recursively the declarations of all identifiers occurring in the references of these implied pointers, defined bases and generic alternatives, and so on. The following predicates define all these dependencies of references.

Metavariables

Besides the metavariables listed in section 6.1, the following metavariable is used here:

ref \text{is-ref} \lor \text{is-spec-init-elem}

A reference or a special initial element (of course also the entry or label constants referenced by special initial elements have to be evaluated before they can be assigned as initial values to entry or label variables).

\begin{align*}
\text{(15)} \quad & \text{is-ready-for-exec(id,e,at,dn)} = \\
& \neg \text{id}(e) \land \\
& (\neg \text{is-AUTO}\in\text{stg-cl}(\text{attr}_0) \land \neg \text{is-defined}(\text{attr}_0) \lor \\
& (\text{vref})(\text{is-dep}(\text{s-aggr}(\text{attr}_0),\text{ref},\text{at}) \land \\
& \neg \text{is-}\Omega(\text{den}_1) \lor \\
& (\text{is-based} \lor \text{is-generic} \lor \text{is-BUILTIN}(\text{attr}_1)))
\end{align*}

where:

\begin{align*}
\text{attr}_0 &= \text{id}(e)(\text{at}) \\
\text{attr}_1 &= \text{s}-\text{n}(\text{ref})(\text{at}) \\
\text{den}_1 &= \text{s}-\text{n}(\text{ref})(\text{dn})
\end{align*}

\begin{align*}
\text{(16)} \quad & \text{is-dep}(\text{aggr,ref,at}) = \\
& (\exists \text{ref-1})(\text{is-dep-1}(\text{aggr,ref-1}) \land \text{is-dep-2}(\text{ref-1,ref,at}))
\end{align*}

\begin{align*}
\text{(17)} \quad & \text{is-dep-1}(\text{aggr,ref}) = \\
& (\exists s)(\text{ref} = s(\text{aggr}) \land \\
& \neg (\exists s-1,s-2)(s = s-1\ast\text{area}\ast s-2))
\end{align*}
AESTRACT SYNTAX AND INTERPRETATION OF PL/I

(18) \text{is-dep-2}(\text{ref-1},\text{ref-2},\text{at}) =
\begin{align*}
&\exists \text{ref-2} = s(\text{ref-1}) \land \text{is-based}(\text{attr}_1) \land \text{is-0-s-pptr}(\text{ref-1}) \land \\
&\text{ref-2} = s-ptr(\text{attr}_1) \land \text{is-defined}(\text{attr}_1) \land \\
&\text{ref-2} = s-base(\text{attr}_1) \land \exists \text{ref-2} = s-pos(\text{attr}_1)) \land \text{is-generic}(\text{attr}_1) \land \\
&\neg \text{is-<>-s-ap}(\text{ref-1}) \land \text{ref-2} = \text{generic-sel}(\text{attr}_1,\text{attr-list}_1) \land \\
&\exists \text{ref-3} = \text{is-generic-dep}(\text{ref-1},\text{ref-2},\text{ref-3},\text{at}) \land \\
&\exists \text{ref} = \text{is-dep-2}(\text{ref-1},\text{ref},\text{at}) \land \text{is-dep-2}(\text{ref},\text{ref-2},\text{at})
\end{align*}

where:
\begin{align*}
\text{attr}_1 &= s-n(\text{ref-1})(\text{at}) \\
\text{attr-list}_1 &= \text{attr-expr-list}(\text{arg-list}*\text{s-ap}(\text{ref-1}),\text{at})
\end{align*}

Ref.:
\begin{align*}
&\text{generic-sel } 6-31(77) \\
&\text{attr-expr-list } 6-35(96) \\
&\text{arg-list } 8-4(6)
\end{align*}

(19) \text{is-generic-dep}(\text{ref-1},\text{ref-2},\text{ref-3},\text{at}) =
\begin{align*}
&\exists i, j, da-1, da-4) (\text{is-entry}(da-1) \land \\
&(da-1 = \text{attr}_1 \land \text{is-scalar-da}(da-1,s-aggr(\text{attr}_1))) \land \\
&1 \leq i \leq \text{length}\text{-}\text{ap}(\text{ref-1}) \land 1 \leq j \leq \text{length}\text{-}\text{elem}(i,s-ap(\text{ref-1})) \land \\
&\text{ref-3} = \text{elem}(j,\text{elem}(i,s-ap(\text{ref-1}))) \land \text{is-generic}(\text{attr}_3) \land \\
&\neg \text{is-<>-s-ap}(\text{ref-3}) \land \neg \text{is-<>-s-descr-list}(i,da-7) \land \\
&1 \leq j \leq \text{length}\text{-}\text{nest-descr-list}(i,da-1) \land \text{is-entry}(da-4) \land \\
&\text{is-scalar-da}(da-4,s-aggr\text{-}\text{elem}(j,nest\text{-}\text{descr-list}(i,da-1))) \land \\
&\text{ref-2} = \text{generic-sel}\text{-}\text{list}(\text{attr}_3,\text{s-descr-list}(da-4))
\end{align*}

where:
\begin{align*}
\text{attr}_1 &= s-n(\text{ref-1})(\text{at}) \\
\text{attr}_3 &= s-n(\text{ref-3})(\text{at})
\end{align*}

Ref.:
\begin{align*}
&\text{generic-sel } 6-31(77)
\end{align*}

Note: This predicate describes the dependency of a function reference \text{ref-1} with an argument \text{ref-3}, which is an argument free generic name, on the selected generic alternative \text{ref-2}. The generic selection is performed according to the relevant parameter descriptor in the declaration of \text{ref-1}.

(20) \text{nest-descr-list}(i,da) =
\begin{align*}
&\neg \text{is-entry}(da) \rightarrow \text{error} \\
&i = 1 \rightarrow s\text{-}\text{descr-list}(da) \\
&T \rightarrow \text{nest-descr-list}(i - 1,s\text{-}\text{da-s}\text{-ret-type}(da))
\end{align*}

(21) \text{is-scalar-da}(da,aggr) =
\begin{align*}
&\text{is-array}(aggr) \rightarrow \text{is-scalar-da}(da,s\text{-}\text{elem}(aggr)) \\
&\text{is-struct}(aggr) \rightarrow \\
&(3i)(1 \leq i \leq \text{length}(aggr) \land \text{is-scalar-da}(da,s-aggr\text{-}\text{elem}(i,aggr))) \\
&\text{is-scalar}(aggr) \rightarrow da = s\text{-}\text{da}(aggr) \\
&T \rightarrow \top
\end{align*}

6. FLOW OF CONTROL WITHIN A SINGLE TASK
6.1.2.2 Denotations of the single declarations

The instruction `eval-den` evaluates the denotation of a single declaration. This evaluation depends very much on the type of the declaration. In the case of an automatic variable it also allocates and initializes storage for the variable (which has already been done by the prepass in the case of static variables).

\[(22) \text{eval-den}(id, attr, body) = \]
\[
\begin{align*}
\text{(is-STATIC } & \text{ or is-CL) } \times \text{stg-cl}(attr) \times (\text{is-file-const } \text{ or is-attn })(attr) \rightarrow \\
\text{PASS:s-den(attr)} & \\
\text{is-AUTO} & \times \text{stg-cl}(attr) \rightarrow \\
\text{PASS(b):} & \\
\text{initialize}
\begin{cases}
\text{gen, s-aggr(attr)};
\text{gen:allocate-auto(b, eva)};
\text{b:un-name,}
\text{eva:eval-aggr(s-aggr(attr))}
\end{cases}
\end{align*}
\]
\[
\begin{align*}
\text{is-defined(attr) & (W)} & (\text{-is-Q} \times \text{init-w(attr) } \text{ or is-} \times \text{init-e(attr)}) \rightarrow \\
\text{eval-aggr(s-aggr(attr))} & \\
\text{is-entry-const(attr) & is-INT*scope(attr) & is-Q(b) & is-Q(body) } \rightarrow \\
\text{PASS:} & \\
\text{is-entry-const(attr) & is-EXT*scope(attr) & is-Q(b) & is-Q(body) } \rightarrow \\
\text{PASS:s-den(attr)(DN)} & \\
\text{is-label-const(attr) } \rightarrow \text{PASS:} & \\
\text{is-format-const(attr) } \rightarrow \\
\text{PASS:} & \\
\text{(is-based } & \text{ or is-generic } \text{ or is-BUILTIN } \text{ or is-COND)(attr) } \rightarrow \text{null}
\end{align*}
\]

\(T \rightarrow \text{error}\)

Where:
\[
\begin{align*}
o111_1 & = s-cn*s-cond-part(attr) \\
o111_4 & = s-no*s-cond-part(attr)
\end{align*}
\]

For: \((\text{is-body } \text{ or is-Q)} \text{ (body)\)}

Ref.:
\[
\begin{align*}
\text{initialize} & 7-15(49) \\
\text{allocate-auto} & 7-10(24) \\
\text{un-name} & 3-10(20) \\
\text{merge-pref} & 10-3(5)
\end{align*}
\]

6.1.2.3 Evaluation of aggregate attributes

The aggregate attributes occurring in the program text in declarations of variables contain as extents (i.e. array bounds, string lengths and area sizes) usually expressions. For allocation of storage however and other purposes during the interpretation aggregate attributes with evaluated extents ("evaluated aggregate attributes", satisfying the predicate is-eva) are needed. This
evaluation of extent expressions of aggregate attributes is performed by the instruction eval-aggr.

In special cases, where the extents are restricted to be (possibly signed) integer constants only, the evaluation is performed by the function eval-aggr-1.

Besides, the qualifying identifiers of structures, the additional information in the scalar data attributes of entry, label and offset variables and the initial attributes are removed on evaluation of an aggregate attribute.

(23) \[ \text{eval-aggr}(\text{aggr}) = \]
\[ \text{is-array}(\text{aggr}) \rightarrow \]
\[ \text{pass}(\text{eva}); \]
\[ \text{s-lbd}(\text{eva}) : \text{eval-intg-expr}(\text{s-lbd}(\text{aggr})), \]
\[ \text{s-ubd}(\text{eva}) : \text{eval-intg-expr}(\text{s-ubd}(\text{aggr})), \]
\[ \text{s-elem}(\text{eva}) : \text{eval-aggr}(\text{s-elem}(\text{aggr})) \]
\[ \text{is-struct}(\text{aggr}) \rightarrow \]
\[ \text{pass}(\text{eva}); \]
\[ \{ \text{s-aggr} \cdot \text{elem}(i) \}(\text{eva}) : \text{eval-aggr}(\text{s-aggr} \cdot \text{elem}(i, \text{aggr})) \mid \]
\[ 1 \leq i \leq \text{length}(\text{aggr}) \]
\[ \text{is-scalar}(\text{aggr}) \rightarrow \]
\[ \text{pass}(\text{eva}); \]
\[ \text{s-da}(\text{eva}) : \text{eval-da}(\text{s-da}(\text{aggr})), \]
\[ \text{s-dens}(\text{eva}) : \text{pass}(\text{s-dens}(\text{aggr})) \]

Ref.: eval-intg-expr 8-22(60)

(24) \[ \text{eval-da}(\text{da}) = \]
\[ \text{is-string}(\text{da}) \rightarrow \]
\[ \text{pass}(\text{eda}); \]
\[ \text{s-length}(\text{eda}) : \text{eval-intg-expr}(\text{s-length}(\text{da})), \]
\[ \text{eda} : \text{pass}(\text{eda}) \]
\[ \text{is-area}(\text{da}) \rightarrow \]
\[ \text{pass}(\text{eda}); \]
\[ \text{s-size} : \text{eval-intg-expr}(\text{s-size}(\text{da})) \]
\[ T \rightarrow \text{pass} : \text{eval-da-1}(\text{da}) \]

Ref.: eval-intg-expr 8-22(60)
(25) \begin{align*}
\text{eval-aggr-1}(\text{aggr}) &= \\mu_0(<s-\text{lb}d:*>,<s-\text{ub}d:*>,<s-\text{elem}:\text{eval-aggr-1}\text{e}lem(\text{aggr})>) \\
\text{is-array}(\text{aggr}) \land \text{is-scalar}(\text{aggr}) &\rightarrow \\
\text{is-struct}(\text{aggr}) &\rightarrow \text{LIST} \mu_0(<s-\text{aggr}:\text{eval-aggr-1}\text{e}lem(i,\text{aggr})>) \\
\text{is-struct}(\text{aggr}) &\rightarrow \text{LIST} \mu_0(<s-\text{aggr}:\text{eval-aggr-1}\text{e}lem(i,\text{aggr})>) \\
\text{is-scalar}(\text{aggr}) &\rightarrow \mu_0(<s-\text{da}:\text{eval-da-1}\text{e}lem(\text{aggr})>,<s-\text{dens}\text{e}:\text{dens}(\text{aggr})>) \\
\end{align*}

(26) \begin{align*}
\text{eval-da-1}(\text{da}) &= \\
\text{is-string}(\text{da}) \land \text{is-scalar}(\text{da}) &\rightarrow \text{da} \\
\text{is-string}(\text{da}) &\rightarrow \mu_1(<s-\text{length}:\text{intg-val}\text{e}:\text{length}(\text{da})>) \\
\text{is-area}(\text{da}) \land \text{is-scalar}(\text{da}) &\rightarrow \text{da} \\
\text{is-area}(\text{da}) &\rightarrow \mu_1(<s-\text{size}:\text{intg-val}\text{e}:\text{size}(\text{da})>) \\
\text{is-entry}(\text{da}) &\rightarrow \text{ENTRY} \\
\text{is-label-da}(\text{da}) &\rightarrow \text{LABEL} \\
\text{is-offset}(\text{da}) &\rightarrow \text{OFFSET} \\
\text{T} &\rightarrow \text{da} \\
\end{align*}

(27) \begin{align*}
\text{intg-val}(\text{expr}) &= \\
-\text{is-size-cond}(\text{BINTG-EDA},\text{intg-val-1}(\text{expr})) &\rightarrow \text{intg-val-1}(\text{expr}) \\
\text{T} &\rightarrow \text{error} \\
\end{align*}

Ref.: is-size-cond 9-8(28)

(28) \begin{align*}
\text{intg-val-1}(\text{expr}) &= \\
\text{is-intg-const}(\text{expr}) &\rightarrow s-v(\text{expr}) \\
\text{is-signed-intg}(\text{expr}) \land \text{is-PLUS*OPR}(\text{expr}) &\rightarrow s-v*OPR(\text{expr}) \\
\text{is-signed-intg}(\text{expr}) \land \text{is-MINUS*OPR}(\text{expr}) &\rightarrow -s-v*OPR(\text{expr}) \\
\text{T} &\rightarrow \text{error} \\
\end{align*}

(29) \begin{align*}
\text{is-signed-intg} &= \\
\text{is-intg-const} \lor \\
<s-\text{opr}:\text{is-PLUS} \lor \text{is-MINUS}, \text{is-opr}:\text{is-intg-const}> \\
\end{align*}
5.1.3 BLOCK EPILOGUE

The block epilogue terminates abnormally all tasks attached during the block activation, if they are yet active (cf. section 5.2) and frees the storage of all local automatic variables. The information about the tasks to be terminated and the variables to be freed is found in the epilogue information. Finally the epilogue reinstalls the six local state components $E_A$, $E_I$, $C_S$, $D$, $C_I$, $C$ of the former block activation which were stacked in the dump $D$.

(31) $\text{epilogue} =$

\begin{verbatim}
usstack;
free-blockend;
term-tasks
\end{verbatim}

Ref.: $\text{term-tasks} ~5-6(17)$

(32) $\text{free-blockend} =$

\begin{verbatim}
s-g:el-free(ptr-set$_4$,S)
S-ag:$\{AG:s-free-set(EI)}$
s-ei:$\{EI:s-free-set(\{\})}$
\end{verbatim}

where:

\begin{verbatim}
ptr-set$_4 = \{s-ppehead(b(AG) | b \in s-free-set(EI))\}
\end{verbatim}

Ref.: $\text{el-free} ~3-17(62)$

(33) $\text{usstack} =$

\begin{verbatim}
s-ba:$s-ba(D)$
s-ci:$s-ci(D)$
s-ch:$s-ch(D)$
s-d:$s-d(D)$
s-cis:$s-cis(D)$
\end{verbatim}

6.2 PROCEDURE CALL

A procedure is activated either by a call statement during sequential interpretation of a statement list (cf. 6.3), or by a call statement during initialization of allocated storage (cf. 7.2) - in both cases by the instruction int-call-st, or by a function reference during expression evaluation (cf. 8.2) - by the instruction eval-function. This section describes all those actions concerned with the activation of a procedure (shortly "procedure call") which differ from the actions normally performed on block activation (cf. 6.1).

Section 6.2.1 defines all actions performed by a call statement or function reference in the calling block activation except the evaluation of arguments. It
ends up with the instruction int-call-1 which branches into three different cases:
The proper non-task procedure call defined in section 6.2.2 which describes all
actions performed in the called block activation: the task call defined in section
5.1; and the call of a builtin function which had been assigned as value to an
entry variable (cf. chapter 12).

Section 6.2.3 defines the passing of arguments to parameters, their evaluation
in the calling block activation and their installation as denotations of the
parameters in the called procedure. Section 6.2.4 defines the return from a
procedure by means of the return statement including the passing of the function
value in the case of a function reference.

It is possible in PL/I to declare a family of entry references by a generic
declaration and to use in a call statement or function reference the family name
instead of an individual entry reference. In this case the selection of the
individual members, governed by the types of the arguments, is performed before
all other actions of the call. This selection is described in section 6.2.5.

Section 6.2.6 defines the fetch and release statement, whose optimizing actions
are essentially implementation defined.

6.2.1 CALL-STATEMENT AND FUNCTION REFERENCE

Both the interpretation of a call statement and of a function reference consist
of evaluation of the called entry reference by the instruction eval-entry-expr
defined in section 6.2.2. (in the case of function reference this is part of the
instruction eval-entry-expr-1 or eval-pos-1) and of the call of the evaluated
entry by the instruction int-call.

The only difference between a procedure call by a call statement and by a
function reference is that in the latter case provisions are made for the
returning of the function value. For this purpose, before the call a unique
aggregate name b is created and passed as argument of the instruction int-call to
the called procedure; this aggregate name is entered as s-funct-den component
into the epilogue information EI of the called procedure (cf. 6.2.2) and is used by the
return statement for allocation of an appropriate dummy variable for the function
value (cf. 6.2.4). After return, the calling block activation can access the
function value via the aggregate name, test its data attribute against the
declared return type and free the storage of the dummy (see the instruction
return-free below).

The instruction int-call performs all actions which are common to all types of
call of an entry: Evaluating of arguments (cf. 6.2.3), raising of check condition
for the called entry (if applicable) before the call and for the passed non-dummy
arguments after the call. It ends up with the case distinction between the proper
procedure call (if the value of the called entry is an entry denotation and there is
no task option specified) defined in section 6.2.2, the task call (if the value
of the called entry is an entry denotation and there is a task option) defined in
section 5.1, and a builtin call (if the value of the called entry is a builtin
denotation) defined in section 12.4.

Metavariables

<table>
<thead>
<tr>
<th>symb.</th>
<th>definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>t</td>
<td>is-call-st</td>
</tr>
<tr>
<td>n</td>
<td>is-n</td>
</tr>
<tr>
<td>den</td>
<td>is-entry-den v is-builtin-den</td>
</tr>
</tbody>
</table>
entry is-entry

description of called entry constant or data
attribute of called entry variable

expr-list is-expr-list

list of argument expressions to be passed

descrl is-descr-list v is-

parameter descriptor list of called entry:
s-descr-list(entry)

argl

list of evaluated arguments (cf. 6.2.3.)

b is-n v is-o

in case of function reference: unique
aggregate name for function value; in case of
call statement: Φ (used also as an indicator for
this case distinction)

ref is-ref v is-o

in case of "first call": called entry
reference used for raising of check condition;
in case of "repeated call": Φ, no check
condition raised (see the note at the
instruction int-call-st)

pa-opt is-pa-opt v is-o

in case of task call: specified task option
s-pa-opt(t); else: Φ

aggr is-descr-scalar

return-type of called entry: s-ret-type(entry)

ba is-ba

block activation name BA

d is-d

dump Φ

(34) int-call-st(t) =

int-call(n,s-arg-list(t),s-descr-list(da),Φ,check-ref,t,s-pa-opt(t));

where:

da = s-da-aggr-ref(s-entry(t),AT)
attr-arg-list = attr-expr-list(s-arg-list(t),AT)
check-ref = (is-<>s-ap-s-entry(t) → Φ,
          is-generic(attrφ) → s-entry(t),
          is-<>s-ap(ref-g0) → Φ,
          T → ref-g0)
attrφ = s-<>s-entry(t)(AT)
ref-g0 = generic-sel(attrφ,attr-arg-list)

Ref.: eval-entry-exp 8-17(47)
aggr-ref 8-25(71)

Note: Since the check condition is raised only for the name of the called entry
variable or entry constant, only the "first call" of a reference containing
more than one argument list raises the check condition. All "repeated
calls" (calls of the returned entry values) have no names of entry
variables or entry constants and raise no check condition. This, in
connection with generic selection, is reflected by the case distinction in
the abbreviation check-ref: Φ denotes that the condition has been raised
already by eval-entry-exp (see the instruction check-entry below).
(35) \texttt{eval-function}(s, \texttt{expr-list}, \texttt{entry}, \texttt{ref}) =
\hspace{1cm}
\begin{align*}
&\text{(is-}\texttt{desc}-\texttt{text}(CI) \text{ v is-allocate-st \texttt{prop-st} \texttt{take-st}(s, \texttt{index}(CI), \texttt{s-text}(CI))) \& \\text{is-s-reducible}(\texttt{entry}) \rightarrow \\
&\text{error} \\
&T \rightarrow \\
&\text{return-free}(b, s, \texttt{ret-type}(\texttt{entry})); \texttt{int-call}(n, \texttt{expr-list}, s, \texttt{descr-list}(\texttt{entry}), b, \texttt{ref}, 0); \texttt{b}: \texttt{yn}.= \texttt{name} \\
\end{align*}

Ref.: take-st 6-39(110)

\texttt{yn}.= \texttt{name} 3-10(20)

Note: Only reducible function calls are allowed within block prologues and allocate statements.

(36) \texttt{return-free}(b, \texttt{aggr}) =
\hspace{1cm}
\begin{align*}
&\text{gen-aggr-match}(s, \texttt{eva}(\texttt{gen}_3), \texttt{aggr}) \rightarrow \\
&\text{pass}(\texttt{op}); \texttt{free}(b); \texttt{op}: \texttt{gen-op}(\texttt{gen}_3) \\
&T \rightarrow \text{error} \\
\end{align*}

where:
\hspace{1cm} \texttt{gen}_3 = \texttt{head}\texttt{b}(\texttt{AG})

Ref.: free 7-18(58)

\texttt{gen-op} 8-22(59)

(37) \texttt{int-call}(n, \texttt{expr-list}, \texttt{descr}, b, \texttt{ref}, \texttt{pa-opt}) =
\hspace{1cm}
\begin{align*}
&\text{check-arg-list}(\texttt{expr-list}, \texttt{argl}, a, \texttt{pa-opt}); \texttt{int-call-1}(n, \texttt{PP}), \texttt{argl}, b, \texttt{pa-opt}); \texttt{check-entry}(\texttt{ref}); \\
&\text{argl}: \texttt{eva-arg-list}(\texttt{expr-list}, \texttt{descr})
\end{align*}

(38) \texttt{check-entry}(\texttt{ref}) =
\hspace{1cm}
\begin{align*}
&\text{is-}\texttt{U}(\texttt{ref}) \rightarrow \texttt{null} \\
&T \rightarrow \texttt{call-check-cond}(\texttt{ref}) \\
\end{align*}

Ref.: \texttt{call-check-cond} 10-21(69)
(39) \( \text{check-arg-list}(\text{expr-list},\text{argl},n,\text{pa-opt}) = \)
\( \text{is-O}(\text{pa-opt}) \land \text{is-entry-den}\cdot n(DN) \rightarrow \)
\( \text{call-check-cond}(\text{argl},\text{check-cond}(\text{expr-list},\text{argl})) \)
\( T \rightarrow \text{null} \)

Ref.: \( \text{is-entry-den} 3-12(24) \)
\( \text{call-check-cond} 10-21(64) \)

(40) \( \text{arg-check-list}(\text{expr-list},\text{argl}) = \)
\( \text{is-<>}(\text{expr-list}) \rightarrow <> \)
\( \text{is-CFN\&-type}\cdot \text{head}(\text{argl}) \rightarrow \)
\( \text{<head}(\text{expr-list})\rightarrow \text{arg-check-list}(\text{tail}(\text{expr-list}),\text{tail}(\text{argl})) \)
\( T \rightarrow \text{arg-check-list}(\text{tail}(\text{expr-list}),\text{tail}(\text{argl})) \)

(41) \( \text{int-call-1}(\text{den},\text{argl},b,\text{pa-opt}) = \)
\( \text{is-entry-den}(\text{den}) \land \text{is-O}(\text{pa-opt}) \land \text{is-active-ba}(\text{sb-}\text{ba}(\text{den}),\text{BA},\text{D}) \rightarrow \)
\( \text{call-proc}(\text{den},\text{argl},b) \)
\( \text{is-entry-den}(\text{den}) \land \text{is-active-ba}(\text{sb-}\text{ba}(\text{den}),\text{BA},\text{D}) \rightarrow \text{call-task}(\text{den},\text{argl},\text{pa-opt}) \)
\( \text{is-built-in-den}(\text{den}) \land \text{is-O}(b) \rightarrow \text{call-built-in}(\text{s-}\text{id}(\text{den}),\text{s-da-list}(\text{den}),\text{argl},b) \)
\( T \rightarrow \text{error} \)

Ref.: \( \text{is-entry-den} 3-12(24) \)
\( \text{call-task} 5-2(1) \)
\( \text{is-built-in-den} 3-12(27) \)
\( \text{call-built-in} 12-11(29) \)

(42) \( \text{is-active-ba}(\text{ba-1},\text{ba},\text{d}) = \)
\( \text{ba-1} = \text{ba} \rightarrow T \)
\( \text{is-O}(\text{d}) \rightarrow F \)
\( T \rightarrow \text{is-active-ba}(\text{ba-1},\text{sb-}\text{ba}(\text{d}),\text{s-}\text{d}(\text{d})) \)

Note: Since it is possible to preserve the denotation of an entry constant via
the storage of an entry variable beyond the life time of the block
activation in which it was declared, the dump has to be scanned whether
that block activation is yet active.

6.2.2 NON-TASK PROCEDURE CALL

This section defines the proper non-task procedure call and the actions
performed in the called procedure.
Metavariables:

den  is-entry-den  denotation of called entry
argl  is-n  is-n  list of evaluated arguments passed to the called entry (cf. 6.2.3)
b     is-n  is-n  aggregate name for function value in case of function call; else 0
t     is-body  called procedure body
id    is-id  identifier specifying called entry point
e     is-nv  is-nv  local environment of called block activation (cf. 6.1.1)
aggr  is-descr-scalar  return type of called entry point
st-list  is-st-list  statement list of called procedure body
indl  is-index-list  index list localizing the called entry point in the statement list of the body.

(43) call-proc(den, argl, b) =
    s-ba:=
    s-ci:=ei_1
    s-ci:=s-ci
    s-ci:=stack(PA, PI, CS, D, S1, G)
    s-ci:=
    s-ci:=s-body(t, s-id(den), argl, e);
    t:=qualify-names(s-body(den), e);
    ei:=s-decl-part•s-body(den))
    where:
    ei_1 = 
    dummy-set_2 = {s-deq-elem(i, argl) | 1 ≤ i ≤ length(argl) & is-DUMMY•s-type•elem(i, argl))
    main_1 = (is-0 (0) -> *, t -> 0)
    cs_1 = 
    Ref.: stack 6-2(2)
    qualify-names 6-3(5)
    ak-e 6-3(4)

Note: The dummy arguments are to be freed by the epilogue of the called procedure and therefore entered into the free set of E1. Only for the initial call of the program (cf. 4.2) the s-main flag of E1 is set to * signalizing that the finish condition is to be raised on its return (cf. 6.2.4).
ABSTRACT SYNTAX AND INTERPRETATION OF PL/I

(44) \[ \text{int-body}(t, \text{id}, \text{arg1}, e) = \]
\[ \quad \text{int-return-st}(u); \]
\[ \quad \text{int-proc-st-list}(s-st-list(t), s-st-loc(ep_1)); \]
\[ \quad \text{prologue}(s-decl-part(t), s-body-part(t), s-cond-part(t), e); \]
\[ \quad \text{install-arg-list}(\text{arg1}, s-param-list(ep_1), s-decl-part(t)); \]
\[ \quad \text{end-ei}(s-ret-type(ep_1))\]

where:
\[ \text{ep}_1 = \text{id} \cdot \text{s-entry-part}(t) \]

Ref.: prologue 6-4(6)

Note: It is allowed only in the case of a non-function call of a procedure that the flow of control comes to the end of the statement list of the procedure. In this case a return statement without an expression is interpreted.

(45) \[ \text{end-ei}(\text{aggr}) = \]
\[ \quad \text{is-\text{\#s-funct-den}(\text{\#})} \rightarrow \text{null} \]
\[ \quad T \rightarrow \text{\#-ei}(:\#; <s-ret-type:aggr>)\]

(46) \[ \text{int-proc-st-list}(\text{st-list}, \text{indl}) = \]
\[ \quad \text{\#-ei}:\#(\text{<s-text:st-list>, <s-c:G>}) \]
\[ \quad \text{\#-e:go-to-3}(\text{indl})\]

Ref.: go-to-3 6-45(125)

6.2.3 ARGUMENT PASSING

The expressions occurring as arguments in the call statement or function reference are tested against the corresponding parameter descriptors in the declaration of the called entry constant or entry variable. Depending on this test one of the following three ways of argument passing is applied:

1. passing of controlled generation list
2. passing of generation
3. passing of a dummy variable.

The relation to be satisfied between the argument expression and the parameter descriptor and the evaluation of the passed argument is defined for these three cases in sections 6.2.3.1, 6.2.3.2, 6.2.3.3, respectively.

In any of these cases the instruction eval-arg constructs an argument having the structure
\[ \text{is-arg} = (<s\text{-den:is-n}, <s\text{-type:is-CTL} \lor \text{is-GEN} \lor \text{is-DUMMY}>). \]

The s-den component is the aggregate name passed to the parameter, the s-type component denotes which of the three types of passing applies.

In the called procedure, the instruction install-arg tests these passed arguments against the declarations of the corresponding parameters and installs, if the test was successful, the passed aggregate names as denotations of the parameters. This is defined in section 6.2.3.4.

5. FLOW OF CONTROL WITHIN A SINGLE TASK
Metavariables

expr-list is-expr-list  list of expressions to be passed as arguments
expr  is-expr  expression to be passed as argument
ref  is-ref  reference to be passed as argument
descrl is-descr-list \* is-*  list of parameter descriptors
descr  is-descr  parameter descriptor
argl  is-argl  list of evaluated arguments
arg  is-arg  evaluated argument (see above)
b  is-b  unique aggregate name of passed argument: s-den(arg)
gen  is-gen  generation of passed argument
type  is-CTL \* is-GEN \* is-DUMMY  type of passed argument: s-type(arg)
param-list is-id-ref-list  list of parameter names
n  is-n  unique name associated with parameter
attr  is-attr  parameter declaration
aggr  is-aggr  aggregate attribute of argument, parameter descriptor or parameter
eva  is-eva  evaluated aggregate attribute of passed argument generation
da  is-da  scalar data attribute
at  is-at  the attribute directory AT
dp  is-decl-part  declaration part of called procedure body
stg-cl is-CTL \* is-AUTO \* is-STAT  storage class of argument
dop  "dummy operand" (cf. 6.2.3.3)
dopl  list of "dummy operands"
dal  is-da-list  list of scalar data attributes
i, j  is-intg-val  integer values
o  is-selector  a selector
(47) \[ \text{eval-arg-list}(\text{expr-list}, \text{descr}) = \]
\[ \text{is-}<>(\text{expr-list}) \lor (\text{is-} * \lor \text{is-}<>) (\text{descr}) \rightarrow \text{PASS:<>} \]
\[ \text{is-}<>(\text{expr-list}) \lor \text{is-}<>(\text{descr}) \rightarrow \text{error} \]
\[ T \rightarrow \]
\[ \text{mk-list} (\text{arg}, \text{argl}) ; \]
\[ \text{argl} := \text{eval-arg-list}(\text{tail}(\text{expr-list}), \text{descr}_1) ; \]
\[ \text{arg} := \text{eval-arg}(\text{head}(\text{expr-list}), \text{descr}) \]

where:
\[ \text{descr}_1 = (\text{is-} *)(\text{descr}) \rightarrow *, \]
\[ T \rightarrow \text{head}(\text{descr}) ]
\[ \text{descr}_4 = (\text{is-} *)(\text{descr}) \rightarrow *, \]
\[ T \rightarrow \text{tail}(\text{descr}) ]

(88) \[ \text{eval-arg}(\text{expr}, \text{descr}) = \]
\[ \text{is-ctl-ref}(\text{expr}, \text{AT}) \lor (\text{is-} *)(\text{descr}) \lor \text{is-CTL-s-stg-cl}(\text{descr}) \lor \]
\[ \text{is-corrEsp}(\text{aggr}_1, \text{s-aggr}(\text{descr})) \rightarrow \]
\[ \text{eval-arg-1}(s-n(\text{expr}(\text{DN}), \text{CTL}) \]
\[ \text{is-var-ref}(\text{expr}, \text{AT}) \lor \text{is-CTL-s-stg-cl}(\text{descr}) \lor \]
\[ \text{gen-aggr-match-1}(\text{aggr}_1, \text{s-aggr}(\text{descr}), s-stg-cl(\text{attr}) \lor \]
\[ \text{connected-match}(\text{expr}, \text{descr}, \text{AT}) \rightarrow \]
\[ \text{eval-gen-arg}(\text{expr}, \text{s-aggr}(\text{descr})) \]
\[ \rightarrow \text{eval-dummy-arg}(\text{expr}, \text{s-aggr}(\text{descr})) \]
\[ T \rightarrow \text{error} \]

where:
\[ \text{aggr}_1 = \text{aggr-ref}(\text{expr}, \text{AT}) \]
\[ \text{attr}_0 = s-n(\text{expr}(\text{AT}) \]

Ref.:
\[ \text{is-var-ref 3-9(21)} \]
\[ \text{aggr-ref 8-25(71)} \]

(89) \[ \text{eval-arg-1}(b, \text{type}) = \]
\[ \text{PASS} : \mu_{0}(<s-den:b>, <s-type:type>) \]

6.2.3.7 Controlled arguments

Passing of controlled generation list (type CTL) applies if the argument is a reference referencing the complete declaration of a controlled variable (no subscripts or name qualifiers), the parameter descriptor specifies controlled storage class, and the aggregate structuring of the argument and of the parameter descriptor are the same with the exception of extents. In this case the aggregate name of the controlled variable itself is passed to the parameter, so that during the called procedure argument and parameter completely share all information contained in the list of generations in the aggregate directory AG and the referenced storage parts.
ABSTRACT SYNTAX AND INTERPRETATION OF PL/I

6.2.3.2 Generation arguments

Passing of a generation (type GEN) applies, if the argument is a reference referencing a variable generation (which may be a subgeneration of a full generation allocated in storage), the parameter descriptor does not specify controlled storage class, the aggregate attribute of the argument and that of the parameter descriptor is match and, provided the parameter descriptor specified as connected, the argument reference is statically recognizable as connected (cf. 8.3.3). The matching of aggregate attributes requires in this case, that the aggregate structuring is the same with the exception of those extents which are specified by * in the descriptor. For those extents which are specified by integers in the descriptor it is required that equality with those of the argument are recognizable in the static program text. That means that they have to be specified by integer constants only and that the argument cannot be of controlled storage class (for controlled variables it is not guaranteed that the extents of the current generation are the same as those in the declaration).

In this case the referenced generation is evaluated, entered into the aggregate directory AG under a newly created aggregate name b, and this aggregate name is passed to the parameter. Thereby during the called procedure argument and
parameter share the information contained in the passed generation and in the storage part referenced by this generation.

(54) \[\text{gen-aggr-match-1}(aggr-1, aggr-2, \text{stg-cl}) = \]
\[\begin{align*}
\text{is-0}(aggr-2) & \rightarrow T \\
\text{is-array}(aggr-2) & \rightarrow \\
\text{is-array}(aggr-1) & \& \text{gen-aggr-match-1}(\text{s-elem}(aggr-1), \text{s-elem}(aggr-2), \text{stg-cl}) \\
\text{is-struct}(aggr-2) & \rightarrow \\
\text{is-struct}(aggr-1) & \& \text{length}(aggr-1) = \text{length}(aggr-2) \& \\
& \text{gen-aggr-match-1}(\text{s-aggr-elem}(i, aggr-1), \text{s-aggr-elem}(i, aggr-2), \text{stg-cl}) \\
& \text{i}=1 \\
\text{is-scalar}(aggr-2) & \rightarrow \\
\text{is-scalar}(aggr-1) & \& \text{s-dens}(aggr-1) = \text{s-dens}(aggr-2) \& \\
& \text{gen-da-match-1}(\text{s-da}(aggr-1), \text{s-da}(aggr-2), \text{stg-cl}) \\
\end{align*}\]

for: \(\text{is-aggr} \& \text{is-0}(aggr-2)\)

Note: If there is no descriptor \((aggr-2=0)\), matching is assumed in any case. Array bounds are not tested here; their mismatch leads to an error in any case (see the function \text{gen-aggr-match} below). Mismatching string lengths and area sizes, however can be corrected by conversion in the case of dummy passing (cf. 6.2.3.3).

(55) \[\text{gen-da-match-1}(da-1, da-2, \text{stg-cl}) = \]
\[\begin{align*}
\text{is-string}(da-2) & \& \text{is-intg-vals-length}(da-2) \rightarrow \\
\text{is-string}(da-1) & \& \text{is-intg-const-s-length}(da-1) \& \text{-is-CTL}(\text{stg-cl}) \& \\
& \text{eval-da-1}(da-1) = da-2 \\
\text{is-area}(da-2) & \& \text{is-intg-vals-size}(da-2) \rightarrow \\
\text{is-area}(da-1) & \& \text{is-intg-const-s-size}(da-1) \& \text{-is-CTL}(\text{stg-cl}) \& \\
& \text{eval-da-1}(da-1) = da-2 \\
T & \rightarrow \text{pure-da}(da-1) = \text{pure-da}(da-2) \\
\end{align*}\]

Ref.: \(\text{is-intg-vals } 8-3(5)\)  
\(\text{is-intg-const } 6-10(30)\)  
\(\text{eval-da-1 } 6-10(26)\)

(56) \[\text{connected-match}(\text{ref}, \text{descr}, \text{at}) = \]
\[\begin{align*}
\text{is-0-s-connected}(\text{descr}) \& \text{is-connected}(\text{ref}, \text{at}) \\
\end{align*}\]

Ref.: \(\text{is-connected } 8-34(109)\)

6. FLOW OF CONTROL WITHIN A SINGLE TASK 21
(57) \texttt{eval-gen-arg(ref,aggr) =}
\begin{verbatim}
  eval-gen-arg-1(b,aggr);
  install-gen(b,gen);
  gen:eval-ref-gen(ref),
  b:un-name
\end{verbatim}

Ref.: \texttt{eval-ref-gen 8-28(82)}
\texttt{un-name 3-10(20)}

(58) \texttt{install-gen(b,gen) =}
\begin{verbatim}
  s-aggr(\texttt{AG};\langle b:\langle gen\rangle\rangle)
\end{verbatim}

Note: Since at task end (cf. 5.2) all but the last elements of the generation lists in the aggregate directory are freed, the passed generation is not freed by the aggregate name \texttt{b} of the argument.

(59) \texttt{eval-gen-arg-1(b,aggr) =}
\begin{verbatim}
  gen-aggr-match(s-eva(gen_1),aggr) \rightarrow eval-arg-1(b,\texttt{EII})
\end{verbatim}

T \rightarrow \texttt{error}

where:

\texttt{gen_1 = head\texttt{B(AG)}}

(60) \texttt{gen-aggr-match(eva,aggr) =}
\begin{verbatim}
  is-\texttt{G}(aggr) \rightarrow T
  is-array(aggr) \& is-\texttt{G}s-lbd(aggr) \& is-\texttt{G}s-nbd(aggr) \rightarrow
  is-array(eva) \& gen-aggr-match(s-elem(eva),s-elem(aggr))
  is-array(aggr) \rightarrow
  is-array(eva) \& s-lbd(eva) = s-lbd(aggr) \& s-nbd(eva) = s-nbd(aggr) \&
  gen-aggr-match(s-elem(eva),s-elem(aggr))
  is-struct(aggr) \rightarrow
  is-struct(eva) \& length(eva) = length(aggr) \&
  length(aggr)
  \forall i \in 1\rightarrow gen-aggr-match(s-aggr\cdot\texttt{elem}(i,eva),s-aggr\cdot\texttt{elem}(i,aggr))
  is-scalar(aggr) \rightarrow
  is-scalar(eva) \& s-dens(eva) = s-dens(aggr) \&
  gen-da-match(s-da(eva),s-da(aggr))
\end{verbatim}

for: (is-aggr \& is-\texttt{G})(aggr)
(61) \text{gen-da-match}(\text{da-1}, \text{da-2}) = \\
\text{is-string}(\text{da-2}) \& \text{is-intg-val}s\text{-length}(\text{da-2}) \lor \text{is-area}(\text{da-2}) \& \\
\text{is-intg-val}s\text{-size}(\text{da-2}) \rightarrow \\
\text{da-1} = \text{da-2} \\
\text{T} \rightarrow \text{pure-da}(\text{da-1}) = \text{pure-da}(\text{da-2})

Ref.: \text{is-intg-val} 9-3(5)

6.2.3.3 Dummy arguments

Passing of dummy variable \text{(type DUMMY)} applies if the parameter descriptor does not specify controlled storage class, at least one of the conditions given above for passing of generation is not satisfied, and none of the following actions lead to an error.

In this case, after pre-evaluation (cf. 8.2.1), in the first step the argument expression is expanded, evaluated and converted according to the parameter descriptor, if present, or the aggregate attribute of the argument expression itself. Since the data attributes of the dummy variable to be allocated for the argument are not fully known in some cases before the expansion and evaluation (string lengths and area sizes need not be known in advance), it is not possible to assign the single components immediately to the dummy. Therefore they are set aside intermediate in an auxiliary object, called "dummy operand" \text{(dop)}, which additionally reflects the structuring of the complete aggregate. This dummy operand has the following structure:

\text{is-dop} = \\
\text{is-or} \lor \text{is-dop-list} \lor \langle \langle \text{s-lb}: \text{is-intg-val}, \langle \text{s-ubd}: \text{is-intg-val}, \rangle, \\
\langle \text{s-elem}: \text{is-dop-list} \rangle \rangle.

Thereby the first alternative is an operand in the scalar case, the second the list of dummy operands of the components in the structure case, and the third the bounds and the list of dummy operands of the components in the array case.

In the second step, from this dummy operand the aggregate attribute of the dummy variable is extracted. Apparently it contains the bounds of arrays and the numbers of components of structures. The determination of the scalar data attributes is somewhat difficult: It has to be guaranteed that all components of an array have the same scalar data attribute. For this purpose in the case of strings and areas the maximum length or size of all components is determined (all other types of data attributes are fixed by the given parameter descriptor or aggregate attribute of the expression).

As third step, the dummy variable with the so determined aggregate attribute is allocated and its generation entered into the aggregate directory \text{AG} under a newly created aggregate name \text{b} (which finally will be passed as aggregate name of the argument). Then the dummy operand is assigned component by component to this dummy variable; thereby strings and areas are possibly converted to longer ones.
(62) \[ \text{eval-dummy-arg} \text{ (expr,aggr) } = \]
\[ \text{eval-dummy-arg-1} \text{ (b,dummy) } ; \]
\[ \text{dummy-assign} \text{ (gen, dop) } ; \]
\[ \text{gen:allocate} \text{ (b,eva,dummy) } ; \]
\[ \text{b:is-name} \text{ ,} \]
\[ \text{eval-dummy-eva} \text{ (dcp) } ; \]
\[ \text{dop:eval-dummy-expr} \text{ (expr-1,aggr-1) } ; \]
\[ \text{expr-1:pre-eval} \text{ (expr) } \]

where:
\[ \text{aggr-1 } = \text{ (is-0 (aggr) } \rightarrow \text{ aggr,} \]
\[ \text{T } \rightarrow \text{ pure-aggr=aggr-expr-1 (expr,AT) } \]

for: \( \text{ (is-aggr } \vee \text{ is-0) (aggr) } \)

Ref.: \[ \text{ allocate } 7-11(25) \]
\[ \text{ una-name } 3-10(20) \]
\[ \text{ pre-eval } 8-11(26) \]

(63) \[ \text{eval-dummy-expr} \text{ (expr,aggr) } = \]
\[ \text{is-scalar} \text{ (aggr) } \rightarrow \]
\[ \text{convert} \text{ (eval-aggr-1 (aggr),op,area-1,area-2) ;} \]
\[ \text{op:eval-expr-1 (expr,s-da (aggr)) } \]
\[ \text{is-array (aggr) } \& \text{ is-intg-val} \text{ s-lbd (aggr) } \& \text{ is-intg-val} \text{ s-ubd (aggr) } \rightarrow \]
\[ \text{pass} \text{ (dop) ;} \]
\[ \text{s-elem} \text{ (dop):iterate-dummy-expr} \text{ (expr,aggr,s-lbd (aggr)) ;} \]
\[ \text{dop:pass} \text{ (aggr) } \]
\[ \text{is-array (aggr) } \& \text{ is-array (aggr-1) } \rightarrow \]
\[ \text{eval-dummy-expr} \text{ (expr,aggr-1) ;} \]
\[ \text{s-elem} \text{ (aggr-1):pass} \text{ (s-elem (aggr)) ;} \]
\[ \text{aggr-1:array-eva-expr} \text{ (expr) } \]
\[ \text{is-struct} \text{ (aggr) } \rightarrow \text{ iterate-dummy-expr} \text{ (expr,aggr-1) } \]
\[ \text{T } \rightarrow \text{ error } \]

where:
\[ \text{aggr-1 } = \text{ aggr-expr (expr,AT) } \]
\[ \text{area-1 } = \text{ s-area } \& \text{ s-da (aggr)} \]
\[ \text{area-2 } = \text{ (is-RTB} \text{ s-da (aggr) } \rightarrow \text{ area-expr (expr,AT),} \]
\[ \text{T } \rightarrow \text{ @ } \]

Ref.: \[ \text{ convert } 8-8(17) \]
\[ \text{eval-aggr-1 } 6-9(25) \]
\[ \text{eval-expr-1 } 8-10(25) \]
\[ \text{is-intg-val} 9-3(5) \]
\[ \text{array-eva-expr} 8-26(76) \]
\[ \text{aggr-expr } 8-23(62) \]
\[ \text{area-expr } 8-28(81) \]

Note: If the parameter descriptor is an array without specified bounds, these are determined from the pre-evaluated expression itself (cf. 8.2.5.2).

24 6. FLOW OF CONTROL WITHIN A SINGLE TASK
(64) \( \text{iterate-dummy-expr}(\text{expr}, \text{aggr}, i) = \)
\[ i > \text{ubd}(\text{aggr}) \rightarrow \text{PASS:<>} \]
\[ \mu \left( \text{head}(\text{dopl}); \langle \text{s-length:}\text{max-list} \left( \left\{ \text{LIST} \begin{array}{c} \text{s-length}\cdot\text{elem}(j,\text{dopl}) \end{array} \right) \right) \rangle \right) \]
\[ \text{is-list} \cdot \text{head}(\text{dopl}) \rightarrow \]
\[ \text{is-op} \cdot \text{head}(\text{dopl}) \rightarrow \]
\[ \mu_0 \left( \langle \text{s}-\text{aggr}\cdot\text{dummy-da}(\ldots \text{LIST} \begin{array}{c} \text{s-da}\cdot\text{ev} \cdot \text{elem}(j,\text{dopl}) \end{array} \rangle, \right) \]
\[ \mu_0 \left( \langle \text{s}-\text{dens}\cdot\text{s-da}\cdot\text{head}(\text{dopl}) \rangle \right) \]

Ref.: \text{uhd 7-13}(36) \text{aggr-part 8-37}(119) \text{mod 8-5}(8)

(65) \( \text{dummy-eva}(\text{dop}) = \)
\[ \text{PASS:dummy-eva-1}(<\text{dop}>) \]

(66) \( \text{dummy-eva-1}(\text{dopl}) = \)
\[ \mu_0 \left( \langle \text{s}-\text{aggr}\cdot\text{dummy-eva-1} \{ \text{CONC} \text{s-elem}\cdot\text{elem}(j,\text{dopl}) \} \rangle \right) \]
\[ \text{is-intg}\cdot\text{val}\cdot\text{s-list}\cdot\text{head}(\text{dopl}) \rightarrow \]
\[ \text{is-list}\cdot\text{head}(\text{dopl}) \rightarrow \]
\[ \text{is-op}\cdot\text{head}(\text{dopl}) \rightarrow \]
\[ \mu_0 \left( \langle \text{s}-\text{da}\cdot\text{dummy-da}(\ldots \text{LIST} \begin{array}{c} \text{s-da}\cdot\text{ev} \cdot \text{elem}(j,\text{dopl}) \end{array} \rangle, \right) \]
\[ \mu_0 \left( \langle \text{s}-\text{dens}\cdot\text{s-da}\cdot\text{head}(\text{dopl}) \rangle \right) \]

Ref.: \text{is-intg}\cdot\text{val} 9-3(5) \text{is-op} 9-9(34)

Note: The dummy operand list dopl consists of dummy operands belonging to the different components of the same array. Therefore the aggregate structure of each of them is the same and is taken from the first dummy operand of the list. This applies not for string lengths and area sizes, where the function dummy-da determines the longest one.

(67) \( \text{dummy-da}(\text{dal}) = \)
\[ \mu_0 \left( \langle \text{s}-\text{size:}\text{max-list} \left( \left\{ \text{LIST} \begin{array}{c} \text{s-size}\cdot\text{elem}(j,\text{dal}) \end{array} \right) \right) \rangle \right) \]
\[ \mu_0 \left( \langle \text{s}-\text{size:}\text{max-list} \left( \left\{ \text{LIST} \begin{array}{c} \text{s-size}\cdot\text{elem}(j,\text{dal}) \end{array} \right) \right) \rangle \right) \]
\[ \text{T} \rightarrow \text{head}(\text{dal}) \]

6. FLOW OF CONTROL WITHIN A SINGLE TASK 25
(68) \[
\text{max-list(list)} =
\begin{align*}
\text{is-<>etail(list)} & \rightarrow \text{head(list)} \\
T & \rightarrow \max(\text{head(list)}, \text{max-list\_tail(list)})
\end{align*}
\]

for: is-intg-val-list(list)

Ref.: is-intg-val 9-5(5)

(69) \[
\text{dummy-assign(gen,dop)} =
\begin{align*}
\text{is-arrays\_eva(gen)} \rightarrow \\
\text{null;}
\begin{align*}
&\text{dummy-assign(sub-gen(gen,<i>),elem(i - s-lbd(dop) + 1, s-elem(dop)))} \mid s-lbd(dop) \leq i \leq s-ubd(dop)} \\
\text{is-structs\_eva(gen)} \rightarrow \\
\text{null;}
\begin{align*}
&\text{dummy-assign(sub-gen(gen,<i>),elem(i,dop))} \mid 1 \leq i \leq \text{length(dop)}
\end{align*}
\end{align*}
\]

\text{is-scalars\_eva(gen)} \rightarrow \text{convert-assign(gen,dop,0,0)}

Ref.: sub-gen 8-36(15)\text{convert-assign} 8-8(16)

Note: Proper conversion occurs only for strings and areas to greater lengths or sizes. So no condition can be raised by these conversion.

6.2.3.4. Installation of arguments in the called procedure

In the called procedure the types of the passed arguments and the evaluated aggregate attributes of their passed generations are tested against the declarations of the corresponding parameters. If they match, the passed aggregate names are entered as denotations of the parameters into the denotation directory.

In the special case of an argument of type CTL, i.e. that a complete generation list has been passed, and a parameter declared as non-controlled, passing of type GEN is simulated entering the current generation of the argument under a newly created aggregate name into the aggregate directory AG.
(70) \[\text{install-arg-list}(arg_l, \text{param-list}, dp) =\]
\[
\text{length}(arg_l) = \text{length}(\text{param-list}) \land E_i \text{is-}\text{PARAM}\text{*s}\text{-scope}(\text{attr}_i) \rightarrow
\]
\[\forall i,\]
\[\begin{cases}
\text{install-arg}(s\text{-den}(arg_i), s\text{-type}(arg_i), n_i, attr_i) & \text{if } 1 \leq i \leq \text{length}(arg_l)\\
T \rightarrow \text{error}
\end{cases}
\]
\[\text{where:}\]
\[arg_i = \text{elem}(i, arg_l)\]
\[attr_i = s\text{-id}\text{*elem}(i, \text{param-list})(dp)\]
\[n_i = s\text{-count}\text{*elem}(i, \text{param-list})\]

(71) \[\text{install-arg}(b, \text{type}, n, \text{attr}) =\]
\[
\text{is-CTL}(\text{type}) \land \text{is-CTL}\text{*s}\text{-stg-cl}(\text{attr}) \land \text{is-corresp}(s\text{-eva}(\text{gen}_1), s\text{-aggr}(\text{attr})) \rightarrow
\]
\[
\text{upd-decl-dn-1}(n, b)
\]
\[
(is\text{-GEN} \lor \text{is-DUMMY})(\text{type}) \land \text{is-}\text{\#s}\text{-stg-cl}(\text{attr}) \land
\]
\[
\text{gen-aggr-match}(s\text{-eva}(\text{gen}_1), \text{eval-aggr-\#s-aggr}(\text{attr})) \land
\]
\[
\text{connected-match-1}(\text{gen}_1, \text{attr}) \land
\]
\[
(V)(\text{is-\#s\text{-init}\$}(\text{attr}) \land \text{is-}\text{\#s\text{-init}}\$)(\text{attr})) \rightarrow
\]
\[
\text{upd-decl-dn-1}(n, b)
\]
\[
\text{is-CTL}(\text{type}) \land \text{is-}\text{\#s}\text{-stg-cl}(\text{attr}) \land \text{is-gen}(\text{gen}_1) \rightarrow
\]
\[
\text{install-arg}(b\text{-1}, \text{GEN}, n, \text{attr});
\]
\[
\text{install-gen}(b\text{-1}, \text{gen}_1);
\]
\[b\text{-true-name}\]
\[T \rightarrow \text{error}
\]
\[\text{where:}\]
\[\text{gen}_1 = \text{head}\text{*h}(\text{AG})
\]

Ref.:
\[\text{upd-decl-dn-1} 6-5(14)
\]
\[\text{eval-aggr-1} 6-9(25)
\]
\[\text{is-gen} 3-14(30)
\]
\[\text{pp-name} 3-10(26)
\]

Note: In the case of type CTL, all generations of the passed generation list have the same aggregate structure, except extent values; the test is against the head generation or, if no generation is allocated yet, against the evaluated aggregate attribute entered into AG by the prepare (cf. 4.3).

(72) \[\text{connected-match-1}(\text{gen}, \text{attr}) =\]
\[
\text{is-\#s\text{-connected}(\text{attr})} \lor \text{is-ptr-val\#s-pp}(\text{gen}) \land \text{s-eva}(\text{gen}) = \text{s-ri}(\text{gen})
\]

Ref.:
\[\text{is-ptr-val} 3-15(36)
\]

Note: If the parameter is declared as connected, the passed generation is tested to be really connected, while the test against the parameter descriptor before the call (cf. 6.2.3.2) by the function connected-match required
additionally that the argument is recognizable as connected in the static program text already.

6.2.4 THE RETURN STATEMENT

This section defines the return from a procedure either by means of a return statement or by exhaustion of its statement list (cf. 6.2.2.). A return statement terminates the innermost active procedure and all nested block activations. Termination of an on unit by a return statement is forbidden.

In the case of a procedure activated by a function call, the calling block activation had passed an aggregate name b as s-funct-den component into the epilogue information xi of the called procedure (cf. 6.2.1). The return statement allocates for the return value a dummy variable according to the aggregate attribute of the result of the return expression and enters its generation under this aggregate name b into the aggregate directory AG, where it is then available to the calling block activation.

Metavariabes:

\[ t \] is-return-st \& is-\& \quad \text{a return statement or, in case of exhaustion of the statement list, } \&.

\[ b \] is-n \quad \text{aggregate name for dummy for returned function value}

\[ op \] is-op \quad \text{returned operand}

(73) \[ \text{is-return-st}(t) = \]

\[ \text{is-\&s-expr}(t) \& \text{is-\&s-funct-den}(xi) \rightarrow \]

\[ \text{return;} \]

\[ \text{test-finish} \]

\[ \neg \text{is-\&s-expr}(t) \& \neg \text{is-\&s-funct-den}(xi) \rightarrow \]

\[ \text{return;} \]

\[ \text{test-finish;} \]

\[ \text{allocate-assign}(s\text{-funct-den}(xi), op); \]

\[ \text{op: convert}(\text{eval-aggr-1s-ret-type}(xi), op-1, \text{area}_1, \text{area}_2); \]

\[ \text{op-1: eval-expr}(s\text{-expr}(t), s\text{-daes-ret-type}(xi)) \]

\[ T \rightarrow \text{error} \]

where:

\[ \text{area}_1 = s\text{-areas-daes-ret-type}(xi) \]

\[ \text{area}_2 = (is-P\text{-Pareas-daes-ret-type}(xi)) \rightarrow \text{area-exprs-expr}(t), \]

\[ T \rightarrow \text{error} \]

Ref.:

convert 8-8(17)

eval-aggr-1 6-9(25)

eval-expr 8-10(24)

area-expr 8-28(81)

28 6. FLOW OF CONTROL WITHIN A SINGLE TASK
(74) \texttt{allocate-assign} (b, op) =
\begin{verbatim}
is-\oplus{b} (AG) →
\texttt{assign}(gen, op);
\hspace{1em} \texttt{gen:allocate}(b, s-\texttt{eva}(op), \texttt{DUMMY})
\texttt{T} →
\texttt{assign}(gen, op);
\hspace{1em} \texttt{gen:allocate}(b, s-\texttt{eva}(op), \texttt{DUMMY});
\hspace{1em} \texttt{free}(b)
\end{verbatim}
Ref.: \texttt{assign} 8-9(23)
\texttt{allocate} 7-11(25)
\texttt{free} 7-18(58)

Note: The second alternative applies only in the case that by a return statement
the finish condition is raised, the on-unit is terminated abnormally by a
goto statement, and a second return statement is executed.

(75) \texttt{test-finish} =
\begin{verbatim}
is-\oplus{s}\texttt{-main}(FJ) → \texttt{null}
\texttt{T} → \texttt{call-cond}(\texttt{FINFIB})
\end{verbatim}
Ref.: \texttt{call-cond} 13-18(54)

Note: If the procedure to be terminated is the main procedure of the program
(cf. 4.2 and 6.2.2) the finish condition is raised before termination of
blocks.

(76) \texttt{return} =
\begin{verbatim}
is-\oplus{BLOCK}\texttt{s-block-act}(FJ) →
\hspace{1em} \texttt{s-\oplus{\mu}(D:s-c:\texttt{return})}
\hspace{1em} s-c:\texttt{epilogue}
\hspace{1em} \texttt{is-FROC\texttt{s-block-act}(FJ)} → \texttt{epilogue}
\hspace{1em} \texttt{is-CN\texttt{s-block-act}(FJ)} → \texttt{error}
\end{verbatim}
Ref.: \texttt{epilogue} 6-11(31)

6.2.5.\textsc{Generic Selection}

If in an entry context a generic identifier occurs instead of an entry
reference the generic identifier is to be replaced by the entry reference found
from the corresponding generic declaration according to the parameter attributes
required in the entry context. The selection of this entry reference is performed
by the function \texttt{generic-sel} (gml, attr-list), where gml is the generic declaration,
i.e. the list of generic members consisting each of an entry reference to be
selected and a generic descriptor list governing the selection, and attr-list is
the list of parameter attributes of the context.

The selection is performed in three steps: First, all generic members are
eliminated from the list gml, whose descriptor list has the wrong length. Second,
in the attribute list attr-list for all scalar components of all attributes which are of type entry it is tested whether there is at least one generic member in gml specifying also entry for the corresponding component (or the corresponding descriptor is missing). If this is true, the attribute component of type entry is left unchanged, otherwise it is replaced by its final return type. Third the elements of gml are inspected one after another whether their generic descriptor lists match the so modified attribute list. The entry reference of the first matching generic member is selected. The second step is defined in section 6.2.5.1, the third one in section 6.2.5.2.

The parameter attributes required in an entry context which govern the generic selection are found in two different ways: If the selected entry reference shall be called immediately, they are the attributes of the argument expressions of the call (cf. 6.2.1, 6.2.2, 6.2.3). These are found by the function attr-expr-list defined in section 6.2.5.3. If the selected entry reference shall be assigned to an entry variable (e.g. on parameter passing) they are taken from the parameter descriptor list declared for that variable (cf. 6.2.3.3, 8.2.2). In some contexts where entry references may occur, a generic identifier is forbidden (right hand side of an assignment statement, entry reference of a generic member). In these cases the attribute list is set to $0$ and the generic selection leads to an error (cf. 8.1.3, 8.1.2, 8.2.3).

**Metavariables**

- **gml** is-generic-member-list
  - List of alternative members of a generic family
- **descr1** is-generic-descr-list
  - List of generic parameter descriptors
- **descr** is-generic-descr
  - Generic parameter descriptor
- **aggr1** is-generic-aggr-list
  - List of aggregate attributes of generic parameter descriptors
- **g-aggr** is-generic-aggr
  - Aggregate attribute of a generic parameter descriptor
- **g-da** is-generic-da $\&$ is-$\ast$
  - Data attribute in a generic parameter descriptor
- **g-dens** is-AL $\&$ is-UNAL $\&$ is-$\ast$
  - Density attribute in a generic parameter descriptor
- **dal** is-generic-da-list
  - List of data attributes in generic parameter descriptors
- **expr-list** is-expr-list
  - List of argument expressions
- **expr** is-expr
  - Argument expression
- **attr-list**
  - List of argument attributes governing generic selection, in some erroneous cases $0$ or $\ast$.
- **attr** is-descr $\&$
  - Attribute of argument consisting essentially of storage class and aggregate attribute
- **aggr** is-aggr
  - Aggregate attribute of argument
- **da** is-da
  - Data attribute of argument
- **at** is-at
  - The attribute directory AT
- **i,j** is-intg-val
  - Integer values
ABSTRACT SYNTAX AND INTERPRETATION OF PL/I

(77) \[\text{generic-sel}(gml, \text{attr-list}) = \]
\[\text{if}-(\text{is}-\Omega \lor \text{is}-\varnothing) (\text{attr-list}) \rightarrow \text{error}\]
\[T \rightarrow \text{generic-sel-1}(gml_1, \text{attr-list}_1)\]

where:
\[gml_1 = \text{reduce-gml}(gml, \text{length}(\text{attr-list}))\]
\[\text{attr-list}_1 = \text{modify-entry-list}(\text{attr-list}, gml_1)\]

(78) \[\text{reduce-gml}(gml, lg) =\]
\[\text{is}-\varnothing(gml) \rightarrow \varnothing\]
\[\text{length}(\text{s-descr-list} \cdot \text{head}(gml)) = lg \rightarrow \text{head}(gml) \cdot \text{reduce-gml}(\text{tail}(gml), lg)\]
\[T \rightarrow \text{reduce-gml}(\text{tail}(gml), lg)\]

for: \text{is-intg-val}(lg)

Ref.: \text{is-intg-val} 9-3(5)

6.2.5.1 Prescan of entry attributes

The question whether for generic selection in the case of an entry argument the attribute of the entry itself or the attribute of the function value resulting from its invocation governs generic selection is solved in this section. If the corresponding generic parameter descriptor is missing (or only a storage class or density is specified), in any case entry is assumed. Otherwise, this question is answered for each scalar component of an aggregate attribute separately. If for at least one of the generic members the corresponding component of the corresponding descriptor specifies either ENTRY or no data attribute, entry is assumed, otherwise the final return type.

Thereby only those generic members are considered whose corresponding descriptor has the same aggregate structure as the argument attribute.

(79) \[\text{modify-entry-list}(\text{attr-list}, gml) =\]
\[\text{length}(\text{attr-list})\]
\[\text{LIST} \cdot \text{modify-entry}(\text{elem}(i, \text{attr-list}), \text{aggrl}_i)\]

where:
\[\text{aggrl}_i = \text{LIST} \cdot \text{s-aggr}\cdot\text{elem}(i, \text{s-descr-list} \cdot \text{elem}(j, gml))\]

(80) \[\text{modify-entry}(\text{attr}, \text{aggrl}) =\]
\[\mu_0(\text{<s-stg-cl}:\text{stg-cl}_i, \text{<s-aggr}:\text{aggr-mod}_i>)\]

where:
\[\text{aggr-mod}_i = \text{modify-entry-1}(\text{s-aggr}(\text{attr}), \text{aggrl})\]
\[\text{stg-cl}_i = \text{(aggr-mod}_i = \text{s-aggr}(\text{attr}) \rightarrow \text{s-stg-cl}(\text{attr}), T \rightarrow \varnothing)\]
ABSTRACT SYNTAX AND INTERPRETATION OF PL/I

modify-entry-1 (aggr, aggrl) =

(∃j) (1 ≤ j ≤ length (aggrl)) & is-≠s-da·elem(j, aggrl) -- aggr
T ← modify-entry-2 (aggr, reduce-aggrl (aggr, aggrl))

reduce-aggrl (aggr, aggrl) =

is-<> (aggrl) ← <>
is-eq·struct (aggr, head (aggrl)) ← head (aggrl) > reduce-aggrl (aggr, tail (aggrl))
T ← reduce-aggrl (aggr, tail (aggrl))

is-eq·struct (aggr, g-aggr) =

is-array (g-aggr) ←

is-array (aggr) & dim (aggr) = dim (g-aggr) &
is-eq·struct (base-elem (aggr), base-elem (g-aggr))
is-array (aggr) ← is-eq·struct (base-elem (aggr), g-aggr)
is-struct (g-aggr) ←
is-struct (aggr) & length (aggr) = length (g-aggr) &
\phantom{i=1}length (g-aggr)
\phantom{i=1}is-eq·struct (s-aggr·elem(i, aggr) , s-aggr·elem(i, g-aggr))
is-scalar (g-aggr) ← is-scalar (aggr)

Ref.: dim 8-29 (86)
base-elem 8-29 (85)

modify-entry-2 (aggr, aggrl) =

is-array (aggr) ← μ (aggr; s-elem; modify-entry-2 (s-elem (aggr) , base-aggrl₁))
is-struct (aggr) ←

\phantom{i=1}length (aggr)
\phantom{i=1}LIST μ₀ (s-aggr·elem(i, aggr) , aggrl-i₁)
is-scalar (aggr) ← modify-entry-3 (aggr, dal₁)

where:

base-aggrl₁ = LIST base-elem·elem (j, aggrl)
\phantom{j=1}length (aggrl)
aggrl-i₁ = LIST s-aggr·elem (i, elem (j, aggrl))
\phantom{j=1}length (aggrl)
dal₁ = LIST s-da·elem (j, aggrl)
\phantom{j=1}

Ref.: base-elem 8-29 (85)

32 6. FLOW OF CONTROL WITHIN A SINGLE TASK
(85) modify-entry-3(aggr,dal) =
    -is-entry•s-da(aggr) → aggr
    {3j} (1 ≤ j ≤ length(dal) & (is-* v is-ENTRY)*elem(j,dal)) → aggr
    is-*final-ret-type(aggr) → error
    T → final-ret-type(aggr)

Ref.: final-ret-type 8-28(80)

Note: The error-case applies for generic arguments; their data attribute is
determined as ENTRY (cf. 6.2.5.3), but before their generic selection they
have no specified return type.

6.2.5.2 The selection

The first element of the generic member list whose descriptor list matches the
given attribute list (after modification as defined in 6.2.5.1) is reflected.
Thereby matching means identity of all those parts which are explicitly specified
in the generic descriptors.

(86) generic-sel-1(gml,attr-list) =
    is<>(gml) → error
    generic-match(attr-list,s-descr-list•head(gml)) → s-entry•head(gml)
    T → generic-sel-1(tail(gml),attr-list)

(87) generic-match(attr-list,descr) =
    length(descr)
    Et generic-attr-match(elem(i,attr-list),elem(i,descr))
    i=1

(88) generic-attr-match(attr,descr) =
    generic-aggr-match(s-aggr(attr),s-aggr(descr)) &
    (is-*v s-stg-cl(descr) v s-stg-cl(attr) = s-stg-cl(descr))

(89) generic-aggr-match(aggr,g-aggr) =
    is-*s-da(g-aggr) → generic-dens-match(aggr,s-dens(g-aggr))
    is-array(g-aggr) →
    is-array(aggr) & dim(aggr) = dim(g-aggr) &
    generic-aggr-match(base-elem(aggr),base-elem(g-aggr))
    T → generic-aggr-match-1(aggr,g-aggr)

Ref.: dim 8-29(86)
     base-elem 8-29(85)

Note: If only a dimension (and possibly storage class and density) is specified
for the descriptor any array argument of that dimension matches,
irrespective what structure the components of that array have. If however any other attribute is specified the aggregate structure has to match fully (this is tested by the following function).

(90) \[ \text{generic-aggr-match-1}(\text{aggr}, g-\text{aggr}) = \]
\[ \begin{array}{l}
\text{is-array}(g-\text{aggr}) \rightarrow \\
\text{is-array}(\text{aggr}) \land \text{dim}(\text{aggr}) = \text{dim}(g-\text{aggr}) \land \\
\quad \text{generic-aggr-match-1}(\text{base-elem}(\text{aggr}), \text{base-elem}(g-\text{aggr})) \\
\text{is-array}(\text{aggr}) \rightarrow \text{generic-aggr-match-1}(\text{base-elem}(\text{aggr}), g-\text{aggr}) \\
\text{is-struct}(g-\text{aggr}) \rightarrow \\
\text{is-struct}(\text{aggr}) \land \text{length}(\text{aggr}) = \text{length}(g-\text{aggr}) \land \\
\quad \text{length}(g-\text{aggr}) \\
\quad \text{generic-aggr-match-1}(\text{base-elem}(\text{aggr}), \text{base-elem}(\text{aggr})) \\
\text{is-struct}(\text{aggr}) \rightarrow \\
\text{is-struct}(g-\text{aggr}) \land \text{generic-da-match}(\text{da}(\text{aggr}), s-\text{da}(g-\text{aggr})) \land \\
\quad \text{generic-dens-match}(\text{aggr}, s-\text{den}(g-\text{aggr}))
\end{array} \]

Ref.: \( \text{dim 8-29(86)} \)
\( \text{base-elem 8-29(85)} \)

(91) \[ \text{generic-da-match}(\text{da}, g-\text{da}) = \]
\[ \begin{array}{l}
\text{is-\texttt{*}}(g-\text{da}) \rightarrow T \\
\text{is-arithm}(g-\text{da}) \rightarrow \text{generic-arithm-match}(\text{da}, g-\text{da}) \\
\text{is-string}(g-\text{da}) \rightarrow \text{generic-string-match}(\text{da}, g-\text{da}) \\
\text{is-pict}(g-\text{da}) \rightarrow \text{generic-pict-match}(\text{da}, g-\text{da}) \\
\text{is-\texttt{AREA}}(g-\text{da}) \rightarrow \text{is-area}(\text{da}) \\
T \rightarrow \text{eval-da-1}(\text{da}) = g-\text{da}
\end{array} \]

Ref.: \( \text{eval-da-1 6-10(26)} \)

(92) \[ \text{generic-arithm-match}(\text{da}, g-\text{da}) = \]
\[ \begin{array}{l}
(\text{is-\texttt{\texttt{*}}} g-\text{mode}(g-\text{da}) \lor s-\text{mode}(\text{da}) = s-\text{mode}(g-\text{da})) \land \\
(\text{is-\texttt{\texttt{*}}} g-\text{base}(g-\text{da}) \lor s-\text{base}(\text{da}) = s-\text{base}(g-\text{da})) \land \\
(\text{is-\texttt{\texttt{*}}} g-\text{scale}(g-\text{da}) \lor s-\text{scale}(\text{da}) = s-\text{scale}(g-\text{da})) \land \\
(\text{is-\texttt{\texttt{*}}} s-\text{prec}(g-\text{da}) \lor s-\text{prec}(\text{da}) = s-\text{prec}(g-\text{da})) \land \\
(\text{is-\texttt{\texttt{*}}} s-\text{scale-f}(g-\text{da}) \lor s-\text{scale-f}(\text{da}) = s-\text{scale-f}(g-\text{da}))
\end{array} \]

Note: Also the case that only the mode CPLX and no other data attribute component is specified for the generic descriptor and that the argument is a picture variable with CPLX mode leads from generic-da-match into this function and yields "true".

(93) \[ \text{generic-string-match}(\text{da}, g-\text{da}) = \]
\[ \begin{array}{l}
(\text{is-\texttt{\texttt{*}}} g-\text{base}(g-\text{da}) \lor s-\text{base}(\text{da}) = s-\text{base}(g-\text{da})) \land \\
(\text{is-\texttt{\texttt{*}}} s-\text{varying}(g-\text{da}) \lor s-\text{varying}(\text{da}) = s-\text{varying}(g-\text{da}))
\end{array} \]
(94) \[ \text{generic-pie-match}(da, g-da) = \]
\[ \text{(is-\text{is-mode}(g-da) \land s-mode(da) = s-mode(g-da)) \land \delta(da; s-mode) = \delta(g-da; s-mode)} \]

(95) \[ \text{generic-dens-match}(aggr, g-dens) = \]
\[ \text{is-\text{is}(g-dens) \land T} \]
\[ \text{is-array}(aggr) \rightarrow \text{generic-dens-match}(\text{base-elem}(aggr), g-dens) \]
\[ \text{length}(aggr) \]
\[ \text{is-\text{s-\text{struct}}}(aggr) \rightarrow \text{Et}_{i=1} \text{generic-dens-match}(s-aggr\text{\text{-elem}}(i, aggr), g-dens) \]
\[ \text{is-\text{s-\text{\text{scalar}}}}(aggr) \rightarrow s-dens(aggr) = g-dens \]

Ref.: base-elem 8-29(65)

Note: If the descriptor specified only a density and nothing else about aggregate structuring and data attributes, any aggregate argument matches if all its components have the specified density.

6.2.5.3 Attributes of arguments

The attributes of arguments needed for generic selection are essentially determined by use of the function aggr-expr (cf. 8.2.5.1). Only in the case of a generic identifier without arguments, where aggr-expr leads to an error since no generic selection can be done, ENTRY is assumed.

(96) \[ \text{attr-expr-list}(expr-list, at) = \]
\[ \text{length}(expr-list) \]
\[ \text{LIST}_{i=1} \text{attr-expr}(\text{elem}(i, expr-list), at) \]

(97) \[ \text{attr-expr}(expr, at) = \]
\[ \text{is-\text{\text{ctl-ref}}}(expr, at) \rightarrow \delta(\text{attr}_0; s\text{-\text{\text{scope}}, s\text{-\text{\text{connected}}}, s\text{-\text{\text{density}}}) \]
\[ T \rightarrow \mu_0(\langle s\text{-aggr}\text{-aggr-expr}-1(expr, at)\rangle) \]

where:
\[ \text{attr}_0 = s-n(expr)(at) \]
(98) \[
\text{aggr-expr-1(expr,at)} =
\]
\[
is\text{-paren-expr-1(expr) } \rightarrow \text{aggr-expr-1(s-op(expr),at)}
\]
\[
is\text{-ref(expr) } \land \text{is-generic(attr}_0\text{) } \land \text{is-<>-ap(expr) } \rightarrow \text{aggr-scalar(ENTRY)}
\]
\[
is\text{-ref(expr) } \land \text{is-BUILTIN(attr}_0\text{) } \land \text{is-float-generic-builtin(id}_0\text{) } \land
\]
\[
is\text{-<>-ap(expr) } \rightarrow
\]
\[
\text{aggr-scalar(ENTRY)}
\]
\[
T \rightarrow \text{aggr-expr(expr,at)}
\]

where:
\[
\text{attr}_0 = \text{s-n(expr)(at)}
\]
\[
\text{id}_0 = \text{head-s-id-list(expr)}
\]

Ref.:  is-paren-expr-1 8-16(43)
aggr-scalar 8-23(64)
is-float-generic-builtin 12-11(26)
aggr-expr 8-23(62)

6.2.6 THE FETCH AND RELEASE STATEMENTS

The fetch and release statements specify lists of references which are expanded into their scalar components and evaluated. The results have to be lists of entry values. The denotations of these entry values are made accessible or unaccessible by the fetch or release statement. This action serves for storage economy in a concrete implementation of PL/I. It is completely implementation defined and has no semantic effect, since each call statement includes an implicit fetch if the called entry denotation is released or not yet fetched.

Metavariables

\[
t \quad \text{is-fetch-st} \lor \text{is-release-st} \quad \text{a fetch or release statement}
\]
\[
\text{ref-list} \quad \text{is-ref-list} \quad \text{list of references for entries to be fetched or released}
\]
\[
type \quad \text{is-FETCH} \lor \text{is-RELEASE} \quad \text{statement type}
\]
\[
dop \quad \text{"dummy operand" resulting from evaluation of reference-list (cf. 6.2.3.3).}
\]
\[
\text{opl} \quad \text{is-op-list} \quad \text{list of entry operands}
\]
\[
\text{entry-list} \quad \text{is-n-list} \quad \text{list of unique names identifying entry denotations}
\]

(99) \[
\text{int-fetch-st}(t) =
\]
\[
\text{fetch-release}(\text{e-entry-list}(t),\text{FETCH})
\]

36 6. FLOW OF CONTROL WITHIN A SINGLE TASK
(100) \[ \text{int-release-st}(t) = \]
\[ \text{fetch-release}(s\text{-entry-list}(t), \text{RELEASE}) \]

(101) \[ \text{fetch-release}(\text{ref-list}, \text{type}) = \]
\[ \text{fetch-release-1}(\text{entry-list}, \text{type}); \]
\[ \text{entry-list}\text{:pass-entry-val-list}(\text{opl}); \]
\[ \text{opl}\text{:pass-list-dop}(\text{dop}); \]
\[ \{\text{elem}(i)(\text{dop}) : \text{eval-dummy-err}(\text{ref}, \text{aggr-ref}(\text{elem}(i, \text{ref-list}), \text{AF})); \]
\[ \text{ref}\text{:pre-eval}(\text{elem}(i, \text{ref-list})) \mid 1 \leq i \leq \text{length}(\text{ref-list}) \]

Ref.: \text{aggr-ref 8-25(71)}
\text{pre-eval 8-11(26)}

(102) \[ \text{list-dcp}(\text{dop}) = \]
\[ \text{is-intg-val}\times\text{s-1bd}(\text{dop}) \rightarrow \text{CONC list-dop}\times\text{elem}(i - \text{s-1bd}(\text{dop}) + 1, \text{s-elm}(\text{dop})) \]
\[ \text{i}\times\text{s-elm}(\text{dop}) \]
\[ \text{is-list}(\text{dop}) \rightarrow \text{CONC list-dop}\times\text{elem}(i, \text{dop}) \]
\[ \text{is-cp}(\text{dop}) \rightarrow <\text{dop}> \]

Ref.: \text{is-intg-val 9-3(5)}
\text{is-op 9-9(34)}

(103) \[ \text{entry-val-list}(\text{opl}) = \]
\[ \text{length}(\text{opl}) \]
\[ \text{Et} \text{is-ENTRY}\times\text{s-das-eva-sel}(i, \text{opl}) \rightarrow \text{LIST op-val}\times\text{elem}(i, \text{cpl}) \]
\[ \text{T} \rightarrow \text{error} \]

Ref.: \text{op-val 9-9(36)}

(104) \[ \text{fetch-release-1}(\text{entry-list}, \text{type}) = \]
\[ \text{is-FETCH}(\text{type}) \rightarrow \text{fetch}(\text{entry-list}) \]
\[ \text{is-RELEASE}(\text{type}) \rightarrow \text{release}(\text{entry-list}) \]

(105) \[ \text{fetch}(\text{entry-list}) = \]
\[ \text{null} \]

(106) \[ \text{release}(\text{entry-list}) = \]
\[ \text{null} \]

6. FLOW OF CONTROL WITHIN A SINGLE TASK 37
6.3 SEQUENTIAL INTERPRETATION OF STATEMENTS

This section defines the sequential flow of control within one block activation through the nested structure of statement lists and if-statements. This flow is governed by the control information CI. Section 6.3.1 defines the general mechanism of the sequential flow and in particular the interpretation of statement lists. Section 6.3.2 defines the if-statement and section 6.3.3 the general actions of a single statement and in particular the case distinction of proper statements.

Metavariables

<table>
<thead>
<tr>
<th>Metavariable</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>st</td>
<td>is-st</td>
</tr>
<tr>
<td>t</td>
<td>is-prop-st</td>
</tr>
<tr>
<td>index</td>
<td>is-index</td>
</tr>
<tr>
<td>expr</td>
<td>is-expr</td>
</tr>
<tr>
<td>op</td>
<td>is-op</td>
</tr>
<tr>
<td>i</td>
<td>is-intg-val</td>
</tr>
<tr>
<td>truth</td>
<td>is-T, is-F</td>
</tr>
</tbody>
</table>

6.3.1 STATEMENT LIST

The control information CI contains as its s-text component the innermost nested statement list or if-statement the component statements of which are currently interpreted. The s-index component of CI is an index uniquely localizing the statement currently being interpreted within the s-text component: If the text t is a statement list the index is an integer value i denoting its i-th element; if the text t is an if-statement (or access statement) the index is a truth value T or F denoting its s-then or s-else component. This localization is defined by the function take-st(t,index).

The nested structure of statement lists and if-statements is reflected by the components s-ci(CI) and s-c(CI) which essentially contain the control information and control of the state immediately before entering the current level of the structure; they are to be reinstalled after leaving the current level. So the control information CI acts as a dump.

Whenever a statement is completed, the instruction int-next-st is interpreted. It increases the index by one, if the text is a statement list which is not yet exhausted, or it returns to the former level of the statement structure by reinstalling the former CI and C.

Whenever a statement list or if-statement alternative is interpreted, the control information CI and control C are stacked into CI, the statement list or if-statement is entered as s-text component of CI and the starting index (1 in the case of the statement list) is entered as index into CI.
ABSTRACT SYNTAX AND INTERPRETATION OF PL/I

30 April 1969

(107) \texttt{int-st-list(t) =}
\begin{verbatim}
  stack-ci(1,t)
end for:is-st-list(t)
\end{verbatim}

(108) \texttt{stack-ci(index,t) =}
\begin{verbatim}
  s-ci:mu(<s-text:t>,<s-index:index>,<s-ci:CI>,<s-c:C>)
  s-c:continue
\end{verbatim}

(109) \texttt{continue =}
\begin{verbatim}
  int-next-st:
  int-st(take-st(s-index(\texttt{CI}),s-text(\texttt{CI})))
\end{verbatim}

(110) \texttt{take-st(index,t) =}
\begin{verbatim}
  is-st-list(t) & is-intg-val(index) & 1 \leq index \leq \text{length}(t) \rightarrow \text{elem}(index,t)
  is-if-st(t) & is-T(index) \rightarrow s-then(t)
  (is-if-st \lor is-access-st)(t) & is-T(index) \rightarrow s-else(t)
  T \rightarrow \text{error}
\end{verbatim}

Ref.: \texttt{is-intg-val 9-3(5)}

Note: Applied during the goto statement the tests made by this function ensure that the index list of the label is correct and that no forbidden gotos into iterated groups, blocks or units are performed (cf. 6.4).

(111) \texttt{int-next-st =}
\begin{verbatim}
  is-intg-val\times s-index(\texttt{CI}) \times s-index(\texttt{CI}) \times \text{length}\times s-text(\texttt{CI}) \rightarrow
  \begin{verbatim}
  continue:
  upd-index
  T \rightarrow
  s-ci:s-ci(\texttt{CI})
  s-ci:s-c(\texttt{CI})
  \end{verbatim}
\end{verbatim}

Ref.: \texttt{is-intg-val 9-3(5)}

(112) \texttt{upd-index =}
\begin{verbatim}
  s-ci:mu(\texttt{CI}:<s-index:s-index(\texttt{CI}) + 1>)
\end{verbatim}

6.3.2 THE IF-STATEMENT

The interpretation of the if-statement consists of evaluation of the expression into a truth value and interpretation of the alternative statement denoted by this truth value. This is done by introducing a new level into the control information \texttt{CI} as described in section 6.3.1, and using the evaluated truth value as new index component of \texttt{CI}. (Note that by the translation into an abstract program each
if-statement has an else-alternative: The null statement if nothing else was specified in the concrete program).

(113) \[ \text{int-if-st}(t) = \]
\[ \text{stack-ci}(\text{truth}, t); \]
\[ \text{truth}: \text{eval-truth}(s-\text{expr}(t)) \]
\[ \text{for} : \text{is-if-st}(t) \]

(114) \[ \text{eval-truth}(\text{expr}) = \]
\[ \text{pass-truth-val(op-1)}; \]
\[ \text{op-1}: \text{convert-1}(\text{aggr-scalar}(\text{BIT-IDA}), \text{op}); \]
\[ \text{op}: \text{eval-expr}(\text{expr}, 0) \]

Ref.: \text{convert-1} 9-29(119)  
\text{aggr-scalar} 8-23(64)  
\text{eval-expr} 8-10(24)  

(115) \[ \text{truth-val}(\text{op}) = \]
\[ (3i) (1 \leq i \leq \text{length} \cdot \text{op-val}(\text{op}) \& \text{is-1-BIT} \cdot \text{elem}(i, \text{op-val}(\text{op}))) \]

Ref.: \text{op-val} 9-9(36)  

6.3.3 INTERPRETATION OF A SINGLE STATEMENT

The interpretation of a single statement consists of updating the statement prefix part of the condition status CS according to the specified condition part of the statement (cf. chapter 10), raising the check condition for the statement labels and interpretation of the proper statement by the instruction int-prop-st which branches into the individual statement interpreting instructions according to the case distinction of the different statement types.

(116) \[ \text{int-st}(st) = \]
\[ \text{int-prop-st}(s-\text{prop-st}(st)); \]
\[ \text{call-check-cond}(\text{label-check-list}_1); \]
\[ \text{upd-st-cs}(s-\text{ons-cond-part}(st), s-no\text{-}s\text{-cond-part}(st)) \]

where:
\[ \text{label-check-list}_1 = \text{LIST} \text{id-ref}\cdot\text{elem}(i, s\text{-}label\text{-}list(st)) \]

Ref.: \text{call-check-cond} 10-21(64)  
\text{upd-st-cs} 10-1(6)  

(117) \[ \text{id-ref}(\text{idr}) = \]
\[ \mu_0(<s\text{-id-list}:<s\text{-id}(\text{idr})>>, <s\text{-n:s\text{-n}(\text{idr})}>, <s\text{-sl}>>) \]

Ref: \text{id-ref}(\text{idr})  

40 6. FLOW OF CONTROL WITHIN A SINGLE TASK
Note: This function constructs a valid reference from an object idr satisfying
the predicate is-id-ref which is essentially a single identifier.

(118) \( \text{int-prop-st}(t) = \)

\[
\begin{align*}
\text{is-block}(t) & \rightarrow \text{int-block}(t) \\
\text{is-group}(t) & \rightarrow \text{int-group}(t) \\
\text{is-st-list}(t) & \rightarrow \text{int-st-list}(t) \\
\text{is-if-st}(t) & \rightarrow \text{int-if-st}(t) \\
\text{is-goto-st}(t) & \rightarrow \text{int-goto-st}(t) \\
\text{is-call-st}(t) & \rightarrow \text{int-call-st}(t) \\
\text{is-return-st}(t) & \rightarrow \text{int-return-st}(t) \\
\text{is-incorporate-st}(t) & \rightarrow \text{int-incorporate-st}(t) \\
\text{is-fetch-st}(t) & \rightarrow \text{int-fetch-st}(t) \\
\text{is-release-st}(t) & \rightarrow \text{int-release-st}(t) \\
\text{is-wait-st}(t) & \rightarrow \text{int-wait-st}(t) \\
\text{is-delay-st}(t) & \rightarrow \text{int-delay-st}(t) \\
\text{is-exit-st}(t) & \rightarrow \text{int-exit-st} \\
\text{is-stop-st}(t) & \rightarrow \text{int-stop-st} \\
\text{is-assign-st}(t) & \rightarrow \text{int-assign-st}(t) \\
\text{is-allocate-st}(t) & \rightarrow \text{int-allocate-st}(t) \\
\text{is-free-st}(t) & \rightarrow \text{int-free-st}(t) \\
\text{is-cr-st}(t) & \rightarrow \text{int-cr-st}(t) \\
\text{is-revert-st}(t) & \rightarrow \text{int-revert-st}(t) \\
\text{is-signal-st}(t) & \rightarrow \text{int-signal-st}(t) \\
\text{is-access-st}(t) & \rightarrow \text{int-access-st}(t) \\
\text{is-enable-st}(t) & \rightarrow \text{int-enable-st}(t) \\
\text{is-disable-st}(t) & \rightarrow \text{int-disable-st}(t) \\
\text{is-open-st}(t) & \rightarrow \text{int-open-st}(t) \\
\text{is-close-st}(t) & \rightarrow \text{int-close-st}(t) \\
\text{is-get-st}(t) & \rightarrow \text{int-get-st}(t) \\
\text{is-put-st}(t) & \rightarrow \text{int-put-st}(t) \\
\text{is-read-st}(t) & \rightarrow \text{int-read-st}(t) \\
\text{is-write-st}(t) & \rightarrow \text{int-write-st}(t) \\
\text{is-rewrite-st}(t) & \rightarrow \text{int-rewrite-st}(t)
\end{align*}
\]

cont'd
6.4 THE GOTO STATEMENT

The denotation of a label consists of two components: A unique block activation name identifying the block activation into which the goto shall lead, and an index list localizing the statement to which the goto shall lead relative to the statement list of the block.

The general case of the goto statement is performed in four steps:

1) The block activations reflected in the dump $D$ are terminated one by one until the block activation into which the goto shall lead is the current one.

2) The levels of statement lists and if-statements reflected in the control information $CI$ are terminated one by one until the statement to which the goto shall lead is contained (possibly at a nested level) in the current text component of $CI$.

3) New levels of statement lists and if-statements are built up in the control information $CI$ according to the index list of the label, simulating the situation which would have occurred if these levels would have been entered by the normal flow of control (cf. 6.3.1).

4) This is done level by level, until the statement to which the goto shall lead is one of the immediate components of the current text component of $CI$.

The index component of $CI$ is adjusted so that it denotes the statement to which the goto shall lead.
After these four steps (some of them may be no action in special cases) the
flow of control continues in the normal way as described in section 6.3.1.

**Metavariables**

- \( t \): the interpreted goto statement
- \( da \): data attribute of label
- \( op \): label operand
- \( den \): label denotation
- \( indl \): index list denoting the statement denoted by the label
- \( index \): index denoting the statement denoted by the label
- \( ci \): control information CI
- \( block-act \): type of block activation:
  - \( is-BLOCK \)
  - \( is-PROC \)
  - \( is-ON \)
  - \( is-\) block-act(?)
- \( cbif \): condition built-in function part of the condition status CS

\[(115) \quad \text{int-goto-st}(t) =
\]
\[
goto-1(den,2,0) ;
\]
\[
den : \text{text-label}(op, da \text{• aggr-ref}(s-label(t), AT));
\]
\[
op : \text{eval-ref}(s-label(t))
\]

Ref.:  
\[
\text{aggr-ref } 8-25(71)
\]
\[
\text{eval-ref } 8-20(54)
\]

\[(120) \quad \text{text-label}(op, da) =
\]
\[
- \text{is-LABEL} \text{• da} \text{• eval}(op) \lor - \text{is-label-den}(den_i) \rightarrow \text{ERROR}
\]
\[
is-LABEL(da) \rightarrow \text{PASS} : den_i
\]
\[
(3i) (1 \leq i \leq \text{length of s-label-list}(da)) \land \text{den}_i = \text{label-den}_i \rightarrow \text{PASS} : den_i
\]
\[
T \rightarrow \text{ERROR}
\]

where:
\[
\text{den}_i = \text{cp-val}(op) (\text{DN})
\]
\[
\text{label-den}_i = \text{(is-label-den}(\text{den}_i) \rightarrow \text{den}_i,
\]
\[
T \rightarrow \text{error}
\]
\[
\text{den}_i = \text{s-nlelen}(i, \text{s-label-list}(da)) (\text{DN})
\]

cont'd
Note: It is tested whether the value of the operand yields really a label denotation and whether in the case of a restricted label variable the label denotation is in its range.

(121) \( \text{goto-1(den,block-act,cbif)} = \)

\[ \text{is-\(O\)(EA)} \to \text{error} \]
\[ \text{s-be(den)} = \text{EA} \to \]
\[ \text{goto-2(s-st-loc(den))}; \]
\[ \text{io-on-actions(block-act,cbif)} \]

\[ T \to \]
\[ \text{s-d:u}(U);<s-c: \text{goto-1(den,s-block-act(EA),s-cbif(CS))}> \]
\[ s-c:\text{epilogue}; \]
\[ \text{io-on-actions(block-act,cbif)} \]

Ref.: \text{epilogue} 6-11(31)

Note: The error test prohibits a goto out of a task. In certain cases, after terminating an on-unit raised by an input/output or attention condition special actions have to be performed; this is done by the following instruction.

(122) \text{io-on-actions(block-act,cbif)} =

\[ \text{is-ON(block-act)} \& \text{-is-\(O\)s-en(ar_4)} \to \]
\[ \text{revert-onfile} ; \]
\[ \text{delete-task-event(s-en(ar_4),NORMAL)} \]

\[ \text{is-ON(block-act)} \& \text{-is-\(O\)s-onident(cbif)} \to \]
\[ \text{revert-onfile} ; \]
\[ \text{call-\(O\)-cond-1(tmt)} ; \]
\[ \text{de-lab-1(s-u(ar_4),s-cond(ar_4),tmt)} ; \]
\[ \text{tmt:wr-ds-lab(s-u(ar_4),s-cond(ar_4),op)} ; \]
\[ \text{op:gen-op(s-gen*s-onident(cbif))} \]

\[ \text{is-ON(block-act)} \& \text{-is-\(O\)s-cond(cbif)} \to \]
\[ \text{revert-onfile} ; \]
\[ \text{change-enab-1(s-cond(cbif))} \]

\[ T \to \text{revert-onfile} \]

where:
\[ ar_4 = s-abn-ret(cbif) \]

Ref.: \text{revert-onfile} 11-40(107) \text{delete-task-event} 5-8(28) \text{call-\(O\)-cond-1} 10-24(72) \text{ds-lab-1} 17-34(95) \text{wr-ds-lab} 11-35(94) \text{gen-op} 8-22(59) \text{change-enab-1} 10-15(44)
ABSTRACT SYNTAX AND INTERPRETATION OF PL/I

(123) goto-2(indl) =
                (3list)(length(list) ≥ 1 & ci-indl•s-ci(c1)•list = indl) ↔
                goto-3((list)(ci-indl•s-ci(c1)•list = indl))
                T ↔
                s-ci:s-ci(c1)
                s-c:goto-2(indl)

for:is-index-list(list)

(124) ci-indl(ci) =
                is-(#)ci ↔ <>
                T ↔ ci-indl•s-ci(ci)•<s-index(ci)>

(125) goto-3(indl) =
                is-<>(tail(indl)) ↔ goto-4(index1)
                T ↔
                s-ci:μ(μ<s-text>s-prop-st(st1)>,<s-ci:μ(c1;<s-index:index1>>>,
                <s-ci:int-next-st>)
                s-c:goto-3(tail(indl))

where:
index1 = head(indl)
st1 = take-st(index1,s-text(c1))

Ref.:  int-next-st 6-39(111)
take-st 6-39(110)

(126) goto-4(index) =
                s-ci:μ(c1;<s-index:index>)
                s-c:continue

Ref.:  continue 6-39(109)

6.5 GROUPS

In PL/I there are two possibilities to specify iterated execution of a
statement list: The while-group, defined in section 6.5.1, and the controlled
group, defined in section 6.5.2. The so-called "simple groups" are statement list
to be executed once only (cf. 6.3.1); they are not denoted as groups in this
document.

Metavariables
\[ \text{int-group}(t) = \]
\[ \text{is-while-group}(t) \rightarrow \text{int-while-group}(t) \]
\[ \text{is-contr-group}(t) \rightarrow \text{int-contr-do}(t, \varnothing) \]

\[ \text{(127)} \]

6.5.1 While-Group

A while group specifies that its statement list is executed repeatedly as long as the evaluation of the while expression yields "true".

\[ \text{(128)} \]

\[ \text{while-continue}(\text{truth}, t) = \]
\[ \text{while-continue}(\text{truth}, t) : \]
\[ \text{int-continue}(\text{truth}, \text{s-do-list}(t), \varnothing); \]
\[ \text{truth}\text{-eval-truth}(\text{s-while}(t)) \]

Ref.: \text{eval-true} 6-80 (114)

\[ \text{(129)} \]

\[ \text{while-continue}(\text{truth}, t) = \]
\[ -\text{truth} \rightarrow \text{null} \]
\[ T \rightarrow \text{int-while-group}(t) \]

46 6. FLOW OF CONTROL WITHIN A SINGLE TASK
6.5.2 CONTROLLED GROUP

A controlled group is executed by interpretation of a list of iteration specifications. Each single iteration specification controls iterated execution of the statement list of the group.

The interpretation of a single iteration specification is performed mainly by the instruction iterate-do: First, the controlling variable is compared with the value of the to-expression. Second, if the result of the comparison was "true", the while-expression is evaluated. If both the comparison and the while-expression evaluation yielded "true", third, the statement list is executed and, fourth, the iteration is continued by adding the value of the by-expression to the controlling variable and starting the circle again.

If the to-expression is missing, the compare is assumed always to yield "true". If the by-expression, but not the to-expression is missing, the by-expression is assumed to be one. If both the by-expression and to-expression are missing, the iteration is not continued in any case. If the while expression is missing, the while-expression evaluation is assumed to yield "true".

The iteration mechanism of the controlled group applies to controlled data item lists in get or put statements as well. In this case, a transmission parameter tr, which is needed for the interpretation of the data item list, is passed as an additional argument through the instructions of the group mechanism.

\[\text{(130)}\]
\[
\text{int-contr-do}(t, tr) =
\]
\[\text{iterate-spec-list}(s\text{-spec-list}(t), \text{gen, ref}, s\text{-do-list}(t), tr);\]
\[
\text{gen: eval}(lp)\left(\text{ref}\right);\]
\[
\text{ref: pre-eval}(s\text{-contr-var}(t))
\]

Ref.:
\[\text{eval-lp 8-9(20)}\]
\[\text{pre-eval 8-11(26)}\]

\[\text{(131)}\]
\[
\text{iterate-spec-list}(\text{spec-list, gen, ref, list, tr}) =
\]
\[\text{is-}\leftarrow(\text{spec-list}) \rightarrow \text{null}\]
\[\text{T} \rightarrow\]
\[\text{iterate-spec-list}(\text{tail(\text{spec-list})}, \text{gen, ref, list, tr});\]
\[\text{int-do-spec(\text{init}_1, \text{by}_1, \text{to}_1, \text{while}_1, \text{gen, ref, list, tr})}\]

where:
\[\text{init}_1 = s\text{-init-\text{head}}(\text{spec-list})\]
\[\text{by}_1 = s\text{-by-\text{head}}(\text{spec-list})\]
\[\text{to}_1 = s\text{-to-\text{head}}(\text{spec-list})\]
\[\text{while}_1 = s\text{-while-\text{head}}(\text{spec-list})\]
ABSTRACT SYNTAX AND INTERPRETATION OF PL/I
30 April 1969

(132) \texttt{int-do-spec} (init, by, to, while, gen, ref, list, tr) = 
\begin{align*}
& \text{(is-arithm v is-string v is-pic) s-da-aggr-lp(ref) v is-0(by) \& is-0(to)} \rightarrow \\
& \text{iterate-do(gen, by-to-ops, while, ref, list, tr);} \\
& \text{convert-assign(gen, cp, area}_1, area}_2); \\
& \text{by-to-ops:eval-by-te(by, to);} \\
& \text{op:eval-exp(init, s-da-aggr-lp(ref))}
\end{align*}

\[ T \rightarrow \text{error} \]

where:
\begin{align*}
\text{area}_1 &= s-area\cdot s-da\cdot aggr\cdot lp(ref) \\
\text{area}_2 &= \text{area-expr(init, AT)}
\end{align*}

Ref.: \texttt{aggr-lp} 8-4(4) \\
\texttt{convert-assign} 8-8(16) \\
\texttt{eval-exp} 8-10(24) \\
\texttt{area-expr} 8-28(84)

(133) \texttt{eval-by-to(by, to)} = 
\begin{align*}
& \text{is-0(by) \& is-0(to)} \rightarrow \\
& \text{pass(by-to-ops);} \\
& \text{s-by(by-to-ops):pass-intg-op()}, \\
& \text{s-to(by-to-ops):eval-exp(to, 2)}
\end{align*}

\[ T \rightarrow \]
\begin{align*}
& \text{pass(by-to-ops);} \\
& \text{s-by(by-to-ops):eval-opt-exp(by),} \\
& \text{s-to(by-to-ops):eval-opt-exp(to)}
\end{align*}

Ref.: \texttt{intg-op} 9-10(39) \\
\texttt{eval-exp} 8-10(24) \\
\texttt{eval-opt-exp} 8-22(61)

Note: The by- and to-expressions are to be evaluated in arbitrary order, but after the initial expression.

(134) \texttt{iterate-do(gen, by-to-ops, while, ref, list, tr)} = 
\begin{align*}
& \text{do-continue(truth, gen, by-to-ops, while, ref, list, tr);} \\
& \text{int-do-list(truth, list, tr);} \\
& \text{truth:eval-while(truth-1, while);} \\
& \text{truth-1:eval-comp(gen, by-to-ops);} \\
& \text{call-check-cond(<ref>)}
\end{align*}

Ref.: \texttt{call-check-cond} 10-21(64)
(135) \( \text{eval-comp}(\text{gen}, \text{by-to-ops}) = \)
\[ \text{is-ops-to(by-to-ops)} \rightarrow \text{PASS:T} \]
\[ T \rightarrow \]
\[ \text{pass-truth-val(op)} : \]
\[ \text{op}=\text{eval-infix-expr(op-1, s-to(by-to-ops), opr)}; \]
\[ \text{op-1}=\text{gen-op-1}(\text{gen}), \]
\[ \text{opr}=\text{comp-opr}(\text{op-2}); \]
\[ \text{op-2}=\text{eval-infix-expr(s-by(by-to-ops), intg-op(0), GE)} \]

Ref.: truth-val 6-40(115)
\[ \text{eval-infix-expr} 9-13(64) \]
\[ \text{intg-op} 9-10(39) \]

(136) \( \text{gen-op-1}(\text{gen}) = \)
\[ \text{is-gen}(\text{gen}) \rightarrow \text{gen-op}(\text{gen}) \]
\[ \text{is-ps-gen}(\text{gen}) \rightarrow \text{ps-gen-op}(s-id(\text{gen}), s-arg-list(\text{gen})) \]

Ref.: is-gen 3-14(30)
\[ \text{gen-op} 8-22(59) \]
\[ \text{is-ps-gen} 12-67(169) \]
\[ \text{ps-gen-op} 12-74(189) \]

(137) \( \text{comp-op}(\text{op}) = \)
\[ \text{truth-val(op)} \rightarrow \text{PASS:LE} \]
\[ T \rightarrow \text{PASS:GE} \]

Ref.: truth-val 6-40(115)

(138) \( \text{eval-while}(\text{truth}, \text{while}) = \)
\[ \neg\text{truth} \rightarrow \text{PASS:T} \]
\[ \text{is-0}(\text{while}) \rightarrow \text{PASS:T} \]
\[ T \rightarrow \text{eval-truth}(\text{while}) \]

Ref.: eval-truth 6-40(114)

(139) \( \text{int-do-list}(\text{truth}, \text{list}, \text{tr}) = \)
\[ \neg\text{truth} \rightarrow \text{null} \]
\[ \text{is-st-list}(\text{list}) \rightarrow \text{int-st-list}(\text{list}) \]
\[ \text{is-item-list}(\text{list}) \rightarrow \text{expand-item-list}(\text{tr}, \text{list}) \]

Ref.: int-st-list 6-38(107)
\[ \text{expand-item-list} 71-76(214) \]

6. FLOW OF CONTROL WITHIN A SINGLE TASK 49
(140) \( \text{de-continue}(\text{truth, gen, by-to-ops, while, ref, list, tr}) = \)
\( \text{~truth \& is-O(by-to-ops) \rightarrow null} \)
\( T \rightarrow \)
\( \text{iterate-de(gen, by-to-ops, while, ref, list, tr}); \)
\( \text{convert-assign(gen, op, O, O)}; \)
\( \text{op-2:eval-infix-expr(op-1, s-by(by-to-ops), ADD)}; \)
\( \text{op-1:gen-op-1(gen)} \)

Ref.: \text{convert-assign 8-8(16)} \text{eval-infix-expr 9-13(64)}

6.6 THE INCORPORATE STATEMENT

The incorporate statement specifies an implementation defined piece of non-PL/I text that is to be executed in implementation defined way. It is the responsibility of the implementation and of the programmer, when using this statement, to be aware that the state of the PL/I machine after execution of the statement is such that a senseful continuation of the computation of the PL/I program is possible.

(141) \( \text{int-incorporate-st(t)} = \)
\( \text{incorporate(s-text(t))} \)
\( \text{foris-incorporate-st(t)} \)

(142) \( \text{incorporate(text)} = \)

Note: This instruction is implementation defined. It interprets an implementation defined non-PL/I text.
This chapter defines the allocate statement, the actions performed on allocation of static, automatic, and dummy variables, the properties of storage mapping as far as they are guaranteed by the PL/I language, and the free statement. The elementary storage functions on which the definitions are built are described in 3.4.

Metavariables

<table>
<thead>
<tr>
<th>Metavariable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ag</td>
<td>aggregate directory</td>
</tr>
<tr>
<td>aggr</td>
<td>aggregate attributes</td>
</tr>
<tr>
<td>al</td>
<td>allocation attributes specified in an allocate statement</td>
</tr>
<tr>
<td>area</td>
<td>optional reference to an area</td>
</tr>
<tr>
<td>at</td>
<td>attribute directory</td>
</tr>
<tr>
<td>b</td>
<td>aggregate name</td>
</tr>
<tr>
<td>base</td>
<td>base of string data attribute</td>
</tr>
<tr>
<td>dens</td>
<td>density</td>
</tr>
<tr>
<td>dn</td>
<td>denotation directory</td>
</tr>
<tr>
<td>eva</td>
<td>evaluated aggregate attributes</td>
</tr>
<tr>
<td>extent</td>
<td>extent of based variable</td>
</tr>
<tr>
<td>gen</td>
<td>generation of variable</td>
</tr>
<tr>
<td>genl</td>
<td>generation list</td>
</tr>
<tr>
<td>i</td>
<td>integer value</td>
</tr>
<tr>
<td>init</td>
<td>initial attribute</td>
</tr>
<tr>
<td>j</td>
<td>integer value</td>
</tr>
<tr>
<td>list</td>
<td>operand</td>
</tr>
<tr>
<td>op</td>
<td>pointer value</td>
</tr>
<tr>
<td>ptr</td>
<td>optional pointer or offset reference</td>
</tr>
<tr>
<td>ref</td>
<td>reference</td>
</tr>
<tr>
<td>spec</td>
<td>extent specification</td>
</tr>
<tr>
<td>t</td>
<td>allocate or free statement</td>
</tr>
</tbody>
</table>

7. ALLOCATION, INITIALISATION, AND FREEING OF VARIABLES
THE ALLOCATE STATEMENT

The allocate statement is interpreted by the instruction
\[ \text{int-allocate-st}(t) = \\text{int-allocate-list}(s-\text{list}(t)) \]
for: is-allocate-st(t)

\[ \text{int-allocate-list}(t) = \]
\[ \text{is-\langle\rangle}(t) \rightarrow \text{null} \]
\[ (\exists i)(\text{is-ready-for-alloc}(t,i,\text{AT},\text{EN},\text{AG}) \rightarrow \]
\[ \text{null}; \]
\[ \text{int-allocate}(t,i) \land \text{is-ready-for-alloc}(t,i,\text{AT},\text{EN},\text{AG}) \]
\[ T \rightarrow \text{error} \]
for: is-al-list(t)

Note: The predicate is-ready-for-alloc determines those allocations which in their execution do not depend on unallocated controlled variables specified in the same statement.
(3) \[\textit{is-ready-for-alloc}(t, i, at, dn, ag) = \]
\[
\left(\left(\exists e \left(\textit{is-agg-elem}(i, t) \Rightarrow \textit{is-gen}(gen_1)\right) \&
\left(\forall ref\right)\textit{is-alloc-dep}(\textit{elem}(i, t), ref, at, dn, ag) \& \textit{is-CTLs-stg-cl}(\textit{attr-ref}_0) \&
\left(\exists j\right)\left(\textit{is-elem}(j, t) \Rightarrow \textit{is-gen-head}(\textit{s-n}(ref)(dn)(ag))\right)\right) \right)
\]

where:
- \(\text{gen}_1 = \text{head}(\textit{s-elem}(i, t)(dn)(ag))\)
- \(\text{attr-ref}_0 = \textit{s-n}(ref)(at)\)

for: \(\textit{is-al-list}(t)\)

Ref.: \(\textit{is-gen} 3-14(30)\)

Note: An allocation is ready for execution if none of the expressions to be evaluated on its execution refers to an unallocated controlled variable specified in the same statement and, if the variable to be allocated is itself controlled and extents are to be determined by its most recent generation, the variable is already allocated.

(4) \[\textit{is-alloc-dep}(t, ref, at, dn, ag) = \]
\[
\textit{is-dep}(\textit{agg}_0, ref, at) \lor \neg\textit{is-O}(\textit{ptr}_0) \& \textit{is-dep-2}(\textit{ptr}_0, ref, at) \lor \neg\textit{is-O}(\textit{area}_0) \& \textit{is-dep-2}(\textit{area}_0, ref, at)
\]

where:
- \(\textit{agg}_0 = \text{mix-spec}(\textit{s-agg}(\textit{attr}_0), \textit{s-agg}(t), \textit{s-eva}(\textit{gen}_0))\)
- \(\textit{attr}_0 = \textit{s-n}(t)(at)\)
- \(\textit{gen}_0 = \text{head}(\textit{s-n}(t)(dn)(ag))\)
- \(\textit{ptr}_0 = (\neg\textit{is-O} \Rightarrow \textit{s-ptr}(t) \Rightarrow \textit{s-ptr}(\textit{attr}_0))\)
- \(\textit{area}_0 = (\neg\textit{is-O} \Rightarrow \textit{s-area}(t) \Rightarrow \textit{s-area}(t)\),
\(T \Rightarrow \textit{area-expr}(\textit{ptr}_0, at)\))

for: \(\textit{is-al}(t)\)

Ref.: \(\textit{is-dep} 6-6(16)\)
\(\textit{is-dep-2} 6-6(18)\)
\(\textit{area-expr} 8-28(81)\)

(5) \[\textit{int-allocate}(t, i) = \]
\[
\textit{s-C:int-Next-St};
\textit{int-allocate-list}(\textit{remove}(t, i));
\textit{int-allocate-1}(\textit{elem}(i, t))
\]

for: \(\textit{is-al-list}(t)\)

Ref.: \(\textit{int-Next-St} 6-39(111)\)

Note: Once an allocation has been selected, it is interpreted by the instruction \(\textit{int-allocate-1}\), and the process of testing and selection is restarted with the list of remaining allocations.

7. ALLOCATION, INITIALISATION, AND FREETING OF VARIABLES 3
(6)  \text{remove}(\text{list}, i) = \\
\text{i} = 1 \rightarrow \text{tail(list)} \\
T \rightarrow \langle \text{head(list)} \rangle \text{remove}(\text{tail(list)}, i - 1)

(7)  \text{int-allocate-1}(t) = \\
is-\text{CTL} \ast \text{sta-cl}(\text{attr}_0) \& \text{is-\text{aggr}}(\text{attr}_0) \& \text{is-\text{area}}(\text{attr}_0) \rightarrow \\
\text{initialize}(\text{gen}, \text{aggr}_0): \\
\text{gen-allocate-cil}(\text{attr}_0, \text{eva}); \\
\text{eva-eval-aggr}(); \\
is-\text{based}(\text{attr}_0) \& \text{is-\text{aggr}}(\text{attr}_0) \& \text{is-\text{area}}(\text{attr}_0) \& \text{is-\text{var-ref}}(\text{ptr}_0, \text{AT}) \& \\
is-\text{PPS-s-aggr-ref}(\text{ptr}_0, \text{AT}) \rightarrow \\
\text{initialize}(\text{gen}, \text{aggr}_0): \\
\text{set-and-refer}(\text{ptr}_0, \text{gen}, \text{eva}, \text{aggr}_0); \\
\text{gen-allocate-based}(\text{eva}); \\
\text{eva-eval-alloc-aggr}(); \\
is-\text{based}(\text{attr}_0) \& \text{is-\text{aggr}}(\text{attr}_0) \& \text{is-\text{var-ref}}(\text{ptr}_0, \text{AT}) \& \\
is-\text{var-ref}(\text{area}_0, \text{AT}) \rightarrow \\
\text{initialize}(\text{gen}, \text{aggr}_0): \\
\text{set-and-refer}(\text{ptr}_0, \text{gen}, \text{eva}, \text{aggr}_0); \\
\text{gen-allocate-area}(); \\
\text{eva-eval-alloc-aggr}(); \\
\text{gen-1-eval-ref-gen}(\text{area}_0) \\
T \rightarrow \text{S-REF}

where:
\text{attr}_0 = \text{s-n}(t) \\
\text{aggr}_0 = \text{s-aggr}(\text{attr}_0) \\
\text{gen}_0 = \text{head}(\text{attr}_0) \rightarrow \text{AG} \\
\text{ptr}_0 = (-\text{is-\text{aggr}}(\text{attr}_0) \rightarrow \text{ptr}(t), \\
T \rightarrow \text{s-ptr}(\text{attr}_0)) \\
\text{area}_0 = (-\text{is-\text{area}}(\text{attr}_0) \rightarrow \text{area}(t), \\
T \rightarrow \text{area-expr}(\text{ptr}_0, \text{AT}))

for: is-\text{are}(t)

Ref.: \text{initialize} 7-15(69) \\
\text{eval-aggr} 6-9(23) \\
\text{is-\text{var-ref}} 8-9(23) \\
\text{aggr-ref} 8-25(71) \\
\text{eval-ref-gen} 8-29(82) \\
\text{area-expr} 8-28(81)

7.1.1 ALLOCATION OF CONTROLLED VARIABLES

Specification of array bounds, string lengths, and area sizes used for the allocation are mixed from those given in the allocate statement, in the declaration of the controlled variable, and in the most recent generation (if existent) of the controlled variable. Rules for overriding specifications are given by the function mix-spec.

7. ALLOCATION, INITIALISATION, AND FREEING OF VARIABLES
ABSTRACT SYNTAX AND INTERPRETATION OF PL/I

The instruction allocate-ctl enters the pointer of the new generation in the allocation state of the main storage S, and puts the new generation on top of the stack of generations associated with the variable in the aggregate directive AG. It also returns the new generation for the use in the instruction initialize.

(8) allocate-ctl(h,eva) =
  - is-correct-eva(eva) → error
  is-free-space(p1,S,CTL) →
    PASS: mk-gen(eva,p1)
    S = el-alloc(p1,E,CTL)
    S-ag:p(AG;ch:mk-gen(eva,p1)>"h(AG)"
  T → stg-overflow

where:
  p1 = alloc-space(alloc-size(eva),S,CTL)

Ref.: is-free-space 3-17(56)
  el-alloc 3-17(55)
  alloc-space 7-11(28)
  alloc-size 7-11(26)

Note: p1 is the pointer value identifying that part of S which is reserved on allocation. The kind of storage reserved on allocation, and the predicate is-free-space, testing whether the allocation is possible, are described in 7.3.1.

(9) is-correct-eva(eva) =
  is-array(eva) →
  is-intg-vars-lbd(eva) & is-intg-vars-ubd(eva) & s-lbd(eva) ≤ s-ubd(eva) &
  is-correct-eva-s-elem(eva)

  length(eva)
  is-struct(eva) → Et is-correct-eva-s-aggr-elem(i,eva)
  i=1

  is-scalar(eva) → is-correct-edas-da(eva)

Ref.: is-intg-val 9-3(5)
  is-correct-edas 9-2(1)

(10) mk-gen(eva,p) =
  is-correct-eva(eva) → po(<s-eava:eva>,<s-mi:eva>,<s-pp:p>)
  T → error

(11) stg-overflow =

Note: This instruction is implementation defined. It comprises actions performed on storage overflow.

7. ALLOCATION, INITIALISATION, AND FREEING OF VARIABLES
ABSTRACT SYNTAX AND INTERPRETATION OF PL/I

30 April 1969

(12) \(\text{mix-spec}(\text{aggr},\text{al},\text{eva}) = \)
\begin{align*}
&\text{is-0(}\text{al}\text{)} \rightarrow \text{aggr} \\
&\text{is-scalar} \cdot \text{base-elem}(\text{al}) \rightarrow \text{mix-spec-1}(\text{aggr},\text{al},\text{eva}) \\
&T \rightarrow \text{mix-spec-2}(\text{aggr},\text{al},\text{eva})
\end{align*}
for: (is-eva \lor is-aggr \lor is-0)(eva)

Ref.: base-elem 8-29(85)

(13) \(\text{mix-spec-1}(\text{aggr},\text{al},\text{eva}) = \)
\begin{align*}
&\text{is-array}(\text{aggr}) \& \text{is-array}(\text{al}) \& \text{dim}(\text{aggr}) = \text{dim}(\text{al}) \rightarrow \\
&\mu_0(\langle s-lbd:\text{mix}(s-lbd(\text{al}),s-lbd(\text{eva})), s-ubd:\text{mix}(s-ubd(\text{al}),s-ubd(\text{eva})), s-elem:\text{mix-spec-1}(s-elem(\text{aggr}),s-elem(\text{al}),s-elem(\text{eva})) \rangle) \\
&\text{is-array}(\text{aggr}) \& \text{is-scalar}(\text{al}) \\
&\mu(\text{aggr};\langle s-elem:\text{mix-spec-1}(s-elem(\text{aggr}),\text{al},s-elem(\text{eva})) \rangle) \\
&\text{is-scalar}(\text{aggr}) \rightarrow \text{mix-spec-scalar}(\text{aggr},\text{al},\text{eva}) \\
&T \rightarrow \text{error}
\end{align*}
for: (is-eva \lor is-aggr \lor is-0)(eva)

Ref.: dim 8-29(86)

(14) \(\text{mix-spec-2}(\text{aggr},\text{al},\text{eva}) = \)
\begin{align*}
&\text{is-array}(\text{aggr}) \& \text{is-array}(\text{al}) \& \text{dim}(\text{aggr}) = \text{dim}(\text{al}) \rightarrow \\
&\mu_0(\langle s-lbd:\text{mix}(s-lbd(\text{al}),s-lbd(\text{eva})), s-ubd:\text{mix}(s-ubd(\text{al}),s-ubd(\text{eva})), s-elem:\text{mix-spec-2}(s-elem(\text{aggr}),s-elem(\text{al}),s-elem(\text{eva})) \rangle) \\
&\text{is-struct}(\text{aggr}) \& \text{is-struct}(\text{al}) \& \text{length}(\text{aggr}) = \text{length}(\text{al}) \rightarrow \\
&\text{length}(\text{aggr}) \text{ LIST } \mu_0(\langle s-aggr:\text{mix-spec-2}(s-aggr\cdot\text{elem}(i,\text{aggr}),s-aggr\cdot\text{elem}(i,\text{al}), \text{aggr}\cdot\text{elem}(i,\text{eva})) \rangle) \\
&\text{is-scalar}(\text{aggr}) \& \text{is-scalar}(\text{al}) \rightarrow \text{mix-spec-scalar}(\text{aggr},\text{al},\text{eva}) \\
&T \rightarrow \text{error}
\end{align*}
for: (is-eva \lor is-aggr \lor is-0)(eva)

Ref.: dim 8-29(86)

6 7. ALLOCATION, INITIALISATION, AND FREEING OF VARIABLES
7.1.2 ALLOCATION IN AREAS

The aggregate attributes of the based variable to be allocated are evaluated by the instruction eval-alloca-aggr. This evaluation implies the evaluation of the expressions in the REFER-options which specify extents of the variables.

The generation of the area in which the allocation is made is obtained by evaluating the s-area component, or, if this is not specified, by evaluating the area reference declared with the offset variable specified in the set reference. The set reference is obtained from the s-trr component, or, if this is not specified, from the pointer reference declared with the based variable. The instruction allocate-area enters the offset identifying the new generation in the allocation state of the area, and returns the new generation for the use in the instruction initialize. If the allocation is not possible the AREA condition is raised. After return from the on-unit the area generation is re-evaluated and the allocation is tried again. Immediately after the allocation the pointer value identifying the new generation is assigned to the generation of the set reference, and the integer values of the expressions specified in the REFER-options are assigned to the respective parts of the new generation.
(18) \( \text{eval-alloc-aggr}(\text{aggr}) = \)

\[
\text{is-array}(\text{aggr}) \rightarrow \\
\quad \text{pass}(\text{eva}) ; \\
\quad s-\text{lbd}(\text{eva}) : \text{eval-alloc-extent}(s-\text{lbd}(\text{aggr})) , \\
\quad s-\text{ubd}(\text{eva}) : \text{eval-alloc-extent}(s-\text{ubd}(\text{aggr})) , \\
\quad s-\text{elem}(\text{eva}) : \text{eval-alloc-aggr}(s-\text{elem}(\text{aggr})) \\
\]

\[
\text{is-struct}(\text{aggr}) \rightarrow \\
\quad \text{pass}(\text{eva}) ; \\
\quad (s-\text{aggr} \cdot \text{elem}(i))(\text{eva}) : \text{eval-alloc-aggr}(s-\text{aggr} \cdot \text{elem}(i,\text{aggr})) \\ 1 \leq i \leq \text{length}(\text{aggr}) \\
\]

\[
\text{is-string} \cdot s-\text{da}(\text{aggr}) \rightarrow \\
\quad \text{pass}(\text{eva}) ; \\
\quad s-\text{length} \cdot s-\text{da}(\text{eva}) : \text{eval-alloc-extent}(s-\text{length} \cdot s-\text{da}(\text{aggr})) ; \\
\quad \text{eva} : \text{pass}(\delta(\text{aggr};s-\text{init})) \\
\]

\[
\text{is-area} \cdot s-\text{da}(\text{aggr}) \rightarrow \\
\quad \text{pass}(\text{eva}) ; \\
\quad s-\text{size} \cdot s-\text{da}(\text{eva}) : \text{eval-alloc-extent}(s-\text{size} \cdot s-\text{da}(\text{aggr})) ; \\
\quad \text{eva} : \text{pass}(\delta(\text{aggr};s-\text{init})) \\
\]

\[
T \rightarrow \text{eval-aggr}(\text{aggr}) \\
\]

Ref.: \( \text{eval-aggr} 6-9(23) \)

(19) \( \text{eval-alloc-extent}(\text{extent}) = \)

\[
\text{is-refer}(\text{extent}) \rightarrow \text{eval-intg-expr}(s-\text{expr}(\text{extent})) \\
\]

\[
T \rightarrow \text{pass-intg-val}(\text{extent}) \\
\]

Ref.: \( \text{eval-intg-expr} 8-22(60) \)
\( \text{intg-val} 6-10(27) \)

(20) \( \text{allocate-area}(\text{gen},\text{eva},\text{area}) = \)

\[
\neg \text{is-area} \cdot \text{gen} \cdot \text{da}(\text{gen}) \lor \neg \text{is-correct-eva}(\text{eva}) \rightarrow \text{ERROR} \\
\text{is-free-space}(o_1,p_1(\text{S}),\text{AREA}) \rightarrow \\
\quad \text{pass} : \text{mk-gen}(\text{eva},o_1,p_1) \\
\quad s-\text{get-asc}(\text{el-alloc}(o_1,p_1(\text{S}),\text{AREA}),p_1,\text{S}) \\
\quad T \rightarrow \\
\quad \text{allocate-area}(\text{gen-1},\text{eva},\text{area}) ; \\
\quad \text{gen-1} : \text{eval-ref-gen}(\text{area}) ; \\
\quad \text{call-cond}(\text{AREA}) \\
\]

where:
\[
o_1 = \text{alloc-space}(\text{alloc-size}(\text{eva}),p_1(\text{S}),\text{AREA}) \\
p_1 = s-\text{pp}(\text{gen}) \\
\]

Ref.: \( \text{gen-\text{da} 7-17(53)} \)
(27) \( \text{is-free-space} 3-17(56) \)
(48) \( \text{el-ass} 3-16 \)

8 7. ALLOCATION, INITIALISATION, AND FREEING OF VARIABLES cont'd
el-alloc 7-17(5)
eval-ref-gen 8-28(62)
call-ccnd 10-18(54)
alloc-space 7-17(28)
alloc-size 7-11(26)

Note: o is the pointer value identifying that part of the area which is reserved on allocation.

(21) set-and-refer(ptr, gen, eva, aggr) =

-is-g(ptr) --
call-check-cond(<ptr>);
initialize-refer(gen, eva, aggr),
convert-assign(gen-1, val-op(PTR, s-pp(gen)), area1, 6);
gen-1:eval-ref-gen(ptr)
T -- error

where:
area1 = area-expr(ptr, #)

Ref.: call-check-cond 10-21(64)
convert-assign 8-8(16)
val-op 9-5(76)
eval-ref-gen 8-28(62)
area-expr 8-28(81)

(22) initialize-refer(gen, eva, aggr) =

null:
{convert-assign(gen-s, cp-s, 0, 0) | is-ref*(aggr)}

where:
gen-s = sub-gen(gen, e1-s)
e1-s = eval-al(s-refer*{aggr}, aggr)

Ref.: convert-assign 8-8(76)
sub-gen 8-36(115)
eval-al 8-30(54)
val-op 9-9(38)

7.1.3 ALLOCATION OF BASED VARIABLES IN MAIN STORAGE

The aggregate attributes of the based variables and the set reference are evaluated as for allocation in areas. The instruction allocate-based enters the pointer value identifying the new generation in the allocation state of the main storage S. The pointer value is also entered into the free-set of the current task to ensure freeing of the based storage at task end. The instruction returns the new generation for the use in the instruction initialize.
(23) **allocate-based**(eva) =
    
    \( \text{is-correct-eva}(eva) \rightarrow \text{error} \)
    
    \( \text{is-free-space}(p_1, S, \text{BASED}) \rightarrow \)
    
    \( \text{PASS}: \text{mk-gen}(eva, p_1) \)
    
    \( S = S \cup \text{el-alloc}(p_1, S, \text{BASED}) \)
    
    \( S = S \cup \mu(T_b) < S \text{-free-set} \cup S \text{-free-set}(T_b) \cup \{p_1\} \)
    
where:

\( p_1 \) = alloc-space(alloc-size(eva), S, BASED)

Ref.: 

* is-free-space 3-17(56)
* el-alloc 3-17(55)
* alloc-space 7-11(28)
* alloc-size 7-11(26)

Note: \( p_1 \) is the pointer value identifying that part of \( S \) which is reserved on allocation.

7.2 ALLOCATION OF AUTOMATIC, STATIC, AND DUMMY VARIABLES

The instruction **allocate-auto**(b,eva) is used for allocating automatic variables, where \( b \) is the aggregate name of the variable and eva are its evaluated aggregate attributes. The instruction enters the pointer value identifying the new generation in the allocation state of the main storage \( S \). The new generation is entered in the aggregate directory \( AG \) under the aggregate name \( b \). The pointer value is entered also in the free-set of the epilogue information \( \text{FI} \) to ensure freeing of the automatic storage at block end. The instruction returns the new generation for use in the initializing instruction.

The instruction **allocate**(b,eva,type) is used for allocating static and dummy variables. It performs the same actions as the instruction **allocate-auto** except that the new pointer value is not entered in the free set, and that the new generation is entered in \( AG \) on top of the null generation (represented as \( \ast \)).

(24) **allocate-auto**(b,eva) =
    
    \( \text{is-correct-eva}(eva) \rightarrow \text{error} \)
    
    \( \text{is-free-space}(p_1, S, \text{AUTO}) \rightarrow \)
    
    \( \text{PASS}: \text{mk-gen}(eva, p_1) \)
    
    \( S = S \cup \text{el-alloc}(p_1, S, \text{AUTO}) \)
    
    \( S = S \cup \mu(\text{AG}; < \text{mk-gen}(eva, p_1)\{b\}) \)
    
    \( S = S \cup \mu(\text{FI}; < S \text{-free-set} \cup S \text{-free-set}(\text{FI}) \cup \{b\} \}
    
where:

\( p_1 \) = alloc-space(alloc-size(eva), S, AUTO)

Ref.: 

* is-correct-eva 7-5(9)
* is-free-space 3-17(56)
* mk-gen 7-5(10)
* el-alloc 3-17(55)
* sto-overflow 7-5(11)
* alloc-space 7-11(28)
* alloc-size 7-11(26)
Note: $p_1$ is the pointer value identifying that part of $S$ which is reserved on allocation.

(25) $\text{allocate}(b,\text{eva},\text{type}) =$

- if $\text{is-correct-eva} (\text{eva}) \rightarrow \text{error}$
- $\text{is-free-space}(p_1,S,\text{type}) \rightarrow$

$\text{PASS}: \text{mk-gen}(\text{eva},p_1)$
$S=S: \text{el-alloc}(p_1,S,\text{type})$
$S=\text{ag}: p(AS; <b; \text{mk-gen}(\text{eva},p_1),*>)$

$T \rightarrow \text{stg-overflow}$

where:
$p_1 = \text{alloc-space}(\text{eva},S,\text{type})$

for: $\text{(is-STATIC} \lor \text{is-DUMMY)} \text{ (type)}$

Ref.: $\text{is-correct-eva} 7-5(9)$
$\text{is-free-space} 3-17(56)$
$\text{mk-gen} 7-5(10)$
$\text{el-alloc} 3-17(55)$
$\text{stg-overflow} 7-5(11)$
$\text{alloc-space} 7-11(28)$

Note: $p_1$ is the pointer value identifying that part of $S$ which is reserved on allocation.

7.3 STORAGE MAPPING

This section presents the properties of storage mapping as far as they are implied by PL/I. Storage mapping is defined implicitly by the equations stated in the following. First, the function alloc-size(eva) is assumed, which determines the size of storage to be associated with a variable with evaluated aggregate attributes eva. This function then is related to the storage mapping function map, which determines the mapping of components of a variable onto storage, and to the function alloc-space, which determines that part of storage to be reserved on allocation of a variable.

The elementary storage functions used in the following are defined in 3.4.

Special properties of the storage mapping functions are described in 7.3.2.

(26) alloc-size(eva) =

Note: This implementation defined function determines the size of storage to be associated with a variable with evaluated aggregate attributes eva.

(27) is-size=alloc-size(eva)

Ref.: is-size 3-15(34)

(28) alloc-space(z,vr,\text{type}) =
Note: This implementation defined function gives a pointer value identifying that part of storage vr which is reserved on allocation of a variable requiring a storage part of size z. The argument "type" specifies the type of the allocation (AUTO, STATIC, CTL, BASED, AREA, DUMMY, or BUFFER).

(29) \( \text{is-ptr-val} \cdot \text{alloc-space}(z, vr, \text{type}) \)

Ref.: \( \text{is-ptr-val} \ 3-15(36) \)

(30) \( p \in \text{alloc-state}(vr) \Rightarrow \text{is-indep}(p, \text{alloc-space}(z, vr, \text{type})) \)

Ref.: \( \text{alloc-state} \ 3-16(53) \)
\( \text{is-indep} \ 3-16(43) \)

Note: Storage space identified by the function alloc-space must be independent of all storage identified in the allocation state.

(31) \( \text{size-1} \cdot \text{alloc-space}(z, vr, \text{type}) = z \)

Ref.: \( \text{size-1} \ 3-15(38) \)

Note: This equation states formally that the storage identified by the function alloc-space must have the right size.

(32) \( \text{map}(eva, i) = \)

Note: This implementation defined function gives the pointer value identifying the storage part given to the ith immediate component of a variable with aggregate attributes eva, within the storage given to the entire variable.

(33) \( \text{is-ptr-val} \cdot \text{map}(eva, i) \)

Ref.: \( \text{is-ptr-val} \ 3-15(36) \)

(34) \( (\text{is-array} \lor \text{is-struct})(eva) \land \text{lbd}(eva) \leq i \leq \text{ubd}(eva) \Rightarrow \text{is-applic}(\text{map}(eva, i), \text{alloc-size}(eva)) \land \text{size-1} \cdot \text{map}(eva, i) = \text{alloc-size} \cdot \text{aggr-part}(eva, i) \)

Ref.: \( \text{is-applic} \ 3-15(37) \)
\( \text{size-1} \ 3-15(38) \)
\( \text{aggr-part} \ 8-37(113) \)

Note: This equation relates the function map with the function alloc-size.

(35) \( (\text{is-array} \lor \text{is-struct})(eva) \land \text{lbd}(eva) \leq i \leq \text{ubd}(eva) \land \text{lbd}(eva) \leq j \leq \text{ubd}(eva) \land i \neq j \Rightarrow \text{is-indep}(\text{map}(eva, i), \text{map}(eva, j)) \)

Ref.: \( \text{is-indep} \ 3-16(43) \)
Note: The storage parts given to components of a variable must be independent of one another.

\[(36)\] alloc-state(vr-1) = alloc-state(vr-2) ⇒ alloc-space(eva, vr-1, AREA) = alloc-space(eva, vr-2, AREA)

Ref.: alloc-state 3-16(53)

Note: Storage space reserved on allocation in an area depends only on the size required and on the allocation state of the area.

\[(37)\] lbd(eva) =
  is-array(eva) ⇒ s-lbd(eva)
  is-struct(eva) ⇒ 1

\[(38)\] ubd(eva) =
  is-array(eva) ⇒ s-ubd(eva)
  is-struct(eva) ⇒ length(eva)

7.3.2 SPECIAL PROPERTIES OF THE STORAGE MAPPING FUNCTION

7.3.2.1 unaligned string aggregates

Unaligned string aggregates are mapped in a structure-independent way. This property is expressed by the properties of the function str-part(base, i, j) stated in the following equations. The function str-part is related to the function map.

\[(39)\] str-part(base, i, j) =

Note: This implementation-defined function gives the pointer value identifying the storage part given to the part of a string aggregate whose base is "base" starting from the ith string element, j string elements long, within the storage given to the entire string aggregate.

\[(40)\] 1 ≤ i-1 ≤ j & 1 ≤ j-1 ≤ (j - i-1 + 1) ⇒
  is-applic(str-part(base, i-1, j-1), alloc-size•str-eva(base, j))

Ref.: is-applic 3-15(37)

\[(41)\] size-1•str-part(base, i, j) = alloc-size•str-eva(base, j)

Ref.: size-1 3-15(38)

\[(42)\] 1 ≤ i-1 ≤ i-2 ≤ (i-2 + j-2 - 1) ≤ (i-1 + j-1 - 1) ⇒
  str-part(base, i-2, j-2) =
  (str-part(base, i-2 - i-1 + 1, j-2))•(str-part(base, i-1, j-1))
7.3.2.2 Mapping of picture variables

Picture variables are treated like string variables of corresponding length.

(46) \( \text{is-pic}\_\text{da}(\text{eva}) \Rightarrow \text{alloca-size}(\text{eva}) = \text{alloca-size}(\text{eva}_1) \)

where:
\[
\text{eva}_1 = \mu(\text{eva}; s_{-\text{da}}: \mu_0(s_{-\text{base}}: \text{base}_1, s_{-\text{length}}: \text{length}_1))
\]
\[
\text{base}_1 = (\text{is-bin-pic}\_\text{da}(\text{eva}) \Rightarrow \text{BIT}, \text{CHAR})
\]
\[
\text{length}_1 = \text{str-length}\_\text{da}(\text{eva})
\]

Ref.: str-length 9-41(164)

7.3.2.3 Relation between the declared and the allocated size of areas

The following equation expresses the property that an allocation in an area is possible if it is possible in an area with the same allocation state but smaller size.

(47) \( \text{size}(\text{vr}_1) = \text{alloca-size}(\text{eva}_1) \land \text{size}(\text{vr}_2) = \text{alloca-size}(\text{eva}_2) \land i_2 > i_1 \land \text{alloca-state}(\text{vr}_1) = \text{alloca-state}(\text{vr}_2) \land \text{is-free-space}(p, \text{vr}_1, \text{AREA}) \Rightarrow \text{is-free-space}(p, \text{vr}_2, \text{AREA}) \)

where:
\[
\text{eva}_1 = \mu_0(s_{-\text{da}}: \mu_0(s_{-\text{size}}: i_1), s_{-\text{dens}}: \text{dens})
\]
\[
\text{eva}_2 = \mu_0(s_{-\text{da}}: \mu_0(s_{-\text{size}}: i_2), s_{-\text{dens}}: \text{dens})
\]

Ref.: size 3-15(33)
alloca-state 3-16(53)
is-free-space 3-17(56)

14 7. ALLOCATION, INITIALISATION, AND FREEING OF VARIABLES
7.3.2.4 *Left-to-right equivalence*

The following equation expresses the property that the mapping of a structure component depends on the structure attributes only up to and including that component.

\[ \text{sub-pp-1}(p, \text{set-eq}(\text{eva}, \text{rl}), \text{eva}) = \text{sub-pp-1}(p, \text{set-eq}(\text{eva}, \text{rl}), \text{left-aggr}(\text{eva}, \text{set-eq}(\text{eva}, \text{rl})) \]

Ref.: sub-pp-1 8-37(117)
      set-eq 8-13(31)
      left-aggr 8-14(33)

7.4 *INITIALISATION OF VARIABLES*

Initialisation of variables is accomplished by the instruction \( \text{initialize}(\text{gen}, \text{aggr}) \), where \( \text{gen} \) is the generation of the variable to be initialized, and \( \text{aggr} \) are the aggregate attributes of the variable.

The distinction is made between the initialisation by a call statement, the initialisation by a list of initial expressions, and the special initialisation of label and entry variables. The latter case in concrete text corresponds to the use of subscripted references as prefixes. In abstract text a label or entry constant appears in this case in the initial attribute together with the subscript list identifying the scalar part to be inserted.

\[ \text{initialize}(\text{gen}, \text{aggr}) = \]
\[ \text{is-array}(\text{aggr}) \rightarrow \text{initialize}(\text{gen}, \text{s-elem}(\text{aggr})) \]
\[ \text{is-struct}(\text{aggr}) \rightarrow \]
\[ \text{null} : \]
\[ \text{initialize}(\text{sub-gen}(\text{gen}, \text{rl}), \text{aggr} \cdot \text{s-elem}(\text{i}, \text{aggr})) \mid 1 \leq i \leq \text{length}(\text{aggr}) \]
\[ \text{is-scalar}(\text{aggr}) \rightarrow \text{initialize-1}(\text{gen}, \text{s-init}(\text{aggr}), \text{s-area} \cdot \text{s-da}(\text{aggr})) \]

Where:
\( \text{rl} = \text{rep-conc}(\text{dim} \cdot \text{eva}(\text{gen}), <>, <i>) \)

Ref.: sub-gen 8-36(115)
      dim 8-29(86)

\[ \text{initialize-1}(\text{gen}, \text{init}, \text{area}) = \]
\[ \text{is-call-st}(\text{init}) \& \text{is-0\*pa-opt}(\text{init}) \rightarrow \text{int-call-st}(\text{init}) \]
\[ \text{is-init-elem-list}(\text{init}) \rightarrow \text{initial-assign}(\text{gen-list}(\text{gen}), \text{init}, \text{area}) \]
\[ \text{is-spec-init-elem-list}(\text{init}) \& \]
\[ \text{(is-LABEL \& is-ENTRY) \& s-da \& base-elem \& s-eva}(\text{gen}) \rightarrow \text{special-initialize}(\text{gen}, \text{init}) \]
\[ \text{?} \rightarrow \text{error} \]

7. ALLOCATION, INITIALISATION, AND FREEING OF VARIABLES 15
special-initialize(gen, init) =

is-<> (init) -> null
T ->

special-initialize(gen, tail(init));
convert-assign(gen, op, 0, 0)

where:

\[
\begin{align*}
\text{gen}_1 &= \text{sub-gen}(\text{gen}, \text{s-sl}\cdot \text{head}(\text{init})) \\
\text{op}_1 &= \text{val-cp}(\text{da}_1, \text{s-n}\cdot \text{head}(\text{init})) \\
\text{da}_1 &= (\text{is-entry}(\text{s-n}(\text{init})(\text{AT}))) \rightarrow \text{ENTRY}, \\
&\hspace{1cm} \text{is-label-const}(\text{s-n}(\text{init})(\text{AT})) \rightarrow \text{LABEL}, \\
&\hspace{1cm} T \rightarrow \text{error}
\end{align*}
\]

initial-assign(gen1, init, area) =

\[-\text{is-<> (gen1)} \& \text{is-area}\cdot \text{gen-da}\cdot \text{head}(\text{gen1}) \& (\text{is-<> (init)} \lor \text{is-*head}(\text{init})) \rightarrow
\]

\[\text{initial-assign}(\text{tail}(\text{gen1}), \text{tail-1}(\text{init}), \text{area});
\text{convert-assign}(\text{head}(\text{gen1}), \text{op}, 0, 0);
\text{op: eval-empty}
\]

\[-\text{is-<> (gen1)} \lor \text{is-<> (init)} \rightarrow \text{null}
\]

\[\text{is-init-iter}\cdot \text{head}(\text{init}) \rightarrow
\]

\[\text{initial-assign}(\text{gen1}, \text{init-1});
\text{init-1: pass-rep-conc}(\text{i, s-init}\cdot \text{head}(\text{init}), \text{tail}(\text{init}));
\text{eval-intg-expr}(\text{s-rep}\cdot \text{head}(\text{init}))
\]

\[-\text{is-*head}(\text{init}) \rightarrow \text{initial-assign}(\text{tail}(\text{gen1}), \text{tail}(\text{init}));
\text{is-exp}\cdot \text{head}(\text{init}) \rightarrow
\]

\[\text{initial-assign}(\text{tail}(\text{gen1}), \text{tail}(\text{init}));
\text{convert-assign}(\text{head}(\text{gen1}), \text{op}, \text{area}, \text{area}_1);
\text{op: eval-expr}(\text{head}(\text{init}), \text{da}_1)
\]

\[-\text{is-cp}\cdot \text{head}(\text{init}) \rightarrow
\]

\[\text{initial-assign}(\text{tail}(\text{gen1}), \text{tail}(\text{init}));
\text{assign}(\text{head}(\text{gen1}), \text{head}(\text{op}));
\]

where:

\[\text{area}_1 = \text{area-expr}(\text{head}(\text{init}), \text{AT})
\]
\[\text{da}_1 = (\text{is-entry}\cdot \text{gen-da}\cdot \text{head}(\text{gen1}) \rightarrow \text{ENTRY},
\]
\[T \rightarrow 0)
\]

Ref.: convert-assign 8-8(16)
eval-empty 12-62(151)
eval-intg-expr 8-22(60)

cont'd
Note: Areas are initialized by the `EMPTY` built-in function, if no explicit initial attribute appears. The last alternative accounts for the use of the instruction for initializing static variables, when for checking purposes the initial attribute has already been evaluated and thus contains a list of operands.

(53) \[ \text{gen-da (gen) = z-da\_s-ev (gen)} \]

(54) \[ \text{ref-conc (i, list-1, list-2) =} \]
\[ \{ \text{CONC list-1 \_ list-2} \} \]

### 7.5 The Free Statement

The free statement is interpreted by the instruction `int-free (t)`, where `t` is the text of the statement. The freeings specified in the statement are executed in any order. The distinction is made between the freeing of controlled variables, the freeing of based variables in areas, and the freeing of based variables in main storage. If an attempt is made to free an active event variable, the `ERROR` condition is raised.

(55) \[ \text{int-free-st (t) =} \]
\[ \text{null; \{int-free (s-list (t), i) | 1 \leq i \leq \text{length}(s-list (t))\}} \]

for: `is-free-st (t)`

(56) \[ \text{int-free (t, i) =} \]
\[ \text{is-ctl-ref (s-ref \_ elem (i,t), AT) & is-\_s-area \_ elem (i,t) \rightarrow free (n_0 (DN))} \]
\[ \text{is-level-1 \_ s-ref \_ elem (i,t) & is-based (attr_0) & is-\_s-area \_ elem (i,t) &} \]
\[ \text{is-ref (ptr_0) & is-\_s-da \_ final \_ ret-type \_ aggr \_ ref (ptr_0, AT) \rightarrow} \]
\[ \text{free-based (gen); \quad \text{gen} : \text{eval-ref-gen (s-ref \_ elem (i,t))} \}
\[ \text{is-level-1 \_ s-ref \_ elem (i,t) & is-based (attr_0) & is-ref (ptr_0) &} \]
\[ \text{is-var-ref (area_0, AT) \rightarrow} \]
\[ \text{free-area (gen-1, gen-2); \quad \text{gen-1} : \text{eval-ref-gen (s-ref \_ elem (i,t))},} \]
\[ \text{gen-2 : \text{eval-ref-gen (area_0)} \}
\[ \text{T \rightarrow ERROR} \]
where:

\[ n_0 = s-n \cdot s-ref \cdot s-elem(i,t) \]
\[ attr_0 = n_0(A) \]
\[ ptr_0 = (\neg s-n \cdot s-\text{ptr-ref} \cdot s-elem(i,t) \to s-\text{ptr-ref} \cdot s-elem(i,t), \]
\[ T \to s-\text{ptr}(attr_0) \]
\[ area_0 = (\neg s-n \cdot s-area \cdot s-elem(i,t) \to s-area \cdot s-elem(i,t), \]
\[ \text{is-FULL} = s-n \cdot (ptr_0(A)) \to \text{area} \cdot s-\text{id-list}(ptr_0) \]
\[ \text{elem}(2, \text{head} \cdot s-ap(ptr_0)), \]
\[ T \to area_0 \]
\[ area_1 = \text{area-exr}(ptr_0,AT) \]

for: \( \text{is-free-list(t)} \)

Ref.:
- \( \text{is-ctl-ref} 6-19(50) \)
- \( \text{final-ret-type} 8-28(60) \)
- \( \text{aggr-ref} 8-25(71) \)
- \( \text{eval-ref-gen} 8-28(82) \)
- \( \text{is-var-ref} 8-9(21) \)
- \( \text{area-exr} 8-28(81) \)

Note: If the \( s \)-area component is 0, the area reference is obtained from the area reference declared with the offset qualifier of the based variable, or if the based variable is qualified by a reference to the \text{POINTER} built-in function, from the second argument of that function.

(57) \( \text{is-level-1}(\text{ref}) = \)
\[ \text{length} \cdot s-id-list(\text{ref}) = 1 \& \text{is-<s-ap}(\text{ref}) \& \text{is-<s-sl}(\text{ref}) \]

(58) \( \text{free}(b) = \)
\[ \text{length} \cdot b(AG) > 1 \& \]
\[ (\text{sgen},rl)(\text{gen} = \text{sub-gen}(\text{head} \cdot b(AG),rl) \& \text{is-active-event}(\text{gen},PA)) \to \]
\[ \text{call-cond}(\text{IFREF}) \]
\[ \text{length} \cdot b(AG) > 1 \& \text{is-active}(\text{gen},PA) \to \text{ERROR} \]
\[ \text{length} \cdot b(AG) > 1 \to \]
\[ s-s: \text{free}([s-\text{ap} \cdot \text{head} \cdot b(AG)],S) \]
\[ s-ag: s: \text{tail} \cdot b(AG) \]
\[ \text{is-gen} \cdot \text{head} \cdot b(AG) \to \text{ERROR} \]
\[ T \to \text{null} \]

Ref.:
- \( \text{sub-gen} 8-36(115) \)
- \( \text{is-active-event} 5-5(11) \)
- \( \text{call-cond} 10-18(54) \)
- \( \text{is-active} 5-5(12) \)
- \( \text{free} 3-17(62) \)
- \( \text{is-gen} 3-14(30) \)
(59) \text{free-based}(\text{gen}) = \\
(\exists \text{gen-1}, \text{rl}) \text{ (gen-1 = sub-gen(\text{gen}, \text{rl}) \& is-active-event(\text{gen-1}, \text{\textit{PA}})) } \rightarrow \\
\text{call-cond(\text{ERROR})} \\
is-active(\text{gen}, \text{\textit{PA}}) \rightarrow \text{error} \\
\text{p}_1 \in \text{alloc-state}(\text{\textit{S}}) \& \text{p}_1 \in \text{s-free-set}(\text{\textit{TE}}) \& \text{is-active}(\text{gen}) \rightarrow \\
\text{s-g:el-free}([\text{p}_1], \text{\textit{S}}) \\
\text{s-te:}p(\text{\textit{Ti}}; \langle \text{s-free-set:s-free-set(\text{\textit{TE}}) - \{\text{p}_1}\} \rangle) \\
\text{T} \rightarrow \text{error} \\
\text{where:} \\
\text{p}_1 = \text{s-pp}(\text{gen}) \\
\text{Ref.:} \\
\text{sub-gen 8-36 (115)} \\
is-active-event 5-5 (11) \\
call-cond 10-18 (54) \\
is-active 5-5 (12) \\
alloc-state 3-16 (53) \\
el-free 3-17 (62) \\

(60) \text{free-area}(\text{gen-1}, \text{gen-2}) = \\
(\exists \text{gen}, \text{rl}) \text{ (gen = sub-gen(\text{gen-1}, \text{rl}) \& is-active-event(\text{gen}, \text{\textit{PA}})) } \rightarrow \\
\text{call-cond(\text{ERROR})} \\
is-active(\text{gen-1}, \text{\textit{PA}}) \rightarrow \text{error} \\
is-area:gen-da(\text{gen-2}) \& (\exists \text{o}) (\text{p}_1 = \text{c*p}_1) \& \text{c}_1 \in \text{alloc-state*p}_2(\text{\textit{S}}) \& \\
is-active(\text{gen-1}) \rightarrow \\
\text{s-g:el-ass(el-free([c_1], p_2(\text{\textit{S}})), p_3, \text{\textit{S}})} \\
\text{T} \rightarrow \text{error} \\
\text{where:} \\
\text{p}_1 = \text{s-pp}(\text{gen-1}) \\
\text{p}_2 = \text{s-pp}(\text{gen-2}) \\
\text{c}_1 = (\text{lo}) (\text{p}_1 = \text{c*p}_2) \\
\text{Ref.:} \\
\text{sub-gen 8-36 (115)} \\
is-active-event 5-5 (11) \\
call-cond 10-18 (54) \\
is-active 5-5 (12) \\
gen-da 7-17 (53) \\
alloc-state 3-16 (53) \\
el-ass 3-16 (48) \\
el-free 3-17 (62)
The first section of this chapter presents the mechanism for handling aggregate assignment statements, including assignment statements with the BY NAME option, the syntactic modification of expressions used in the expansion of aggregate assignment statements, and the definition of scalar assignment statements. The second section defines the evaluation of expressions down to the level of data operations, which are described in chapter 9. The third section describes the reference to proper, based, and defined variables.

Metavariables

aggr is-aggr aggregate attributes
aggrl is-aggr-list list of aggregate attributes
ap is-expr-list-list argument part of reference
area is-ref \ is-a optional reference to an area
at is-at attribute directory
byname is-ctt BY NAME option
const is-const constant
da is-da data attribute
descrl is-decr-list \ is-n \ is-G descriptor list generic for selection
evl is-intg-val-list evaluated list of name qualifiers
eva is-eva evaluated aggregate attributes
expr is-expr-1 expression
extent is-extent declared extent
gen is-gen generation of variable
genl is-gen-list generation list
i is-intg-val integer value
id is-id identifier
isub is-isub isub variable
k is-intg-val integer value
list is-list
le is-ref-1-list left-part of assignment statement
mi is-eva mapping information
m is-n unique name
**ABSTRACT SYNTAX AND INTERPRETATION OF PL/I**

30 April 1969

op \(\text{is-op}\) operand

opr \(\text{is-infix-opr} \lor \text{is-prefix-opr}\)

r \(\text{is-ptr-val}\)

p \(\text{is-pp}\)

sl \(\text{is-id-list}\) list of name qualifiers

ref \(\text{is-ref-1}\)

rl \(\text{is-r-list}\) reference list

where \(\text{is-r} = \text{is-intg-val} \lor \text{is-}\)

sl \(\text{is-subscr-expr-list}\)

t \(\text{is-assign-st}\)

s \(\text{is-selector}\)

**Abbreviations**

\[\text{id}_0 = \text{head-}\text{id-list(ref)}\] main identifier of a reference

\[\text{n}_0 = \text{s-n}\text{(Ref)}\] unique name of referenced item

\[\text{attr}_0 = \text{n}_0\text{(AT)}\] attributes of referenced item

\[\text{aggr}_0 = \text{s-aggr}\text{(attr}_0\text{)}\] aggregate attributes of referenced variable

\[\text{aggr}_1 = \text{sub-aggr}\text{(aggr}_0,\text{sl}\text{(ref)},\text{ecl}_1\text{)}\] aggregate attributes of referenced part of variable

\[\text{eva}_0 = ((\text{is-rcp-}\text{var} \lor \text{is-based})\text{(attr}_0\text{)} \rightarrow \text{eva}_2\text{gen}(\text{ref}), \text{is-defined}\text{(attr}_0\text{)} \rightarrow \text{n}_0\text{(FN)})\] evaluated aggregate attributes of referenced variable

\[\text{eva}_1 = \text{sub-aggr}\text{(eva}_0,\text{sl}\text{(ref)},\text{ecl}_1\text{)}\] evaluated aggregate attributes of referenced part of variable

\[\text{ecl}_1 = \text{eval-rl}\text{(tail-}\text{id-list(ref)},\text{aggr}_0\text{)}\] evaluated list of name qualifiers

### 8.1 THE ASSIGNMENT STATEMENT

The assignment statement is interpreted by the instruction \(\text{int-assign-st}(t)\), where \(t\) is the text of the statement. The proper interpretation is preceded by the pre-evaluation of the left-part and the right-part of the statements. The pre-evaluation comprises the evaluation of the pointer qualifiers and of the aggregate attributes of references to based variables, and the fixing of the generations of pointer variables (cf. 8.2.1).

For aggregate assignment statements the pre-evaluation is followed by the syntactic expansion of the text of the statement. The text is expanded iteratively into a sequence of texts of scalar assignment statements, which are
interpreted sequentially. The expansion is governed by the aggregate attributes of the references in the left-part of the statement. The expansion process is described in 8.1.1, the syntactic modification occurring at each expansion step in 8.1.2.

A scalar assignment statement is interpreted by evaluating the generations of the references of the left-part (or pseudo-generations in the case of references to pseudo-variables) in left-to-right order, evaluating the right-part expression, and assigning the resulting operand to the left-part generations in left-to-right order. Before the actual assignment to a generation the operand to be assigned is converted to the attributes of the generation. The interpretation of scalar assignment statements is described in 8.1.3, the evaluation of generations of references to variables in 8.3, the evaluation of expressions in 8.2, and the conversion of operands in 9.

After the execution of all assignments the check condition is raised for the variables in the left-part for which the check-condition is enabled.

(1) \text{int-assign-st}(t) =
\text{call-check-cond}(lp) :
\text{int-assign}(lp, expr, s-bynam(t));
\text{expr-eval}(s-lp(t));
lp:\text{expr-eval-list}(s-lp(t))
for:is-assign-st(t)

Ref.: \text{call-check-cond} 10-21(64)
\text{expr-eval} 8-11(26)

(2) \text{expr-eval-list}(lp) =
\text{is-<>}(lp) \rightarrow \text{PASS}::<>
T ->
\text{nk-list}(ref,ref-list);
\text{ref-list:pre-eval-list}(tail(lp));
\text{ref:pre-eval}(head(lp))

Ref.: \text{expr-eval} 8-11(26)

8.1.1 EXPANSION OF AGGREGATE ASSIGNMENT STATEMENTS

The expansion is governed by the aggregate attributes of the references in the left-part lp of the statement and the BY NAME option bynam, which is either * or $. The aggregate attributes of a left-part reference ref are determined by the function \text{aggr-lg}(ref,at). The effect of the expansion is the sequential execution of scalar assignment statements.

8. ASSIGNMENT STATEMENT, EXPRESSION EVALUATION, REFERENCE TO VARIABLES 3
ABSOLUTE SYNTAX AND INTERPRETATION OF PL/I

30 April 1969

(3) \textbf{int-assign}(lp, expr, byname) =

\[
((31)(\text{is-\text{*}elem(i, lp)} \text{ \& is-*}(expr)) \rightarrow \text{null}
\]

\[\text{length}(lp)\]

\[\text{Et} \quad \text{is-scalar}\text{-aggr-lp}(\text{elem}(i, lp), AT) \rightarrow \text{int-scalar-assign}(lp, expr)\]

\[i = 1\]

\[\text{length}(lp)\]

\[\text{Et} \quad \text{is-array}\text{-aggr-lp}(\text{elem}(i, lp), AT) \rightarrow \]

\[i = 1\]

\textbf{iterate-assign}(lp, expr, byname, eva, i);

\[1: \text{pass-s-ldd}(\text{eva});\]

\[\text{eva: array-eva-expr(head(lp))}\]

\[\text{length}(lp)\]

\[\text{Et} \quad \text{is-struct}\text{-aggr-lp}(\text{elem}(i, lp), AT) \rightarrow\]

\[i = 1\]

\textbf{iterate-assign}(lp, expr, byname, aggr-lp(head(lp), AT), 1)

Ref.: array-eva-expr 8-26(76)

(4) \textbf{aggr-lp}(ref, at) =

\[\text{is-BUILTIN}(\text{attr}_0) \text{ \& is-ref(ref)} \wedge\]

\[\text{id}_0 = \text{sk-id}(	ext{COMP7}, \text{sk-id}(	ext{CFLX})) \rightarrow \text{is-ref(ref)} \wedge\]

\[\text{is-corresp-1}(\text{aggr-ref}(\text{ref}_1, \text{at}), \text{aggr-ref}(\text{ref}_2, \text{at})) \rightarrow\]

\[\text{aggr-ref}(\text{ref}_1, \text{at})\]

\[\text{is-var}(\text{attr}_0) \wedge \text{is-BUILTIN}(\text{attr}_0) \wedge \text{is-\text{*}arg-list\text{*}s-ap(ref)} \rightarrow\]

\[\text{aggr-ref}(\text{ref}, \text{at})\]

\[T \rightarrow \text{error}\]

where:
\[\text{attr}_0 = s-n(\text{ref})(\text{at})\]
\[\text{id}_0 = \text{head-s-id-list}(\text{ref})\]
\[\text{ref}_1 = \text{elem}(1, \text{arg-list}\text{*}s-ap(ref))\]
\[\text{ref}_2 = \text{elem}(2, \text{arg-list}\text{*}s-ap(ref))\]

Ref.: aggr-ref 8-25(71)

Note: If ref is the reference to a built-in function, the aggregate attributes of the first argument of the built-in function are taken. If ref is a reference to the COMPLEX built-in function, the two arguments of the function must be equally structured. The equal structuring is tested by the predicate is-corresp-1. Aggregate attributes of a reference are determined by the function aggr-ref(ref, at).

(5) \text{is-corresp-1}(\text{aggr}_1, \text{aggr}_2) =

\[\text{is-array}(\text{aggr}_1) \rightarrow \text{is-array}(\text{aggr}_2)\]

\[\text{is-struct}(\text{aggr}_1) \rightarrow \text{is-struct}(\text{aggr}_2)\]

\[\text{is-scalar}(\text{aggr}_1) \rightarrow \text{is-scalar}(\text{aggr}_2)\]

4 8. ASSIGNMENT STATEMENT, EXPRESSION EVALUATION, REFERENCE TO VARIABLES
(6) \( \text{arg-list}(ap) = \)

\[ \text{is-<>}(ap) ~\rightarrow~ <> \]
\[ T \rightarrow \text{head}(ap) \]

(7) \( \text{iterate-assign}(lp, expr, byname, eva, i) = \)

\[ i > \text{ubd}(eva) \rightarrow \text{null} \]
\[ T \rightarrow \]

\( \text{iterate-assign}(lp, expr, byname, eva, i + 1); \)
\( \text{int-assign}(lp-1, expr-1, byname); \)
\( \text{is-<>}(expr-1); \)
\( \text{ubd}(expr-1, lp, eva, part_i); \)
\[ 1 \leq j \leq \text{length}(lp) \}
\[ \text{mod}(\text{expr-1}, \text{eva}, \text{part}_1) \]
\[ \{ \text{mod}(\text{expr-1}, \text{eva}, \text{part}_1) \} \]

where:

\[ \text{part}_i = (\text{is-<>}(byname) \land \text{is-struct}(aggr_1) \rightarrow \text{qual-elem}(i, aggr_1), \]
\[ T \rightarrow i \]
\[ \text{aggr}_1 = \text{aggr}-lp(\text{head}(lp), A1) \]

Ref.: \( \text{ubd 7-13(38)} \)

Note: At each step the instruction \text{int-assign} is applied to a modified left-part
and right-part. The modification is accomplished by the instruction \text{mod}.
If the BY NAME option is present and the left-part references are
structures, the instruction \text{mod} is given the qualifying identifier of the
left-most structure reference which corresponds to the step number \( i \). In
all other cases the instruction \text{mod} is given the step number \( i \) as argument.

8.1.2 SYNTACTIC MODIFICATION OF EXPRESSIONS

The instruction \( \text{mod}(expr, eva, i) \) returns the modified expression \( expr \), where \( eva \)
are the controlling evaluated aggregate attributes and \( i \) is the step number \( i \), in
the case of BY NAME structure expansion, a qualifying identifier. In the latter
example, if any of the references in \( expr \) does not have a structure component
selected by the qualifying identifier, the instruction returns *.

(8) \( \text{mod}(expr, eva, i) = \)

\[ \text{is-infix-expr-1}(expr) \rightarrow \]

\( \text{pass-return-mod}(expr-1); \)
\( \text{s-cr-1}(expr-1): \text{mod}(s-op-1(expr), eva, i), \)
\( \text{s-cr-2}(expr-1): \text{mod}(s-op-2(expr), eva, i), \)
\( \text{s-op}(expr-1): \text{pass}(s-op(expr)) \)

\[ \text{is-prefix-expr-1}(expr) \rightarrow \]

\( \text{pass-return-mod}(expr-1); \)
\( \text{s-cr}(expr-1): \text{mod}(s-op(expr), eva, i), \)
\( \text{s-op}(expr-1): \text{pass}(s-op(expr)) \)

\[ \text{is-paren-expr-1}(expr) \rightarrow \]

\( \text{pass-return-mod}(expr-1); \)
\[ \text{s-op}(expr-1): \text{mod}(s-op(expr), eva, i) \]

cont'd
is-ref(expr) → mod-ref(expr, eva, i)

is-ccnst(expr) → PASS: expr

is-isub(expr) → error

for: (is-intg-val v is-id)(i)

Ref.: is-infix-expr-1 8-16(41)
      is-prefix-expr-1 8-16(42)
      is-paren-expr-1 8-16(43)
      is-intg-val 9-3(5)

(9) return-mcd(expr) =

    is-**s-op-1(expr) v is-**s-op-2(expr) v is-**s-op(expr) → 1

T → expr

(10) mod-ref(ref, eva, i) =

    is-BUILTIN(attr0) → gcd-builtins(ref, eva, i)

    is-var(attr0) & is-array(eva) & is-array(eva) & dim(eva1) = dim(eva) &
    s-lbd(eva1) = s-lbd(eva) & s-ubd(eva1) = s-ubd(eva) →

    PASS:μ(ref;<s-sl:insert(s-sl(ref), bintg-ccnst(i))>)

    is-var(attr0) & is-struct(eva) & is-struct(eva) & length(eva1) = length(eva) &
    is-intg-val(i) →

    PASS:μ(ref;<s-id-list:s-id-list(ref)<s-gual-elem(i, agrgr1)>>)

    is-var(attr0) & is-struct(eva) & is-struct(eva) & is-id(i) &
    (3k)(s-gual-elem(k, agrgr1) = i) →

    PASS:μ(ref;<s-id-list:s-id-list(ref)′<i′>>)

    is-var(attr0) & is-struct(eva) & is-struct(eva) & is-id(i) → PASS:*

    (is-var(attr0) & is-struct(eva) & is-array(eva)) v
    (is-var(attr0) & is-scalar(eva) v is-entry(attr0) v is-generic(attr0) v
    is-label-ccnst(attr0) v is-format-ccnst(attr0) v is-file-ccnst(attr0) &
    is-intg-val(i)→

    PASS:ref

T → 0

for: (is-intg-val v is-id)(i)

Ref.:  gcd-builtins 12-1(1)
       dim 8-29(86)
       is-intg-val 9-3(5)

(11) bintg-ccnst(i) =

    μ0(<s-da:BINTG-IDA>, <s-ν;i>)

6 8. ASSIGNMENT STATEMENT, EXPRESSION EVALUATION, REFERENCE TO VARIABLES
(12) \( \text{insert}(\text{list}, \text{const}) = \)
\[
\begin{align*}
\text{is} & \rightarrow (\text{list}) \rightarrow \text{<const>}
\text{is} & \rightarrow \text{<head}(\text{list}) \rightarrow \text{<const>}\text{tail}(\text{list})
\text{T} & \rightarrow \text{<head}(\text{list})\rightarrow \text{insert}(\text{tail}(\text{list}), \text{const})
\end{align*}
\]

8.1.3 INTERPRETATION OF SCALAR ASSIGNMENT STATEMENTS

Scalar assignment statements are interpreted by the instruction
\text{int-scalar-assgn}(\text{lp}, \text{expr})\), where \text{lp} is the left-part of the statement, and \text{expr} is the expression in the right-part. Both \text{lp} and \text{expr} have already been pre-evaluated (cf. 8.2.1). The left-part references are evaluated by the instruction \text{eval-\text{lp}} in left-to-right order. If a left-part reference is the reference to a pseudo-variable, the result of the evaluation is a pseudo-generation (cf. 12.7.1), in all other cases it is a generation. The expression \text{expr} is evaluated by the instruction \text{eval-\text{expr}-1}\), which returns an operand. The operand is converted to the data attributes of the left-part generations and assigned in left-to-right order. If a left-part generation is a pseudo-generation, the assignment is a pseudo-assignment (cf. 12.7.1).

Conversion of operands is done by the instruction
\text{convert}(\text{eva}, \text{op}, \text{area-1}, \text{area-2})\), where \text{eva} are the target attributes and \text{op} is the operand to be converted. \text{area-1} is an optional reference to an area variable for the case that \text{op} is a pointer operand to be converted implicitly to an offset operand, \text{area-2} is an optional reference to an area variable for the case that \text{op} is an offset operand to be converted implicitly to a pointer operand. If no area reference is specified, \text{area-1} and \text{area-2}, respectively, are \&. Implicit conversion of offset or pointer operands is described by the instruction \text{convert-1}\) (cf. 9.5).

(13) \text{int-scalar-assgn}(\text{lp}, \text{expr}) =
\text{convert-assign-list}(\text{genl}, \text{op}, \text{dal}_1, \text{dal}_2):
\text{op} \rightarrow \text{eval-\text{expr}-1}(\text{expr}, \text{da}_1);
\text{genl} \rightarrow \text{eval-\text{lp}-list}(\text{lp})
\]

where:
\[
\begin{align*}
\text{dal}_1 & = \text{length}(\text{lp})
\end{align*}
\]
\[
\begin{align*}
\text{dal}_2 & = \text{s-da-\text{final-type-aggr-expr}}(\text{expr}, \text{AT})
\text{da}_1 & = (\text{is-entry}\rightarrow \text{head}(\text{dal}_1) \rightarrow \text{ENTRY},
\text{T} \rightarrow \&)
\end{align*}
\]

Ref.:
\text{eval-\text{expr}-1} 8-16(25)
\text{final-ret-type} 8-28(86)
\text{aggr-expr} 8-22(62)

Note: The second argument of \text{eval-\text{expr}-1} determines whether an entry operand is expected from the expression evaluation or not.
(14) \textit{convert-assign-list}(gen1,op,dal,da) = \\
\quad \text{is} \rightarrow \text{null} \\
\quad T \rightarrow \\
\quad \text{convert-assign-list}(\text{tail}(gen1), op, \text{tail}(dal), da); \\
\quad \text{convert-assign}(\text{head}(gen1), op, s\text{-area head}(dal), s\text{-area}(da)) \\
\quad \text{for: is-s-gen-list}(gen1) \\

(15) is-g-gen = \\
\quad is-gen \lor is-ps-gen \\
\text{Ref.: } is-gen 3-14(30) \\
\quad is-ps-gen 12-67(169) \\

(16) \textit{convert-assign}(gen,op,area-1,area-2) = \\
\quad is-ps-gen(gen) \rightarrow \text{pseudo-assign}(gen,op) \\
\quad T \rightarrow \\
\quad \text{assign}(gen,op-1); \\
\quad cp-1\text{-convert}(s\text{-eva}(gen), op, area-1, area-2) \\
\quad \text{for: is-g-gen}(gen) \\
\text{Ref.: } is-ps-gen 12-67(169) \\
\quad \text{pseudo-assign} 12-68(172) \\

(17) \textit{convert}(eva,cr,area-1,area-2) = \\
\quad (is-\text{PTR} \lor is\text{-offset}) (s\text{-da}(eva)) \rightarrow \text{ptr\text{-offset-convert}}(eva,cr,area-1,area-2) \\
\quad T \rightarrow \text{convert-1}(eva,op) \\
\text{Ref.: } \text{convert-1} 5-29(119)
(18) \[ \text{eval-offset} \text{convt}(\text{ev}, \text{op}, \text{area-1}, \text{area-2}) = \]
\[\text{is-PTR}\cdot\text{op} \cdot \text{da}(\text{ev}) \land \text{is-PTR}\cdot\text{cp} \cdot \text{da}(\text{op}) \land \text{is-offset}\cdot\text{op} \cdot \text{da}(\text{op}) \implies \]
\[\text{FAIL} \land \]
\[\text{is-PTR}\cdot\text{op} \cdot \text{da}(\text{ev}) \land \text{is-offset}\cdot\text{op} \cdot \text{da}(\text{op}) \land \text{is-var-ref}(\text{area-2}, \text{AT}) \implies \]
\[\text{eval-ptr(} \text{op, gen)}; \text{gen: eval-ref-gen(} \text{area-2)} \]
\[\text{is-offset}\cdot\text{op} \cdot \text{da}(\text{ev}) \land \text{is-PTR}\cdot\text{op} \cdot \text{da}(\text{op}) \land \text{is-var-ref}(\text{area-1}, \text{AT}) \implies \]
\[\text{eval-offset(} \text{op, gen)}; \text{gen: eval-ref-gen(} \text{area-1)} \]
\[T \implies \text{ERROR} \]

Ref.: \[\text{cp} \cdot \text{da} \text{9-9}(37) \]
\[\text{eval-ptr} 12-62(157) \]
\[\text{eval-ref-gen} 8-28(62) \]
\[\text{eval-offset} 12-63(156) \]

(19) \[\text{eval-lr-list(} \text{lp) =} \]
\[\text{is-<>}(\text{lp}) \implies \text{FAIL:<>} \]
\[T \implies \]
\[\text{lk-list(} \text{gen, genl)}; \text{gen: eval-lr-list(} \text{tail(} \text{lp)})); \text{gen: eval-lr(} \text{head(} \text{lp)})); \]

(20) \[\text{eval-lr(} \text{ref) =} \]
\[\text{is-FILLIN} \cdot \text{attr}(\text{ref}) \land \text{is-<>} \cdot \text{s-sl(} \text{ref)} \land \text{is-s-ptr(} \text{ref)} \land \text{is-<>} \cdot \text{tail-1} \cdot \text{s-ap(} \text{ref)} \implies \]
\[\text{eval-ts-lr(} \text{id}, \text{arg-list(} \text{ref)}) \]
\[\text{is-var-ref(} \text{ref, AT) \implies eval-ref-gen-1(} \text{ref)} \]
\[T \implies \text{ERROR} \]

Ref.: \[\text{eval-ts-lr} 12-67(170) \]
\[\text{eval-ref-gen-1} 8-28(63) \]

(21) \[\text{is-var-ref(} \text{ref, at) =} \]
\[\text{is-ref(} \text{ref)} \land \text{is-<>} \cdot \text{s-ap(} \text{ref)} \land \text{is-var(} \text{s-n(} \text{ref)} \cdot \text{(at)})) \]

(22) \[\text{is-var =} \]
\[\text{is-prop-var} \lor \text{is-defined} \lor \text{is-based} \]

8. ASSIGNMENT STATEMENT, EXPRESSION EVALUATION, REFERENCES TO VARIABLES 5
8.2 EVALUATION OF EXPRESSIONS

This section defines the pre-evaluation of expressions, the evaluation of scalar expressions in entry-context and in non-entry-context, some special cases of expression evaluation needed in other parts of the definition, and the evaluation of aggregate attributes of expressions.

The evaluation of aggregate expressions in PL/I is reduced to the sequential evaluation of scalar expressions. For this mechanism see the definition of the assignment statement (8.1), the definition of the evaluation of arguments (6.2.3), and the definition of the get and put statement (11.6).

Scalar expressions are evaluated by the instruction eval-expr(expr,da), where expr is the text of the expression, and the data attribute da specifies the context in which the expression is to be evaluated. The instruction returns an operand. The pre-evaluated expression is evaluated by the instruction eval-entry-expr-1 if da is an entry data attribute, otherwise by the instruction eval-expr-2.

(24) eval-expr(expr,da) =
    eval-expr-1(expr-1,da);
    expr-1:pre-eval(expr)

for: (is-entry v is-0)(da)
The description-list is present in the second argument of eval-entry-sel-1 for the case that expr is the reference to a generic name and generic selection has to be performed (cf. 6.2.5).

8.2.1 EVALUATION OF POINTER QUALIFIERS AND REFER-CPTIONS

Before the expansion and evaluation of an expression, the generations of proper and based variables referred to in the expression are determined. This pre-evaluation implies the evaluation of pointer qualifiers and REFER-cptions of based variables. References to proper and based variables are modified in that the generations of the referenced variables are inserted as s-gen component of the text of the references. The pre-evaluation is performed by the instruction pre-eval(expr), which returns the text of the pre-evaluated expression expr.

6.2.1.1 Pre-evaluation of expressions

Pre-evaluated are those references in an expression which are relevant for the expansion of aggregate expressions, i.e. references in subscript expressions, in arguments of user defined functions, and in non-expanding arguments of built-in functions are ignored for the pre-evaluation.

(25) eval-sel-1(expr, da) =

is-entry(da) =>

case-val-of(entry, e):
  b-val-entry-exp-1(expr, s-descr-list(da))
T => eval-sel-2(expr)

Ref. z: val-ct 9-5(38)

Note: The description-list is present in the second argument of eval-entry-sel-1 for the case that expr is the reference to a generic name and generic selection has to be performed (cf. 6.2.5).

8.2.1 EVALUATION OF POINTER QUALIFIERS AND REFER-CPTIONS

Before the expansion and evaluation of an expression, the generations of proper and based variables referred to in the expression are determined. This pre-evaluation implies the evaluation of pointer qualifiers and REFER-cptions of based variables. References to proper and based variables are modified in that the generations of the referenced variables are inserted as s-gen component of the text of the references. The pre-evaluation is performed by the instruction pre-eval(expr), which returns the text of the pre-evaluated expression expr.

6.2.1.1 Pre-evaluation of expressions

Pre-evaluated are those references in an expression which are relevant for the expansion of aggregate expressions, i.e. references in subscript expressions, in arguments of user defined functions, and in non-expanding arguments of built-in functions are ignored for the pre-evaluation.

(26) pre-eval(expr) =

is-infix-exp(expr) =>

case(expr-1):
  s-op-1(expr-1) : pre-eval(s-op-1(expr)),
  s-op-2(expr-1) : pre-eval(s-op-2(expr)),
  s-expr(expr-1) : pass(s-expr(expr))

is-prefix-exp(expr) =>

case(expr-1):
  s-cf(expr-1) : pre-eval(s-cf(expr)),
  s-op(expr-1) : pass(s-op(expr))

is-paren-exp(expr) =>

case(expr-1):
  s-cf(expr-1) : pre-eval(s-cf(expr)),
  s-op(expr-1) : pass(s-op(expr))

is-ref(expr) => pre-eval-ref(expr)

T => pass:expr
Note: For each reference a new generation is forced from the evaluated aggregate attributes of the base variable. The resulting aggregate attributes of the base variable, which is the evaluation of the aggregate directory, is the base variable. The evaluation of aggregate attributes is the base variable.

Note: For each reference a new generation is forced from the evaluated aggregate attributes of the base variable. The resulting aggregate attributes of the base variable, which is the evaluation of the aggregate directory, is the base variable. The evaluation of aggregate attributes is the base variable.

Note: For each reference a new generation is forced from the evaluated aggregate attributes of the base variable. The resulting aggregate attributes of the base variable, which is the evaluation of the aggregate directory, is the base variable. The evaluation of aggregate attributes is the base variable.

Note: For each reference a new generation is forced from the evaluated aggregate attributes of the base variable. The resulting aggregate attributes of the base variable, which is the evaluation of the aggregate directory, is the base variable. The evaluation of aggregate attributes is the base variable.

Note: For each reference a new generation is forced from the evaluated aggregate attributes of the base variable. The resulting aggregate attributes of the base variable, which is the evaluation of the aggregate directory, is the base variable. The evaluation of aggregate attributes is the base variable.

Note: For each reference a new generation is forced from the evaluated aggregate attributes of the base variable. The resulting aggregate attributes of the base variable, which is the evaluation of the aggregate directory, is the base variable. The evaluation of aggregate attributes is the base variable.

Note: For each reference a new generation is forced from the evaluated aggregate attributes of the base variable. The resulting aggregate attributes of the base variable, which is the evaluation of the aggregate directory, is the base variable. The evaluation of aggregate attributes is the base variable.

Note: For each reference a new generation is forced from the evaluated aggregate attributes of the base variable. The resulting aggregate attributes of the base variable, which is the evaluation of the aggregate directory, is the base variable. The evaluation of aggregate attributes is the base variable.

Note: For each reference a new generation is forced from the evaluated aggregate attributes of the base variable. The resulting aggregate attributes of the base variable, which is the evaluation of the aggregate directory, is the base variable. The evaluation of aggregate attributes is the base variable.

Note: For each reference a new generation is forced from the evaluated aggregate attributes of the base variable. The resulting aggregate attributes of the base variable, which is the evaluation of the aggregate directory, is the base variable. The evaluation of aggregate attributes is the base variable.

Note: For each reference a new generation is forced from the evaluated aggregate attributes of the base variable. The resulting aggregate attributes of the base variable, which is the evaluation of the aggregate directory, is the base variable. The evaluation of aggregate attributes is the base variable.

Note: For each reference a new generation is forced from the evaluated aggregate attributes of the base variable. The resulting aggregate attributes of the base variable, which is the evaluation of the aggregate directory, is the base variable. The evaluation of aggregate attributes is the base variable.

Note: For each reference a new generation is forced from the evaluated aggregate attributes of the base variable. The resulting aggregate attributes of the base variable, which is the evaluation of the aggregate directory, is the base variable. The evaluation of aggregate attributes is the base variable.

Note: For each reference a new generation is forced from the evaluated aggregate attributes of the base variable. The resulting aggregate attributes of the base variable, which is the evaluation of the aggregate directory, is the base variable. The evaluation of aggregate attributes is the base variable.

Note: For each reference a new generation is forced from the evaluated aggregate attributes of the base variable. The resulting aggregate attributes of the base variable, which is the evaluation of the aggregate directory, is the base variable. The evaluation of aggregate attributes is the base variable.

Note: For each reference a new generation is forced from the evaluated aggregate attributes of the base variable. The resulting aggregate attributes of the base variable, which is the evaluation of the aggregate directory, is the base variable. The evaluation of aggregate attributes is the base variable.

Note: For each reference a new generation is forced from the evaluated aggregate attributes of the base variable. The resulting aggregate attributes of the base variable, which is the evaluation of the aggregate directory, is the base variable. The evaluation of aggregate attributes is the base variable.

Note: For each reference a new generation is forced from the evaluated aggregate attributes of the base variable. The resulting aggregate attributes of the base variable, which is the evaluation of the aggregate directory, is the base variable. The evaluation of aggregate attributes is the base variable.
The evaluation of aggregate attributes of based variables, which implies the evaluation of contained \( \text{BIF} \)-options, is performed by the instruction
\[
\text{eval-based-agg}(\text{agg}, \text{f}, \text{eql})\]
where \( \text{agg} \) are the aggregate attributes to be evaluated, \( \text{f} \) is the pointer value resulting from the evaluation of the pointer qualifier of the based reference, \( \text{eql} \) is the evaluated name qualification list of the reference. The attributes are evaluated only up to the referenced point which is determined by \( \text{eql} \). \( \text{BIF} \)-options are evaluated in left-to-right order as long as the order is relevant, otherwise in any order.

(30) \[
\text{eval-based-agg}(\text{agg}, \text{f}, \text{eql}) = \\
\text{eval-based-agg-1}(\text{agg}, \text{f}, \text{set-eql}(\text{agg}, \text{eql}))
\]

Note: \( \text{set-eql}(\text{agg}, \text{eql}) \) gives the reference-list identifying the right-most structure component up to which \( \text{agg} \) is to be evaluated.

(31) \[
\text{set-eql}(\text{agg}, \text{eql}) = \\
is-<>(\text{eql}) \rightarrow <>
\]
\[
is-struct(\text{agg}) \rightarrow \text{head}(\text{eql})^\ast \text{set-eql}(\text{agg}\text{elem} \text{head}(\text{eql}), \text{agg}), \text{tail}(\text{eql}))
\]
T \( \rightarrow <> 
\)

(32) \[
\text{eval-based-agg-1}(\text{agg}, \text{f}, \text{eql}) = \\
(is-array v is-scalar)(\text{agg-eql}_2) \& is-1-list(\text{eql}) \rightarrow \\
\text{eval-eval-agg-1}(\text{agg-eql}_1)
\]
\[
(is-array v is-scalar)(\text{agg-eql}_3) \rightarrow \\
\text{eval-construct-ri}(\text{mi-1}, \text{mi-2}, \text{eql}); \\
\text{mi-2}: \text{eval-refer-options}(\text{agg-eql}_2, \text{gen}, \text{agg}); \\
\text{gen}: \text{eval-construct-ri}(\text{mi-1}, \text{p}); \\
\text{mi-1}: \text{eval-based-agg-1}(\text{agg-eql}_1, \text{f}, \text{set-eql}(\text{eql}))
\]
\[
is-struct(\text{agg-eql}_2) \rightarrow \text{eval-based-agg-1}(\text{agg}, \text{f}, \text{eql}^\ast \langle \text{length}(\text{agg-eql}_2) \rangle)
\]

where:
\[
\text{agg-eql}_0 = \text{left-agg}(\text{agg}, \text{eql})
\]
\[
\text{agg-eql}_1 = \text{left-agg}(\text{agg}, \text{reset-eql}(\text{eql}))
\]
\[
\text{agg-eql}_2 = \text{set-agg}(\text{agg}, <> , \text{eql})
\]
Ref.: eval-aggr-1 6-9(25)
     sk-gen 7-5(10)

Note: aggr-eql₀ are the aggregate attributes up to and including the referenced
part, aggr-eql₁ are the aggregate attributes up to but excluding the
referenced part, aggr-eql₂ is the referenced part of the aggregate
attributes. The function eval-aggr-1 evaluates aggregate attributes with
integer constant extents, containing no REFER-crtions. The instruction
eval-refer-crtions(aggr-eql₂,gen,aggr) evaluates all REFER-crtions in
aggr-eql₂ in any order, using gen as the generation of the based variable.
The function construct-mi(mi-1,mi-2,eql) constructs evaluated attributes by
inserting mi-2 in mi-1 at the point determined by eql.

\[(33) \quad \text{left-aggr}(aggr,eql) = \]
\[\text{is-<>}(eql) \rightarrow aggr\]
\[\text{I} \rightarrow\]
\[\begin{align*}
\text{head}(eql) & : 1 \\
\text{LIST} & \text{elem(i,aggr)}: i \downarrow \text{<}< \\
\mu_{0} & \text{(<s-aggr:left-aggr(eql)(s-aggr.elem(head(eql),aggr),tail(eql)))>} \]
\end{align*}\]

for:is-struct(aggr)

\[(34) \quad \text{construct-mi}(mi-1,mi-2,eql) = \]
\[\text{is-<>}(\text{tail}(eql)) \rightarrow \mu(mi-1;\text{<s-aggr.elem(i₁):mi-2}>)\]
\[\text{I} \rightarrow\]
\[\begin{align*}
\mu & (mi-1;\text{<s-aggr.elem(i₁):construct-mi}(s-aggr.elem(i₁,mi-1),mi-2,tail(eql)))> \]
\end{align*}\]

where:
\[i₁ = \text{head}(eql)\]

for:is-struct(mi-1)

\[(35) \quad \text{reset-eql}(eql) = \]
\[\text{last}(eql) > 1 \rightarrow \text{first}(eql) \text{<last}(eql) - 1\]
\[\text{I} \rightarrow \text{reset-eql}\text{first}(eql)\]
(36) \texttt{eval-refer-options} (aggr-1, gen, aggr) =
\begin{align*}
is-array (aggr-1) &\rightarrow \\
\text{\texttt{pass} (mi) ;} &\quad \text{\texttt{s-lbd} (mi) := eval-extent (s-lbd (aggr-1), gen, aggr),} \\
&\text{\texttt{s-ubd} (mi) := eval-extent (s-ubd (aggr-1), gen, aggr),} \\
&\text{\texttt{s-elem} (mi) := eval-refer-options (s-elem (aggr-1), gen, aggr).}
\end{align*}
\begin{align*}
is-struct (aggr-1) &\rightarrow \\
\text{\texttt{pass} (mi) ;} &\quad \{\text{\texttt{s-aggr*} (elem (i)) (mi) := eval-refer-options (s-aggr*elem (i, aggr-1), gen, aggr) \} } \\
&\quad 1 \leq i \leq \text{length} (aggr-1)\}
\end{align*}
\begin{align*}
is-string \ast da (aggr-1) &\rightarrow \\
\text{\texttt{pass} (mi) ;} &\quad \text{\texttt{s-lengths-da} (mi) := eval-extent (s-lengths-da (aggr-1), gen, aggr) ;} \\
&\quad \text{\texttt{ni := pass (6 (aggr-1); s-init))}}
\end{align*}
\begin{align*}
is-area \ast da (aggr-1) &\rightarrow \\
\text{\texttt{pass} (mi) ;} &\quad \text{\texttt{s-sizes-da} (mi) := eval-extent (s-sizes-da (aggr-1), gen, aggr) ;} \\
&\quad \text{\texttt{ni := pass (6 (aggr-1); s-init))}}
\end{align*}
\begin{align*}
T &\rightarrow \text{\texttt{pass \mu (s-da; eval-da-1 \ast da (aggr-1) >, <s-dens; s-dens (aggr-1) >)}}
\end{align*}

Ref.: \texttt{eval-da-1 6-10 (26)}

(37) \texttt{eval-extent} (extent, gen, aggr) =
\begin{align*}
is-refer (extent) &\rightarrow \\
\text{\texttt{pass-de-val} (cf-1) ;} &\quad \text{\texttt{cf-1 := convert-1} (aggr-\text{scalar} (\text{BINTG-128}), cf) ;} \\
&\quad \text{\texttt{cf := gen-1 \text{\texttt{cf}}} (gen-1, cf)) ;} \\
&\quad \text{\texttt{gen-1 := pass-sub-gen} (gen, eval-\text{cf} (s-refer (extent), aggr))}
\end{align*}
\begin{align*}
T &\rightarrow \text{\texttt{pass-int-val} (extent)}
\end{align*}

Ref.: \texttt{cf-val 9-9 (36)}
\begin{align*}
\text{\texttt{convert-1 9-28 (119)} &}
\end{align*}
\begin{align*}
\text{\texttt{eval-\text{cf} 8-36 (274)} &}
\end{align*}
\begin{align*}
\text{\texttt{int-val 6-16 (27)} &}
\end{align*}

\textbf{Note:} The attributes \texttt{aggr} are used to interpret the name-qualification list contained in the \texttt{BFI} option.

8.2.1.3 \textbf{Evaluation of pointer qualifiers}

The pointer qualifier of a based reference \texttt{ref} is evaluated by the instructions \texttt{eval-ptr-ref} (ref), which returns the resulting operand. If no explicit pointer qualifier is present, the pointer reference contained in the declaration of the based variable is evaluated. If the pointer qualifier is the reference to an offset variable, the resulting offset operand is converted to a pointer operand by the instruction \texttt{ptr-copy} (ptr, area). \texttt{area} is the reference to an area variable if contained in the declaration of the offset variable, \texttt{&} otherwise.
(38) \[
\text{eval-*ptr-ref}(\text{ref}) = \\
\text{if is-*s.ptr}(\text{ref}) \rightarrow \\
\text{ptr-cmp}(\text{cp}, \text{area}_1); \\
\text{cp}: \text{eval-ref}(\text{s.ptr}(\text{ref}))
\]

\[
\text{if is-*s.ptr}(\text{attr}_0) \rightarrow \\
\text{ptr-cmp}(\text{cp}, \text{area}_2); \\
\text{cp}: \text{eval-ref}(\text{s.ptr}(\text{attr}_0))
\]

\[T \rightarrow \text{error} \]

where:

\[
\text{area}_1 = \text{area-expr}(\text{s.ptr}(\text{ref}), \text{AT})
\]

\[
\text{area}_2 = \text{area-expr}(\text{s.ptr}(\text{attr}_0), \text{AT})
\]

(39) \[
\text{ptr-cmp}(\text{cp}, \text{area}) = \\
\text{is-ftr*cp-da}(\text{cp}) \rightarrow \text{PASS}\text{op}
\]

\[
\text{is-cifset*cp-da}(\text{cp}) \& \text{is-var-ref}(\text{area}, \text{AT}) \rightarrow \\
\text{eval-*ptr}(\text{cp}, \text{gen}); \\
\text{gen}: \text{eval-ref-gen}(\text{area})
\]

\[T \rightarrow \text{error} \]

Ref.: \[
\text{cp-da} 5-9(37) \\
\text{is-var-ref} 8-9(27) \\
\text{eval-*ptr} 12-63(157) \\
\text{eval-ref-gen} 8-28(82)
\]

6.2.14 Abstract syntax of pre-evaluated expressions

The pre-evaluation of expressions changes the abstract syntax of the expressions, since references to variables may contain in the component \text{s-gen} the current generation of the variable. The abstract syntax of pre-evaluated expressions is given by the predicate \text{is-expr-1}.

(40) \text{is-expr-1} =

\[\text{is-infix-expr-1} \lor \text{is-prefix-expr-1} \lor \text{is-paren-expr-1} \lor \text{is-ref-1} \lor \text{is-const} \lor \text{is-isub} \lor \text{is-isub-val}\]

(41) \text{is-infix-expr-1} =

\[<\text{s-cpr}: \text{is-infix-cpr}>, \text{<s-cp-1:is-expr-1>}, \text{<s-cp-2:is-expr-1>}\]

(42) \text{is-prefix-expr-1} =

\[<\text{s-cpr}: \text{is-prefix-cpr}>, \text{<s-cp:is-expr-1>}\]

(43) \text{is-paren-expr-1} =

\[<\text{s-cp:is-expr-1}>\]
8.2.2 EVALUATION OF EXPRESSIONS IN ENTRY-CONTEXT

Scalar expressions in a context which expects an entry operand are evaluated by the instruction \( \text{eval-entry-expr}(\text{ref}, \text{descr}) \), where \( \text{ref} \) is the text of the expression and \( \text{descr} \), if present, is a list of descriptors for determining the generic selection for the case that \( \text{ref} \) is a generic reference. The instruction returns an operand. Expressions in entry-context are right-part expressions of assignment statements where the left-part references are references to entry variables, or argument expressions with corresponding entry parameters.

\[
\text{eval-entry-expr}(\text{ref}, \text{descr}) = \\
\text{eval-entry-expr-1}(\text{expr-1}, \text{descr}); \\
\text{expr-1:is-eval}(\text{ref})
\]

Note: The pre-evaluation of expressions by the instruction \( \text{pre-eval} \) is described in 8.2.1.

Note: The component \( s-v \) contains the actual value of the isub-variable (cf. 8.3.3.1).
\[(48)\]
\[
eval-entry-expr-1(\text{ref}, \text{descrl}) =
\]
\[
is-taren-expr(\text{ref}) \rightarrow \text{eval-entry-expr-1}(\text{s-cp}(\text{ref}), \text{descrl})
\]
\[
is-ref(\text{ref}) \rightarrow \text{error}
\]
\[
is-var(\text{attr}_0) \& is-entry\_s-da(\text{aggr}_1) \rightarrow
\]
\[
\text{eval-entry}(\text{n}, \text{s-da}(\text{aggr}_1), \text{s-ap}(\text{ref}), \text{ref});
\]
\[
\text{cp}\_\text{pass-cp-val}(\text{cp});
\]
\[
\text{cp}\_\text{eval-var-ref}(\text{ref})
\]
\[
is-entry(\text{attr}_0) \& is-<>\_s-sl(\text{ref}) \& is-\_s-ptr(\text{ref}) \rightarrow
\]
\[
\text{eval-entry}(\text{n}, \text{attr}_0, \text{s-ap}(\text{ref}), \text{ref})
\]
\[
is-generic(\text{attr}_0) \& is-<>\_s-sl(\text{ref}) \& is-\_s-ptr(\text{ref}) \rightarrow
\]
\[
\text{eval-entry-expr}(\mu(\text{ref-g}_0; <\text{s-ap}:\text{s-ap}(\text{ref-g}_0)\_\text{s-ap}(\text{ref}>)_0), \emptyset)
\]
\[
is-BUILTIN(\text{attr}_0) \& is-flat-generic-built-in(\text{id}_0) \& is-<>\_s-ap(\text{ref}) \&
\]
\[
is-<>\_s-sl(\text{ref}) \& is-\_s-ptr(\text{ref}) \rightarrow
\]
\[
\text{builtin-sel}(\text{id}_0, \text{descrl}, \text{n});
\]
\[
\text{true-page}
\]
\[
T \rightarrow \text{error}
\]

\text{where:}
\[
\text{ref-g}_0 = \text{generic-sel}(\text{attr}_0, \text{descrl}_1),
\]
\[
\text{descrl}_1 = (\text{is-<>}\_s-ap(\text{ref}) \rightarrow \text{descrl},
\]
\[
1 \rightarrow \text{attr-expr-list}(\text{head}\_s-ap(\text{ref}), \text{AI})
\]
\[
\text{fcr: is-expr-1}(\text{ref})
\]

\text{Ref.:}
\[
is-var 6-9(22)
\]
\[
cp\_val 9-9(36)
\]
\[
is-flat-generic-built-in 12-11(26)
\]
\[
\text{builtin-sel} 12-11(27)
\]
\[
\text{assign} 3-15(20)
\]
\[
gegeneric-sel 6-31(77)
\]
\[
\text{attr-expr-list} 6-35(46)
\]

\text{Note:} Generic selection is described in 6.2.5.

\[(49)\]
\[
eval-entry(n, \text{da}, \text{at}, \text{ref}) =
\]
\[
is-<>(\text{ap}) \rightarrow \text{pass-n}
\]
\[
is-entry(\text{da}) \rightarrow
\]
\[
\text{eval-entry}(n-1, \text{s-da}\_s-ret-type(\text{da}), \text{tail(ap)}, \emptyset);
\]
\[
\text{r-1: pass-cp-val}(\text{cp});
\]
\[
\text{cp}\_\text{eval-function}(n, \text{head(ap), da, ref})
\]
\[
T \rightarrow \text{error}
\]

\text{Ref.:}
\[
cp\_val 9-9(36)
\]
\[
\text{eval-function} 6-13(35)
\]
Note: An entry value is called as a function as long as the argument part of the reference is not exhausted.

8.2.3 EVALUATION OF EXPRESSIONS IN NON-ENTRY CONTEXT

Scalar expressions in a non-entry context, i.e. when not an entry operand is expected as the result, are evaluated by the instruction `eval-expr-2(expr)`, where `expr` is the pre-evaluated text of the expression. The instruction returns an operand.

The definition of data operations performed in expression evaluation is given in chapter 9, the definition of procedure calls, performed on evaluating functions, is 6.2.

(50) \[ \text{eval-expr-2( } expr \text{ ) } = \]

\[ \text{is-infix-expr-1( } expr \text{ ) } \& \]

\[ \{\text{is-PTR( } da-expr_1 \text{ ) } \&\text{ is-offset( } da-expr_2 \text{ ) } \&\text{ is-PTR( } da-expr_3 \text{ ) } \} \rightarrow \]

\[ \text{eval-infix-expr( } cp-1, op-1, \text{ infix-opr( } expr \text{ ) } ) ; \]

\[ cp-1: \text{ptr-ccnv( } op-3, \text{ area}_1 \text{ ) } ; \]

\[ cp-3: \text{eval-expr-2( } s-op-1( \text{ } expr \text{ ) } ) ; \]

\[ cp-2: \text{ptr-ccnv( } op-4, \text{ area}_2 \text{ ) } ; \]

\[ cp-4: \text{eval-expr-2( } s-op-2( \text{ } expr \text{ ) } ) \}

\[ \text{is-infix-expr-1( } expr \text{ ) } \rightarrow \]

\[ \text{eval-infix-expr( } cp-1, op-2, \text{ infix-opr( } expr \text{ ) } ) ; \]

\[ cp-1: \text{eval-expr-2( } s-op-1( \text{ } expr \text{ ) } ) , \]

\[ cp-2: \text{eval-expr-2( } s-op-2( \text{ } expr \text{ ) } ) \}

\[ \text{is-prefix-expr-1( } expr \text{ ) } \rightarrow \]

\[ \text{eval-prefix-expr( } cf, s-off( \text{ } expr \text{ ) } ) ; \]

\[ cp-1: \text{eval-expr-2( } s-op( \text{ } expr \text{ ) } ) \}

\[ \text{is-ccnst( } expr \text{ ) } \rightarrow \text{ PASS:eval-ccnst( } expr \text{ ) } \]

\[ \text{is-isub( } expr \text{ ) } \rightarrow \text{ PASS:isub-op( } expr \text{ ) } \]

where:

\[ \text{area}_1 = \text{area-expr( } s-op-1( \text{ } expr \text{ ) } , \text{ AT} \) \]

\[ \text{area}_2 = \text{area-expr( } s-op-2( \text{ } expr \text{ ) } , \text{ AT} \) \]

Ref.: `eval-infix-expr 9-13(64)`

\[ \text{eval-prefix-expr 9-26(105)} \]

Note: Before infix-operations between pointer and offset operands, the offset operand is implicitly converted to a pointer operand.
(51) \( \text{infix-cfr}(\text{expr}) = \)
\( \text{is-c-expconnt}(\text{expr}) \rightarrow \text{C-EXPR} \)
\( T \rightarrow \text{s-cfr}(\text{expr}) \)
for: \( \text{infix-exp-1}(\text{expr}) \)
Ref.: \( \text{is-c-expconnt} 9-26(105) \)

Note: The special operator C-EXPR is inserted if the rules for integer constant exponentiation apply.

(52) \( \text{eval-ccnst}(\text{ccnst}) = \)
\( \text{is-cCorrect-eda}(\text{eda}(\text{ccnst})) \land \text{is-type-match}(\text{eda}(\text{ccnst}), \text{s-v}(\text{ccnst})) \land \)
\( \text{is-prec-arith}(\text{eda}(\text{ccnst})) \land \text{is-size-cond}(\text{eda}(\text{ccnst}), \text{s-v}(\text{ccnst})) \land \)
\( \text{is-string-eda}(\text{eda}(\text{ccnst})) \land \text{is-str-size-cond}(\text{eda}(\text{ccnst}), \text{s-v}(\text{ccnst})) \)
\( \rightarrow \text{val-cf}(\text{eda}(\text{ccnst}), \text{s-v}(\text{ccnst})) \)
\( T \rightarrow \text{error} \)
Ref.: \( \text{is-cCorrect-eda} 9-2(1) \)
\( \text{is-type-match} 9-4(13) \)
\( \text{is-size-cond} 9-8(28) \)
\( \text{is-str-size-cond} 9-8(29) \)
\( \text{val-cf} 9-9(38) \)

(53) \( \text{isub-cf}(\text{isub}) = \)
\( \sim \text{is-c-v}(\text{isub}) \rightarrow \text{val-cf}(\text{BINTG-IFA}, \text{s-v}(\text{isub})) \)
\( T \rightarrow \text{error} \)
Ref.: \( \text{val-cf} 9-9(38) \)

(54) \( \text{eval-ref}(\text{ref}) = \)
\( \text{eval-ref-1}(\text{ref-1}); \)
\( \text{ref-1:fix-eval}(\text{ref}) \)

2) B. ASSIGNMENT STATEMENT, EXPRESSION EVALUATION, REFERENCE TO VARIABLES
(55) \texttt{eval-ref-1(ref) =}

\begin{align*}
& \text{is-var} \left( \text{attr}_0 \right) \wedge \text{is-entry} \left( \text{da} \left( \text{aggr}_1 \right) \right) \rightarrow \\
& \text{eval-entry-ref} \left( n, \text{da} \left( \text{aggr}_1 \right), \text{s-ap} \left( \text{ref} \right), \text{ref} \right); \\
& \text{cp} : \text{eval-var-ref} \left( \text{ref} \right) \\
& \text{is-var-ref} \left( \text{ref}, \text{AT} \right) \rightarrow \text{eval-var-ref} \left( \text{ref} \right) \\
& \text{is-entry} \left( \text{attr}_0 \right) \wedge \text{is-} < > * \text{s-sl} \left( \text{ref} \right) \wedge \text{is-} \text{eq} * \text{s-ptr} \left( \text{ref} \right) \rightarrow \\
& \text{eval-entry-ref} \left( n_0, \text{attr}_0, \text{s-ap} \left( \text{ref} \right), \text{ref} \right) \\
& \text{is-generic} \left( \text{attr}_0 \right) \wedge \text{is-} < > * \text{s-sl} \left( \text{ref} \right) \wedge \text{is-} \text{eq} * \text{s-ptr} \left( \text{ref} \right) \rightarrow \\
& \text{eval-ref} \left( \mu \left( \text{ref-g} \rightarrow \text{s-ap} \left( \text{ref-g} \right) \rightarrow \text{s-ap} \left( \text{ref} \right) \right) \right) \\
& \text{is-built-in} \left( \text{attr}_0 \right) \wedge \text{is-} < > * \text{s-sl} \left( \text{ref} \right) \wedge \text{is-} \text{eq} * \text{s-ptr} \left( \text{ref} \right) \wedge \\
& \text{is-} < > * \text{tail-1} * \text{s-ap} \left( \text{ref} \right) \rightarrow \\
& \text{eval-built-in} \left( \text{id}_0, \text{arg-list} \left( \text{ref} \right) \right) \\
& \left( \text{is-label-const} \wedge \text{is-format-const} \wedge \text{is-file-const} \left( \text{attr}_0 \right) \wedge \text{is-} < > * \text{s-sl} \left( \text{ref} \right) \wedge \\
& \text{is-} < > * \text{s-ap} \left( \text{ref} \right) \wedge \text{is-} \text{eq} * \text{s-ptr} \left( \text{ref} \right) \rightarrow \\
& \text{FAIL} : \text{val-cp} \left( \text{aggr-scalar} \left( \text{da}_0, \text{ref} \right) \right) \\
& \text{T} \rightarrow \text{error} \\
\end{align*}

where:
\begin{align*}
\text{ref-g}_0 &= \text{generic-sel} \left( \text{attr}_0, \text{descr}_1 \right) \\
\text{descr}_1 &= \text{attr-exr-list} \left( \text{arg-list} * \text{s-ap} \left( \text{ref} \right), \text{AT} \right) \\
\text{da}_0 &= \left( \text{is-label-const} \wedge \text{is-format-const} \left( \text{attr}_0 \right) \rightarrow \text{LABEL} \\
& \text{is-file-const} \left( \text{attr}_0 \right) \rightarrow \text{FILE} \right) \\
\end{align*}

Ref.: \text{is-var 8-9}(22) \\
\text{cp-val 9-9}(36) \\
\text{is-var-ref 9-9}(21) \\
\text{eval-built-in 12-3}(3) \\
\text{arg-list 8-4}(6) \\
\text{val-cp 9-9}(38) \\
\text{generic-sel 6-31}(77) \\
\text{attr-exr-list 6-35}(96)

Note: Generic selection is described in 6.2.6.

(56) \texttt{eval-entry-ref(n,da,ap,ref) =}

\begin{align*}
& \text{eval-entry-ref-1(cp,s-da*g-ret-type(d),tail-1(ap),0);} \\
& \text{cp} : \text{eval-function} \left( n, \text{da}, \text{arg-list} \left( \text{ap} \right), \text{ref} \right) \\
\end{align*}

Ref.: \text{eval-function 6-13}(25) \\
\text{arg-list 8-4}(6)
\begin{itemize}
\item \textbf{(57) eval-entry-ref-1 (cf, da, ap) =}
\item \hspace{1em} is-ENTRY*cf-da(op) -> \texttt{eval-entry-ref} (op-val (op), da, ap, 0)
\item \hspace{1em} is-<> (ap) -> \texttt{PASS:op}
\item \hspace{1em} T -> \texttt{error}
\end{itemize}

Ref.: \texttt{cf-da 9-9(37)}
\texttt{cf-val 9-9(36)}

\textbf{Note:} An entry value is always called as a function. If the argument part is exhausted, the function is given the empty argument list.

\begin{itemize}
\item \textbf{(58) eval-var-ref (ref) =}
\item \hspace{1em} gen-op (gen);
\item \hspace{1em} gen = \texttt{eval-ref-gen-1 (ref)}
\end{itemize}

Ref.: \texttt{eval-ref-gen-1 8-28(83)}

\begin{itemize}
\item \textbf{(59) gen-cf (gen) =}
\item \hspace{1em} is-scalar-s-eva (gen) -> \texttt{PASS:mu0 (<s-eva:s-eva (gen)>, <s-vr:s-pp (gen) (g)>)}
\item \hspace{1em} T -> \texttt{error}
\end{itemize}

\section*{8.2.4 SPECIAL CASES OF EXPRESSION EVALUATION}

\begin{itemize}
\item \textbf{(60) eval-insts-expr (expr) =}
\item \hspace{1em} \texttt{(is-a v is-0) (expr) -> PASS:expr}
\item \hspace{1em} \texttt{is-expr (expr) ->}
\item \hspace{2em} \texttt{PASS:cr-val (op)};
\item \hspace{2em} \texttt{cr:convert-1 (qopr-scalar (BINTC-ELA), op-1)};
\item \hspace{2em} \texttt{cr-1:eval-expr (expr, 0)}
\item \hspace{1em} T -> \texttt{error}
\item \hspace{1em} \texttt{for: (is-expr v is-refer v is-a v is-s) (expr)}
\end{itemize}

Ref.: \texttt{cp-val 9-9(36)}
\texttt{convext-1 9-29(119)}

\textbf{Note:} The instruction is used when the value of an expression is to be converted to an integer value.
8.2.5 Attributes of Expressions

8.2.5.1 Aggregate Attributes of Expressions

The function \texttt{aggr-expr(expr,at)} returns the aggregate attributes of the result of the expression \texttt{expr}, given the attribute directory \texttt{at}. The function is used in all places where the attributes of the result of an expression must be known before the actual evaluation of the expression. The attributes of operands resulting from infix and prefix operations are defined in chapter 9.

(61) \[
\texttt{aggr-c-expr(expr,at)} = \\
(\texttt{is-c-exp(expr) \rightarrow aggr-c-exp(aggr-expr(s-op-1(expr),at),s-cp-2(expr))})
\]

\texttt{is-infix-expr-1(expr) \rightarrow \\
aggr-infix-expr(aggr-expr(s-op-1(expr),at),aggr-expr(s-cp-2(expr),at), s-cp expr)}

\texttt{is-prefix-expr-1(expr) \rightarrow \\
aggr-prefix-expr(aggr-expr(s-cp(expr),st),s-op(expr))}

\texttt{is-paren-expr-1(expr) \rightarrow aggr-expr(s-op(expr),at)}

\texttt{is-ref-1(expr) \rightarrow aggr-ref(expr,at)}

\texttt{is-struct(expr) \rightarrow aggr-scalar*da(expr)}

\texttt{is-iasub(expr) \rightarrow error}

Ref.: \texttt{is-c-exp 5-25}\texttt{(105)}

(62) \[
\texttt{aggr-c-exp(aggr,ccnst)} = \\
\texttt{is-array(aggr) \rightarrow u\langle aggr: :s-cmap:aggr-c-exp(s-cmap(s-aggr,ccnst)) \rangle)}
\]

\[
\texttt{is-struct(aggr) \rightarrow \textit{LIST}}_{i=1} \textit{\langle e-aggr:aggr-c-exp(s-aggr-s-elm(i,aggr),ccnst) \rangle)}
\]

\texttt{is-scalar(aggr) \rightarrow aggr-scalar*da-c-op(deltlenst*da\langle aggr,ccnst \rangle)}

Ref.: \texttt{eda-c-op 9-25}\texttt{(106)}
\begin{align*}
(64) & \quad \text{aggr-scalar}(\text{da}) = \\
& \quad \mu_0(\langle s-\text{da}:\text{da}\rangle, \langle s-\text{dens}:\text{default-dens}(\text{da})\rangle) \\
& \quad \text{for: } (\text{is-\text{da}} \land \text{is-ed}\text{a})(\text{da}) \\
(65) & \quad \text{default-dens}(\text{da}) = \\
& \quad (\text{is-string} \lor \text{is-ric})(\text{da}) \rightarrow \text{UNAL} \\
& \quad T \rightarrow \text{AI} \\
(66) & \quad \text{aggr-infix-expr}(\text{aggr-1}, \text{aggr-2}, \text{opr}) = \\
& \quad \text{is-array}(\text{aggr-1}) \land \text{is-array}(\text{aggr-2}) \land \dim(\text{aggr-1}) = \dim(\text{aggr-2}) \rightarrow \\
& \quad \mu(\text{aggr-1}; \langle s-\text{elem}:\text{aggr-infix-expr}(s-\text{elem}(\text{aggr-1}), s-\text{elem}(\text{aggr-2}), \text{opr})\rangle) \\
& \quad \text{is-array}(\text{aggr-1}) \land \neg\text{is-array}(\text{aggr-2}) \rightarrow \\
& \quad \mu(\text{aggr-1}; \langle s-\text{elem}:\text{aggr-infix-expr}(s-\text{elem}(\text{aggr-1}), \text{aggr-2}, \text{opr})\rangle) \\
& \quad \neg\text{is-array}(\text{aggr-1}) \land \text{is-array}(\text{aggr-2}) \rightarrow \\
& \quad \mu(\text{aggr-2}; \langle s-\text{elem}:\text{aggr-infix-expr}(\text{aggr-1}, s-\text{elem}(\text{aggr-2}), \text{opr})\rangle) \\
& \quad \text{is-struct}(\text{aggr-1}) \land \text{is-struct}(\text{aggr-2}) \land \text{length}(\text{aggr-1}) = \text{length}(\text{aggr-2}) \rightarrow \\
& \quad \text{length}(\text{aggr-1}) \\
& \quad \text{LIST}_{i=1}^{\text{length}(\text{aggr-1})} \mu_0(\langle s-\text{aggr}:\text{aggr-infix-expr}(s-\text{aggr}\text{elem}(i, \text{aggr-1}),
\text{aggr}\text{elem}(i, \text{aggr-2}), \text{opr})\rangle) \\
& \quad \text{is-struct}(\text{aggr-1}) \land \text{is-scalar}(\text{aggr-2}) \rightarrow \\
& \quad \text{length}(\text{aggr-2}) \\
& \quad \text{LIST}_{i=1}^{\text{length}(\text{aggr-2})} \mu_0(\langle s-\text{aggr}:\text{aggr-infix-expr}(s-\text{aggr}\text{elem}(i, \text{aggr-1}),
\text{aggr}\text{elem}(i, \text{aggr-2}), \text{opr})\rangle) \\
& \quad \text{is-scalar}(\text{aggr-1}) \land \text{is-struct}(\text{aggr-2}) \rightarrow \\
& \quad \text{length}(\text{aggr-2}) \\
& \quad \text{LIST}_{i=1}^{\text{length}(\text{aggr-2})} \mu_0(\langle s-\text{aggr}:\text{aggr-infix-expr}(\text{aggr-1}, s-\text{aggr}\text{elem}(i, \text{aggr-2}), \text{opr})\rangle) \\
& \quad \text{is-scalar}(\text{aggr-1}) \land \text{is-scalar}(\text{aggr-2}) \rightarrow \\
& \quad \text{aggr-scalar\cdot da-infix}(s-\text{da}(\text{aggr-1}), s-\text{da}(\text{aggr-2}), \text{opr}) \\
& \quad T \rightarrow \text{error} \\
\end{align*}

Ref.: dim 8-29(86)

\begin{align*}
(67) & \quad \text{da-infix}(\text{da-1}, \text{da-2}, \text{opr}) = \\
& \quad \text{eda-infix}(\text{delete-length}(\text{da-1}), \text{delete-length}(\text{da-2}), \text{opr}) \\
\end{align*}

Ref.: eda-infix 9-20(87)

Note: The length of resulting strings in general is not known before the actual expression evaluation. The expression determining the length is replaced
by *.

(68) delete-length(da) =
    is-string(da) → μ(da;<s-length:*>)
    T = da

(69) aggr-prefix-expr(aggr,opr) =
    is-array(aggr) → μ(aggr;<s-elem:aggr-prefix-expr(s-elem(aggr),opr)>)
    is-struct(aggr) →
        length(aggr)
        LIST μₙ(<s-aggr:aggr-prefix-expr(s-aggr·elem(i,aggr),opr)>)
    is-scalar(aggr) → aggr-scalar·da-prefix(s-da(aggr),opr)

(70) da-prefix(da,opr) =
    eda-prefix(delete-length(da),opr)

Ref.: eda-prefix 9-28(117)

(71) aggr-ref(ref,at) =
    is-var-ref(ref,at) → aggr₁
    (is-prop-var v is-defined v is-based) (attr₀) → aggr-entry(aggr₁,s-ap(ref))
    is-entry-ccnst(attr₀) → scalar-entry(aggr-scalar(attr₀),s-ap(ref))
    is-file-ccnst(attr₀) → aggr-scalar(FILE)
    (is-label-ccnst v is-format-ccnst) (attr₀) → aggr-scalar(LABEL)
    is-generic(attr₀) → aggr-generic(ref-g₀,s-ap(ref),at)
    is-BUILTIN(attr₀) → aggr-builtin(id₀, arg-list•s-ap(ref), at)

where:
    attr₀ = s-n(ref)(at)
    aggr₁ = sub-aggr(s-aggr(attr₀),s-nl(ref),egl₁)
    egl₁ = eval-gl(tail•s-id-list(ref),aggr₀)
    ref-g₀ = generic-sel(attr₀,attr-list₁)
    attr-list₁ = attr-expr-list(arg-list•s-ap(ref),at)

Ref.: is-var-ref 8-9(21)
    aggr-builtin 12-7(18)
    arg-list 8-4(6)
    eval-gl 8-30(98)
    generic-sel 6-31(77)
    attr-expr-list 6-35(96)
ABSTRACT SYNTAX AND INTERPRETATION OF PL/I

(72) \[ \text{aggr-entry(aggr, ap)} = \]
\[ \begin{align*}
\text{is-array}(aggr) & \rightarrow \mu(\text{aggr}; \langle s-elem: \text{aggr-entry}(s-elem(aggr), ap) \rangle) \\
\text{length}(aggr) & \\
\text{is-struct}(aggr) & \rightarrow \text{LIST} \left[ \mu_s(\langle s-aggr: \text{aggr-entry}(s-aggr-elem(i, aggr), ap) \rangle) \right] \\
\text{is-scalar}(aggr) & \rightarrow \text{scalar-entry}(aggr, ap)
\end{align*} \]

(73) \[ \text{scalar-entry(aggr, ap)} = \]
\[ \begin{align*}
\text{is-}(ap) & \rightarrow \text{aggr} \\
\text{is-entry}\ast\text{da}(aggr) & \rightarrow \text{scalar-entry(s-ret-type}\ast\text{da}(aggr), \text{tail}(aggr)) \\
T & \rightarrow \text{error}
\end{align*} \]

(74) \[ \text{aggr-generic(ref, ap, at)} = \]
\[ \text{scalar-entry(aggr-ref(ref, at), ap)} \]

(75) \[ \text{sub-aggr(aggr, sl, egl)} = \]
\[ \begin{align*}
\text{is-}\ast\text{-list}(sl) \& \text{is-}(egl) & \rightarrow \text{aggr} \\
\text{is-array}(aggr) \& \left( \text{is-}(sl) \lor \text{is-}\ast\text{head}(sl) \right) & \rightarrow \\
\mu(\text{aggr}; \langle s-elem: \text{sub-aggr}(s-elem(aggr), \text{tail-1}(sl), egl) \rangle) \\
\text{is-array}(aggr) & \rightarrow \text{sub-aggr}(s-elem(aggr), \text{tail}(sl), egl) \\
\text{is-struct}(aggr) \& \text{is-}(egl) & \rightarrow \\
\text{sub-aggr}(s-aggr-elem(\text{head}(egl), aggr), sl, \text{tail}(egl)) \\
T & \rightarrow \text{error}
\end{align*} \]

8.2.5.2 EValuated attributes of array expressions

For the syntactic expansion of array expressions it is necessary to know array-bounds before the actual evaluation. The instruction \text{array-eva-expr(expr)} returns the aggregate attributes of the result of the expression with evaluated array bounds, where expr is the pre-evaluated expression. For the pre-evaluation of expressions see 8.2.1.
(76) \[ \text{array-expr} = \]
\[ \text{is-array-aggr-expr}(expr, \text{AT}) \rightarrow \text{PASS} \ast \]
\[ \text{is-infix-expr-1}(expr) \rightarrow \]
\[ \text{PASS} : \text{array-eva-infix}(\text{eva-1}, \text{eva-2}) ; \]
\[ \text{eva-1:array-eva-expr}(\text{s-cp-1}(expr)) , \]
\[ \text{eva-2:array-eva-expr}(\text{s-cp-2}(expr)) \]
\[ (\text{is-prefix-expr-1} \land \text{is-paren-expr-1})(expr) \rightarrow \text{array-eva-expr}(\text{s-cp}(expr)) \]
\[ \text{is-func-var}(\text{attr}_0) \land \text{is-base-attr}(\text{attr}_0) \rightarrow \]
\[ \text{PASS} : \text{sub-aggr}(\text{s-eva-s-gen}(expr), \text{s-sl}(expr), \text{eqI}_1) \]
\[ \text{is-defined}(\text{attr}_0) \rightarrow \text{PASS} : \text{sub-aggr}(\text{den}_0, \text{s-sl}(expr), \text{eqI}_1) \]
\[ \text{is-BUILTIN}(\text{attr}_0) \rightarrow \]
\[ \text{array-eva-built-in}(\text{head}\ast \text{id-list}(expr), \text{arg-list}\ast \text{s-at}(expr), 1) \]
\[ \text{T} \rightarrow \text{error} \]

where:
\[ \text{attr}_0 = \text{s-n}(expr, \text{AT}) \]
\[ \text{den}_0 = \text{s-n}(expr, \text{BN}) \]
\[ \text{eqI}_1 = \text{eval-cl}(\text{tail}\ast \text{id-list}(expr), \text{s-aggr}(\text{attr}_0)) \]

Ref.: \text{arg-list 8-4(6)}
\text{eval-cl 8-30(34)}

(77) \[ \text{array-eva-infix}(\text{eva-1}, \text{eva-2}) = \]
\[ \text{is-<}(\text{eva-2}) \rightarrow \text{eva-1} \]
\[ \text{is-<}(\text{eva-1}) \rightarrow \text{eva-2} \]
\[ \text{dim}(\text{eva-1}) = \text{dim}(\text{eva-2}) = 0 \rightarrow \ast \]
\[ \text{dim}(\text{eva-1}) = \text{dim}(\text{eva-2}) \land \text{s-lbd}(\text{eva-1}) = \text{s-lbd}(\text{eva-2}) \land \]
\[ \text{s-uid}(\text{eva-1}) = \text{s-uid}(\text{eva-2}) \rightarrow \]
\[ \mu(\text{eva-1}; \langle\text{s-elem:array-eva-infix}(\text{s-elem}(\text{eva-1}), \text{s-elem}(\text{eva-2}))\rangle) \]
\[ \text{T} \rightarrow \text{error} \]

Ref.: \text{dim 8-29(66)}

(78) \[ \text{array-eva-built-in}(\text{id}, \text{expr-list}, i) = \]
\[ \text{is-<>}(\text{expr-list}) \rightarrow \text{PASS} \ast \]
\[ \text{T} \rightarrow \]
\[ \text{PASS} : \text{array-eva-infix}(\text{eva-1}, \text{eva-2}) ; \]
\[ \text{eva-1:array-eva-built-in-arg}(\text{id}, \text{head}(\text{expr-list}), i), \]
\[ \text{eva-2:array-eva-built-in}(\text{id}, \text{tail}(\text{expr-list}), i + 1) \]

where:
\[ i_i = (\text{id} \ast (\text{mk-id}(\text{MIN}), \text{mk-id}(\text{MIN})) \rightarrow 1, \]
\[ \text{T} \rightarrow i) \]

8. ASSIGNMENT STATEMENT, EXPRESSION EVALUATION, REFERENCE TO VARIABLES 27
8.2.5.3 Attributes of scalar references in non-entry context

An entry value resulting from the evaluation of a reference in non-entry context is always invoked as a function. The function final-ret-type returns the attributes of the final result.

8.3 Evaluation of References to Variables

The generation referenced by a reference ref to a variable is evaluated by the instruction eval-ref-gen(ref). The referenced variable may be a proper variable of any storage class, a defined variable, or a based variable.

Note: For pre-evaluation of references see 8.2.1.

Note: With the help of the function ref-dim the correct length of the subscript-list is tested.
(84) \[ \text{ref-dim}(\text{aggr},\text{ecl}) = \]
\[ \begin{align*}
\text{is-\text{x}(ecl)} & \rightarrow \text{dim}(\text{aggr}) \\
\text{is-array}(\text{aggr}) & \rightarrow \text{dim}(\text{aggr}) + \text{ref-dim}(\text{base-elem}(\text{aggr}),\text{ecl}) \\
\text{is-struct}(\text{aggr}) & \rightarrow \text{ref-dim}(\text{s-aggr}\cdot\text{elem}(\text{head}(\text{ecl}),\text{aggr}),\text{tail}(\text{ecl})) \\
T & \rightarrow \text{error}
\end{align*} \]

(85) \[ \text{base-elem}(\text{aggr}) = \]
\[ \begin{align*}
\text{is-array}(\text{aggr}) & \rightarrow \text{base-elem}\cdot\text{elem}(\text{aggr}) \\
T & \rightarrow \text{aggr}
\end{align*} \]

(86) \[ \text{dim}(\text{aggr}) = \]
\[ \begin{align*}
\text{is-array}(\text{aggr}) & \rightarrow 1 + \text{dim}\cdot\text{elem}(\text{aggr}) \\
T & \rightarrow 0
\end{align*} \]

(87) \[ \text{eval-sub-gen}(\text{ref},\text{rl}) = \]
\[ \begin{align*}
\text{is-isub-def}(\text{attr}_0) & \rightarrow \\
\text{pass-sub-gen}(\text{gen},\text{rl}); \\
\text{gen} &: \text{eval-gen}(\text{ref}) \\
T & \rightarrow \text{eval-issub-gen}(\text{ref},\text{rl})
\end{align*} \]

(88) \[ \text{eval-gen}(\text{ref}) = \]
\[ \begin{align*}
\text{is-typ-var}(\text{attr}_0) & \& \text{is-\text{x-s-ptr}(ref)} \& \text{is-based}(\text{attr}_0) \rightarrow \text{PASS}: \text{s-gen}(\text{ref}) \\
\text{is-defined}(\text{attr}_0) & \& \text{is-non-varying}(\text{aggr}_0) \& \text{is-\text{x-s-ptr}(ref)} \& \\
\text{is-based}(\text{attr}_{-0}) & \& \text{is-defined}(\text{attr}_{-0}) \& \text{is-var-ref}\cdot\text{s-base}(\text{attr}_{-0}) \rightarrow \\
\text{eval-def-gen}(\text{ref}) \\
T & \rightarrow \text{ERROR}
\end{align*} \]

where:
\[ \text{attr}_{-0} = \text{s-\text{x-s-base}(attr}_0) (\text{AT}) \]

Ref.:
\[ \text{is-var-ref} 8-9(21) \]

Note: For proper and based variables the pre-evaluation already has entered the current generation into the s-gen component of the reference.

(89) \[ \text{is-non-varying}(\text{aggr}) = \]
\[ \neg(\exists \text{e} (\neg \text{is-\text{x-s-varying}(aggr})) \]

(90) \[ \text{is-isub-def}(\text{attr}) = \]
\[ \text{is-defined}(\text{attr}) \& (\exists \text{e} (\text{is-isub-s-s-\text{x-s-base}(attr})) \]

8. ASSIGNMENT STATEMENT, EXPRESSION EVALUATION, REFERENCE TO VARIABLES 29
8.3.1 EVALUATION OF SUBSCRIPTS AND NAME QUALIFICATION

The instruction \( \text{mk-rl}(s1, e1, eva) \) returns a list of integer values and \(*\)'s, representing evaluated subscripts and name qualifications. \( s1 \) is the subscript list, \( e1 \) the evaluated qualification list, and \( eva \) the evaluated aggregate attributes of the referenced variable. If subscript values exceed array bounds, the SUBSCRIPT RANGE condition is called.

(91) \[
\text{mk-rl}(s1, e1, eva) = \\
\text{is-<>(e1)} \land \text{is-<->list}(s1) \rightarrow \text{PASS:<>}
\]

\[
\text{is-<}(s1) \rightarrow \text{PASS-mk-rl-1}(e1, eva)
\]

\[
\text{is-struct}(eva) \land \text{is-<>(e1)} \rightarrow
\]

\[
\text{mk-list}(\text{head}(e1), \text{list});
\text{list:mk-rl}(s1, \text{tail}(e1), s-aggr\cdot elem(\text{head}(e1), eva))
\]

\[
\text{is-array}(eva) \rightarrow
\]

\[
\text{mk-list}(1, \text{list});
\text{list:mk-rl}(\text{tail}(s1), e1, \text{e1}\cdot elem(eva));
\text{test-subscript}(\text{s-lbd}(eva), \text{s-ubd}(eva), i);
\text{i:eval-into-exp} \text{(head}(s1))
\]

\( T \rightarrow \text{error} \)

Ref.: eval-into-exp 8-22(60)

(92) \[
\text{mk-rl-1}(e1, eva) =
\]

\[
\text{is-<>(e1)} \rightarrow <>
\]

\[
\text{is-struct}(eva) \rightarrow \langle \text{head}(e1) = \text{mk-rl-1}(\text{tail}(e1), s-aggr\cdot elem(\text{head}(e1), eva))
\]

\[
\text{is-array}(eva) \rightarrow \langle >\text{mk-rl-1}(e1, s-elem(eva))
\]

\( T \rightarrow \text{error} \)

(93) \[
\text{test-subscript}(k-1, k-2, i) =
\]

\[
\text{is-<}(i) \lor (k-1 \leq i \leq k-2) \rightarrow \text{null}
\]

\( T \rightarrow \text{call-cond} \text{(SUBEQ)} \)

Ref.: \text{call-cond} 10-18(54)
8.3.2 REFERENCE TO DEFINED VARIABLES

8.3.2.1 isub-defined variables

The number of dimensions of the defined variable determines the number of isub-variables to be created. The values of the isub-variables are determined by the evaluated subscripts of the reference to the defined variable. The values are entered as s-v component of the references to the isub-variables in the base reference by the function insert-isubs. The base reference, modified in this way, is evaluated and the resulting generation is treated like in the simple-defined case. That part of the reference list not used to give values to isubs is used to determine the appropriate sub-generation.

\[
\text{eval-isub-ref}(\text{Ref}, rl) = \\
\text{length}(rl) \geq \text{dim}(\text{aggr}) \land \text{is-non-varying}(\text{aggr}) \land \\
\text{is-var-ref}(\text{isub-base}(\text{attr}), \text{AT}) \land \neg (\text{is-based} \land \neg \text{is-defined})(\text{attr-b}) \land \\
\text{is-corr-} (\text{base-elem}(\text{aggr}), \text{aggr-b}) \land \text{is-s-subs}(\text{attr}) \\
\text{pass-sub-gen}(\text{gen}, \text{rest-list}(rl, \text{dim}(\text{aggr}))) \\
\text{gen} = \text{eval-ref-gen}(\text{insert-isubs}(\text{isub-list}(rl, \text{dim}(\text{aggr})))) \\
\text{T} \rightarrow \text{error}
\]

where:
\[
\text{attr-b} = \text{s-s-base}(\text{attr}), \text{AT} \\
\text{aggr-b} = \text{aggr-ref}(\text{isub-base}(\text{attr}), \text{AT})
\]

Ref.:
\text{is-var-ref} 6-9(21)
\text{is-corr-} 6-20(57)
\text{aggr-ref} 6-25(71)

\[
\text{rest-list}(rl, k) = \\
\mu(\langle ; \langle \text{elem}(i) : \text{elem}(i + k, rl) \rangle | 1 \leq i \leq (\text{length}(rl) - k))
\]

\[
\text{insert-isubs}(\text{ref}, \text{list}) = \\
\mu(\langle ; \langle \text{elem}(i) : \text{elem}(i, \text{list}) \rangle | \text{is-} \text{isub-ref}(\text{ref}))
\]

\[
\text{isub-val}(i, \text{list}) = \\
\neg \text{is-s-subs}(i, \text{list}) \rightarrow \text{elem}(i, \text{list})
\]

T \rightarrow =
The discrimination between the two cases is made by the predicates is-corresp and is-overlay. In the simple-defined case the pointer-part and the mapping information of the base generation are retained, and only the eva-part is replaced by the evaluated aggregate attributes of the defined variable. The eva-part and the mi-part of the resulting generation then may differ in array-counts and string-lengths. In the overlay-defined case a new generation is formed from the base generation by determining a single pointer value identifying the referenced part of the base, which forms the pointer part of the new generation, and installing the evaluated aggregate attributes of the defined variable as the eva-part and the mi-part of the new generation.

\begin{align*}
(99) \quad \text{is-sub-list}(r, k) = \\
\mu(\langle i \rangle; \langle \text{elem}(i); \text{elem}(i, r) \rangle \mid 1 \leq i \leq k)
\end{align*}

### 8.3.2.2 Simple and overlay defined variables

\begin{align*}
(100) \quad \text{eval-def-gen}(\text{ref}) = \\
\text{is-corresp}(\text{aggr}_0, \text{aggr-b}_\lambda) \land \text{is-pos}(\text{attr}_0) \implies \\
\text{pass-simple-def-gen}(\text{gen}, \text{eva}_0); \\
\text{gen}=\text{eval-def-gen}(\text{s-base}(\text{attr}_0))
\end{align*}

\begin{align*}
\text{is-overlay}(\text{attr}_0, \text{AT}) \implies \\
\text{pass-overlay-gen}(\text{gen}, \text{eva}_0, i); \\
\text{gen}=\text{eval-ref-gen}(\text{s-base}(\text{attr}_0)), \\
i: \text{eval-into-expr}(\text{s-pos}(\text{attr}_0))
\end{align*}

T \rightarrow \text{error}

where:
\begin{align*}
\text{aggr-b}_\lambda = \text{aggr-ref}(\text{s-base}(\text{attr}_0), \text{AT})
\end{align*}

Ref.:
\begin{align*}
is-corresp 6-26(51) \\
\text{eval-into-expr} 8-22(66) \\
\text{aggr-ref} 6-25(71)
\end{align*}

\begin{align*}
(101) \quad \text{simple-def-gen}(\text{gen}, \text{eva}) = \\
\text{test-simple-def}(\text{s-eva}(\text{gen}), \text{eva}) \implies \\
\mu_0(\langle \text{evaseva}, \langle \text{mi:s-mi}(\text{gen}) \rangle, \langle \text{s-pf:s-pf}(\text{gen}) \rangle)
\end{align*}

T \rightarrow \text{error}
(102) \text{test-simple-def}(\text{eva-1}, \text{eva-2}) = \\
\quad \text{is-array}(\text{eva-1}) \rightarrow \\
\quad \text{s-lbd}(\text{eva-1}) \leq \text{s-lbd}(\text{eva-2}) \land \text{s-ubd}(\text{eva-1}) \geq \text{s-ubd}(\text{eva-2}) \land \\
\quad \text{test-simple-def}(\text{s-elem}(\text{eva-1}), \text{s-elem}(\text{eva-2})) \\
\quad \text{is-struct}(\text{eva-1}) \rightarrow \\
\quad (\text{length}(\text{eva-1}) \\
\quad \text{Et} \quad \text{test-simple-def}(\text{s-aggr-elem}(i, \text{eva-1}), \text{s-aggr-elem}(i, \text{eva-2})) \\
\quad \text{i} = 1 \\
\quad \text{is-string-s-da}(\text{eva-1}) \rightarrow \text{s-length-s-da}(\text{eva-1}) \geq \text{s-length-s-da}(\text{eva-2}) \\
\text{eva-1} = \text{eva-2} \rightarrow T \\
T \rightarrow 1

(103) \text{is-overlay}(\text{attr}, \text{at}) = \\
\quad (\text{is-bit-aggr}\text{s-aggr}(\text{attr}) \land \text{is-bit-aggr}(\text{aggr-b1}) \land \text{is-char-aggr}\text{s-aggr}(\text{attr}) \land \\
\quad \text{is-char-aggr}(\text{aggr-b1}) ) \land \text{is-connected}(\text{s-base}(\text{attr}), \text{at})

where:
\begin{align*}
\text{aggr-b1} &= \text{aggr-ref}(\text{s-base}(\text{attr}), \text{at})
\end{align*}

Ref.:
\begin{align*}
\text{aggr-ref} & \odot \text{25}(71)
\end{align*}

(104) \text{overlay-gen}(\text{gen, eva, i}) = \\
\quad (\text{string-extent-eva}\text{-eva}(\text{gen}) - \text{posn}(i)) \geq (\text{string-extent}(\text{eva}) - 1) \rightarrow \\
\quad \mu_0(\langle \text{<e-eva-eva}>; \langle \text{s-eva:eva} >, \\
\quad \langle \text{s-fp: str-part}(\text{s-base}\text{-gen-da}(\text{gen}), \text{posn}(i), \\
\quad \text{string-extent}(\text{eva})); \text{s-pf}(\text{gen}) >) \\
T \rightarrow \text{error}
\end{align*}

Ref.:
\begin{align*}
\text{str-part} & \odot \text{7-13}(35) \\
\text{gen-da} & \odot \text{7-17}(53)
\end{align*}

Note: The properties of the function \text{str-part}, which returns a single pointer value, are defined in 7.3.3.1.

(105) \text{posn}(i) = \\
\quad \text{is-5}(i) \rightarrow 1, \\
\quad T \rightarrow 1
ABSTRACT SYNTAX AND INTERPRETATION OF E/I  
30 April 1969

(106) is-bit-aggr(agr) =
\[
\text{is-string*da}(agr) \land \text{(is-BIT*base*da}(agr) \lor \text{is-bin-pics*da}(agr)) \land \text{is-5*varying*da}(agr) \land \text{is-UNAL*dens}(agr) \land \\
\text{is-scalar}(agr) \rightarrow 1
\]
\[
\text{is-array}(agr) \rightarrow \text{is-bit-aggr*elem}(agr)
\]
\[
\text{is-struct}(agr) \rightarrow \prod_{i=1}^{\text{length}(agr)} \text{is-bit-aggr*elem}(i,agr)
\]

(107) is-char-aggr(agr) =
\[
\text{is-string*da}(agr) \land \text{(is-CHAR*base*da}(agr) \lor \text{is-pics*da}(agr) \land \text{is-bin-pics*da}(agr)) \land \text{is-5*varying*da}(agr) \land \text{is-UNAL*dens}(agr) \land \\
\text{is-scalar}(agr) \rightarrow 1
\]
\[
\text{is-array}(agr) \rightarrow \text{is-char-aggr*elem}(agr)
\]
\[
\text{is-struct}(agr) \rightarrow \prod_{i=1}^{\text{length}(agr)} \text{is-char-aggr*elem}(i,agr)
\]

(108) string-extent(eva) =
\[
\text{is-array}(eva) \rightarrow (s-ubd(eva) - s-lbd(eva) + 1) \cdot \text{string-extent*elem}(eva)
\]
\[
\text{is-struct}(eva) \rightarrow \sum_{i=1}^{\text{length}(eva)} \text{string-extent*elem}(i,eva)
\]
\[
\text{(is-string \lor is-pic)(s-da}(eva)) \rightarrow \text{str-length*da}(eva)
\]

Ref.: str-length 9-41(164)

8.3.3 CONNECTED VARIABLES

For overlay defining, parameter passing, and record input/output it is necessary to determine whether the reference to a variable is statically recognizable as being connected. The property is tested by the predicate is-connected(ref,at), where ref is the text of the reference and at is the attribute directory.
(109) is-connected(ref,at) =

\[\text{is-ccnn-1}(\text{aggr}, \text{sl}, \text{egl}) \land \]
\[\text{is-ccnn-2}(\text{ref}, \text{at}) \lor \text{is-ccnn-3}(\text{aggr}) \land \text{is-scalar}(\text{aggr})\]

where:
attr = \text{s-n}(\text{ref})(\text{at})
aggr = \text{aggr-ref}(\text{ref}, \text{at})
egl = \text{eval-\text{gl}}(\text{tail} \cdot \text{s-id-list}(\text{ref}), \text{s-aggr}(\text{attr}))
Ref.: \text{aggr-ref 8-25(71)}

(110) is-ccnn-1(\text{aggr}, \text{sl}, \text{egl}) =

\[\text{is-*list}(\text{sl}) \land \text{is-<>}(\text{egl}) \rightarrow \top\]
\[\text{is-array}(\text{aggr}) \land (\text{is-<>}(\text{sl}) \lor \text{is-*head}(\text{sl})) \rightarrow \bot\]
\[\text{is-array}(\text{aggr}) \rightarrow \text{is-ccnn-1}(\text{s-elem}(\text{aggr}), \text{tail}(\text{sl}), \text{egl})\]
\[\text{is-struct}(\text{aggr}) \land \neg \text{is-<>}(\text{egl}) \rightarrow \]
\[\text{is-ccnn-1}(\text{s-aggr-elem}(\text{head}(\text{egl}), \text{aggr}), \text{sl}, \text{tail}(\text{egl}))\]
\[T \rightarrow \text{error}\]

(111) is-ccnn-2(\text{ref}, \text{at}) =

\[\text{is-prop-var}(\text{attr}) \rightarrow \neg \text{is-*s-stg-cl}(\text{attr}) \lor \neg \text{is-*s-connected}(\text{attr})\]
\[\text{is-based}(\text{attr}) \rightarrow \top\]
\[\text{is-isub-def}(\text{attr}) \rightarrow \bot\]
\[\text{is-ccrresp}(\text{s-aggr}(\text{attr}), \text{aggr-b}) \land \text{is-*s-pos}(\text{attr}) \rightarrow \]
\[\text{is-connected}(\text{s-base}(\text{attr}), \text{at}) \land \neg \text{is-CTL-s-stg-cl}(\text{attr-b}) \land \]
\[\text{is-ccrresp-ccnn}(\text{s-aggr}(\text{attr}), \text{aggr-b}, \text{sl}(\text{ref}), \text{egl})\]
\[\text{is-overlay}(\text{attr}, \text{at}) \rightarrow \top\]
\[T \rightarrow \text{error}\]

where:
attr = \text{s-n}(\text{ref})(\text{at})
attr-b = \text{s-n-s-base}(\text{attr})(\text{at})
aggr-b = \text{aggr-ref}(\text{s-base}(\text{attr}), \text{at})
egl = \text{eval-\text{gl}}(\text{tail} \cdot \text{s-id-list}(\text{ref}), \text{s-aggr}(\text{attr}))
Ref.: \text{is-ccrresp 6-20(51)}
\text{aggr-ref 8-25(71)}

8. ASSIGNMENT STATEMENT, EXPRESSION EVALUATION, REFERENCE TO VARIABLES. 35
(112) \( \text{is-corresp-conn}(\text{agr}r-1, \text{agr}r-2, s_1, e_1) = \)

\[
\text{is-array}(\text{agr}r-1) \land (\text{is-<>}(s_1) \lor \text{is-**head}(s_1)) \\
\text{equal-int}(s_1\text{bd}(\text{agr}r-1), s_1\text{bd}(\text{agr}r-2)) \land \\
\text{equal-int}(s_1\text{ubd}(\text{agr}r-1), s_1\text{ubd}(\text{agr}r-2)) \land \\
\text{is-corresp-conn}(s_1\text{elem}(\text{agr}r-1), s_1\text{elem}(\text{agr}r-2), \text{tail}(s_1), e_1) \\
\text{is-array}(\text{agr}r-1) \rightarrow \text{is-corresp-conn}(s_1\text{elem}(\text{agr}r-1), s_1\text{elem}(\text{agr}r-2), \text{tail}(s_1), e_1) \\
\text{is-struct}(\text{agr}r-1) \land \text{is-<>}(e_1) \\
\text{length}(\text{agr}r-1) \\
\begin{align*}
\text{is-struct}(\text{agr}r-1) & \rightarrow \\
\text{is-corresp-conn}(s_1\text{aggr}\text{elem}(i_1, \text{agr}r-1), s_1\text{aggr}\text{elem}(i_1, \text{agr}r-2), s_1, \text{<>}) & \text{is-struct}(\text{agr}r-1) & \rightarrow \\
\text{is-corresp-conn}(s_1\text{aggr}\text{elem}(i_1, \text{agr}r-1), s_1\text{aggr}\text{elem}(i_1, \text{agr}r-2), s_1, \text{tail}(e_1)) & \text{is-scalar}(\text{agr}r-1) & \rightarrow \\
\text{is-string}\text{~s\text{-}da}(\text{agr}r-1) & \lor \\
\text{equal-int}(s_1\text{~length}\text{~s\text{-}da}(\text{agr}r-1), s_1\text{~length}\text{~s\text{-}da}(\text{agr}r-2))
\end{align*}
\]

where:
\[
i_1 = \text{head}(e_1)
\]

(113) \( \text{equal-int}(\text{expr}-1, \text{expr}-2) = \)

\[
\text{is-signd-intg}(\text{expr}-1) \land \text{is-signd-intg}(\text{expr}-2) \land \\
\text{intg-val}(\text{expr}-1) = \text{intg-val}(\text{expr}-2)
\]

Ref.: \( \text{is-signd-intg 6-10(29)} \)
\( \text{intg-val 6-10(27)} \)

(114) \( \text{is-corr-3}(\text{agr}) = \)

\[
\text{is-array}(\text{agr}) \rightarrow T \\
\text{length}(\text{agr}) \\
\text{is-struct}(\text{agr}) \rightarrow \begin{align*}
\text{is-corr-3} & \star s_\text{aggr}\text{elem}(i, \text{agr}) & \text{is-struct}(\text{agr}) \\
\text{is-scalar}(\text{agr}) & \rightarrow \text{is-string}\text{~s\text{-}da}(\text{agr})
\end{align*}
\]

8.3.4 SUB-GENERATIONS OF GENERATIONS

The function \( \text{sub-gen}(\text{gen}, \text{rl}) \) gives the sub-generation of \( \text{gen} \) determined by the reference list \( \text{rl} \). The reference list is a list of integer values and/or *'s, resulting from the evaluation of subscripts and name qualification of the reference to a variable (cf. 8.3.1).

(115) \( \text{sub-gen}(\text{gen}, \text{rl}) = \)

\[
\mu_0(\langle s\text{-eva} : \text{sub-eva}(s\text{-eva}(\text{gen}), \text{rl}) \rangle, \langle s\text{-mi} : \text{sub-eva}(s\text{-mi}(\text{gen}), \text{rl}) \rangle, \\
\langle s\text{-pp} : \text{sub-pp}(s\text{-pp}(\text{gen}), \text{rl}, s\text{-mi}(\text{gen})) \rangle)
\]

36 8. ASSIGNMENT STATEMENT, IMPEDIMENT EVALUATION, REFERENCE TO VARIABLES
30 April 1969

ABSTRACT SYNTAX AND INTERPRETATION OF PL/I

(116) \text{sub-ff}(\text{FF,rl,mi}) = \\
\quad \text{is-<>}(\text{rl}) \rightarrow \text{FF} \\
\quad \text{is-rtr-val}(\text{FF}) \rightarrow \text{sub-ff-1}(\text{FF,rl,mi}) \\
\quad \text{is-*head}(\text{rl}) \rightarrow \text{LIST} \\
\quad \text{length}((\text{pp})) \\
\quad \text{sub-pp}(\text{elem}(i,\text{pp}),\text{tail}(\text{rl}),\text{s-elem}(\text{mi})) \\
\quad T \rightarrow \text{sub-ff}(\text{elem}(\text{head}(\text{rl})) - 1,\text{pp}),\text{tail}(\text{rl}),\text{s-elem}(\text{mi})) \\
\text{Ref.:} \quad \text{is-rtr-val} 3-15(36)

(117) \text{sub-ff-1}(\text{p,rl,mi}) = \\
\quad \text{is-<>}(\text{rl}) \rightarrow \text{p} \\
\quad \text{is-*head}(\text{rl}) \land \text{is-array}(\text{mi}) \rightarrow \\
\quad \text{s-ubd}(\text{mi}) \\
\quad \text{LIST} \\
\quad \text{sub-pp-1}((\text{map}(\text{mi},i))\text{p},\text{tail}(\text{rl}),\text{s-elem}(\text{mi})) \\
\quad (\text{is-array} \lor \text{is-struct})(\text{mi}) \rightarrow \\
\quad \text{sub-pp-1}((\text{map}(\text{mi},\text{head}(\text{rl})))\text{p},\text{tail}(\text{rl}),\text{aggr-part}(\text{mi},\text{head}(\text{rl}))) \\
\text{Ref.:} \quad \text{map} 7-12(32)

Note: The properties of the storage mapping function \text{map}, which gives a pointer value, are described in 7.3.

(118) \text{sub-eva}(\text{eva,rl}) = \\
\quad \text{is-<>}(\text{rl}) \rightarrow \text{eva} \\
\quad \text{is-*head}(\text{rl}) \land \text{is-array}(\text{eva}) \rightarrow \mu(\text{eva};\text{s-elem}:\text{sub-eva}(\text{s-elem}(\text{eva}),\text{tail}(\text{rl}))) \\
\quad (\text{is-array} \lor \text{is-struct})(\text{eva}) \land \text{lb}(\text{eva}) \leq \text{head}(\text{rl}) \leq \text{ub}(\text{eva}) \rightarrow \\
\quad \text{sub-eva}(\text{aggr-part}(\text{eva},\text{head}(\text{rl})),\text{tail}(\text{rl})) \\
\quad T \rightarrow \text{error} \\
\text{Ref.:} \quad \text{lb} 7-13(37) \\
\quad \text{ub} 7-13(36)

(119) \text{aggr-part}(\text{aggr},i) = \\
\quad \text{is-array}(\text{aggr}) \rightarrow \text{s-elem}(\text{aggr}) \\
\quad \text{is-struct}(\text{aggr}) \rightarrow \text{s-aggr-elem}(i,\text{aggr})
This chapter describes operands and the operations that can be applied to them. Values and the functions \textit{value} and \textit{rep} for the transition between values and their value representations are defined in 9.1, the evaluation of \textit{infix} expressions in 9.3, the evaluation of \textit{prefix} expressions in 9.4, the \textit{convert-1} instruction in 9.5. In 9.2, some auxiliary definitions of evaluated data attributes are given, and in 9.6, all definitions necessary for the interpretation of pictures are collected. Since all data attributes occurring in this chapter are evaluated scalar data attributes we shall refer to them in the sequel just as data attributes.

**Metavariables**

- \textit{eva} \hspace{1cm} \textit{is-correct-eva}\hspace{1cm} evaluated aggregate attribute
- \textit{eda, eda-tg} \hspace{1cm} \textit{is-correct-eda}\hspace{1cm} evaluated scalar data attribute that can be associated with a value (cf. 9.1) or an area attribute
- \textit{v, w, x, v-im} \hspace{1cm} \textit{is-value}\hspace{1cm} value
- \textit{vr} \hspace{1cm} \textit{is-vr}\hspace{1cm} value representation
- \textit{op} \hspace{1cm} \textit{is-op}\hspace{1cm} operand
- \textit{opr} \hspace{1cm} \textit{is-operator}\hspace{1cm} operator
- \textit{base} \hspace{1cm} \textit{is-CHAR v is-BIT}\hspace{1cm} base of a string
- \textit{u, w, i, j} \hspace{1cm} \textit{is-intg-val}\hspace{1cm} integer value
- \textit{a} \hspace{1cm} \textit{is-prop-arithm(eda)}\hspace{1cm} arbitrary selector function

**Abbreviations**

For \textit{is-prop-arithm(eda)}:

\begin{align*}
\text{p-eda} &= \text{s-prec(eda)} \quad \text{precision of eda} \\
\text{g-eda} &= \text{s-scale-f(eda)} \quad \text{scale-factor of eda} \\
\text{base-eda} &= \begin{cases} 
\text{is-DEC-s-base(eda)} \rightarrow 10, \\
\text{is-BIN-s-base(eda)} \rightarrow 2 
\end{cases} \quad \text{number base of eda} \\
\text{n-eda} &= \text{max-prec(eda)} \quad \text{maximum number of digits associated with eda}
\end{align*}

for \textit{is-prop-arithm(eda)} & \textit{is-FLT-s-scale(eda)}:
max-flt-eda = max-flt(edata)  upper limit for values representable with edata
min-flt-eda = min-flt(edata)  lower limit for values representable with edata

for is-prop-arithms(edata) & is-CPLX-s-mode(edata):
eds-real = μ(eda; s-mode:REAL)  real data attribute corresponding to edata

for is-op(op), is-op(op-1), is-op(op-2):
eva-op = s-eva(op)  evaluated aggregate attribute part of operand op
eds-op = s-eda-eva(op)  evaluated data attribute of operand op
eds-op-1 = s-eda-eva(op-1)  evaluated data attribute of operand op-1
eds-op-2 = s-eda-eva(op-2)  evaluated data attribute of operand op-2
v-op = s-vs(op)  value representation part of operand op
v-op-1 = op-val(op-1)  value of operand op-1
v-op-2 = op-val(op-2)  value of operand op-2

9.1 VALUES, VALUE REPRESENTATIONS, OPERANDS, AND OPERATORS

A value is an object satisfying the predicate is-value. There are different types of values, corresponding to different types of data attributes; the class of these data attributes is defined by the predicate is-correct-edata and is essentially the class of evaluated scalar data attributes, excluding areas (which have no values) and certain nonsensical attributes, such as string data attributes with negative length, etc.

Given a value v and an evaluated aggregate attribute eva having a data attribute part eda which matches v, the function rep(eva,v) may be applied to yield a representation of v. Conversely, given a value representation vr and an evaluated aggregate attribute eva having a data attribute part eda which matches vr, the function value (eva,vr) will yield a value. The function value is defined in terms of the function val; only under certain conditions are val and rep inverses of each other.

9.1.1 A CLASS OF DATA ATTRIBUTES
ABSTRACT SYNTAX AND INTERPRETATION OF PL/I

(1)  is-correct-eda(eda) =

    is-prop-arithm(eda) \& (is-num-scale-f(eda) \equiv is-flt-scale(eda)) \&
    C < p-eda \& N-eda \& (is-string \& is-intg-values-length)(eda) \&
    s-length(eda) \geq 0 \& (is-area \& is-intg-values-size)(eda) \&
    s-size(eda) \geq 0 \& is-correct-pic(eda) \& is-ENTRY(eda) \& is-LABEL(eda) \&
    is-OFFSET(eda) \& is-PTR(eda) \& is-FIELD(eda) \& is-TASK(eda) \& is-EVENT(eda)

for: is-eda(eda)

Ref.: is-correct-pic 9-48(195)

9.1.2 VALUES

(2)  is-value =

    is-num-val \& is-char-val-list \& is-bit-val-list \& is-ptr-val \& is-event-val \&
    is-n

Ref.: is-ptr-val 3-15(36)

9.1.2.1 Numeric values

(3)  is-num-val =

    Note: This predicate characterizes the class of (real and complex) numbers that
    are either rational or have a rational real and imaginary part.

(4)  is-real-val(v) =

    is-num-val(v) \& imag(v) = 0

(5)  is-intg-val =

    Note: This predicate characterizes the class of all (positive, zero, and
    negative) integer values. They are elementary objects and belong to the
    class characterized by is-num-val.

2.1.2.2 Character values

(6)  is-char-val =

    is-alpham-char \& is-BLANK \& is-EQ \& is-PLUS \& is-MINUS \& is-ASTER \& is-SLASH \&
    is-LEFT-PAR \& is-RIGHT-PAR \& is-COMMA \& is-POINT \& is-APOSTE \& is-PERC \&
    is-SYMIC \& is-COLON \& is-NOT \& is-AND \& is-OR \& is-CT \& is-LT \& is-QUEST \&
    is-extralingual-char

(7)  is-alpham-char =

    is-letter \& is-digit \& is-BREAK

9. DATA, OPERATIONS AND CONVERSIONS 3
ABSTRACT SYNTAX AND INTERPRETATION OF PL/I

4 9. DATA, OPERATIONS AND CONVERSIONS
9.1.3 REPRESENTING AND RETRIEVING SCALAR VALUES

In this section, the relation between scalar values and their representations is treated; this is done axiomatically, by introducing two implementation-defined functions \text{val} and \text{rep} and formulating certain postulates concerning them. The function \text{val}, which differentiates the treatment of pictures from all other cases, is defined in terms of the function \text{val}. An instruction \text{test-rep} is defined that raises the SIZE or \text{STRZ} conditions if a value cannot be represented.

For each evaluated aggregate attribute \text{eva}, the set \text{vr-set(eva)} of value representations is taken, for the given \text{eva}, as the domain of the function \text{val(eva,vr)}. Also, a set \text{v-set(eda)} of values is defined, which is the domain of the function \text{rep(eva,v)} for a given \text{eva} whose attribute part is \text{eda}; this is the set of all values whose type matches \text{eda} and which, when represented with an \text{eva} corresponding to \text{eda}, do not raise the SIZE condition. Finally, a subset \text{v-0-set(eda)} of \text{v-set(eda)} is defined, which for non-arithmetic \text{eda} is \text{v-set(eda)} itself, and for arithmetic \text{eda} is determined by the precision and scale factor of \text{eda}. This \text{v-0-set(eda)} has the important property that its members are exactly representable with the given \text{eda}, i.e. the following can be derived from the axioms given below:

\begin{align*}
(15) \quad v \in v-0-set(eda(eva)) \Rightarrow val(eva,rep(eva,v)) = v
\end{align*}

Metavable

\begin{align*}
eda & \quad is-corect-eda & \text{in this section only} \\
& \quad -is-area
\end{align*}

\begin{align*}
(16) \quad v-set(eda) = \\
& \quad \{v \mid is-type-match(eda,v) \& (is-TASK(eda) \Rightarrow v < 2 \uparrow PRI-PREC) \& \ (is-arithm(eda) \Rightarrow -is-size-cond(eda,v))\}
\end{align*}
(17) \[ v-0-set(eda) = \]
\[ (\text{is-arithm} \& \text{is-CPLX-s-mode})(eda) \rightarrow \]
\[ \{ \text{cplx}(v-re,v-im) \mid v-re \in v-0-set(eda\text{-real}_1) \& v-im \in v-0-set(eda\text{-real}_1) \} \]
\[ \text{is-FIX-s-scale}(eda) \rightarrow \]
\[ \{ v \mid v \in v-set(eda) \& \text{is-intg-val}(v \cdot \text{base-eda}_1 \uparrow g-eda_1) \} \]
\[ \text{is-FLT-s-scale}(eda) \rightarrow \]
\[ \{ v \mid v \in v-set(eda) \& \]
\[ (\exists n) (\text{is-intg-val}(n) \& \text{is-intg-val}(v \cdot \text{base-eda}_1 \uparrow n) \&
\[ \text{abs}(v \cdot \text{base-eda}_1 \uparrow n) < \text{base-eda}_1 \uparrow p-eda_1) \}
\[ T \rightarrow v-set(eda) \]

Note: For arithmetic eda, this definition expresses the rule that \( p-eda_1 \) is the "number of digits" and \( g-eda_1 \) the "number of digits behind the decimal or binary point". Since no particular normalization rule has been assumed, the implementation-defined limits for floating-point numbers have been expressed as limits for the size of the number itself, not of the exponent. For later use, the definition has been given also for complex eda.

The functions

(18) \[ \text{val}(eva,v) = \]

(19) \[ \text{rep}(eva,v) = \]

are implementation-defined and are characterized by the axioms (22), (23), (24), (25), and (26).

(20) \[ v-r-set(eva) = \]
\[ \{ vr \mid (\exists v)(v \in v-set-s-da(eva) \& vr = \text{rep}(eva,v)) \} \]

Note: For given eva, this function yields the set of value representations constituting the range of the function \( \text{rep}(eva,v) \) when the domain of \( \text{rep}(eva,v) \) is restricted to the set of values representable with eva.

(21) \[ v-0-set-s-da(eva) \subseteq \{ v \mid (\exists vr)(v = \text{val}(eva,vr) \& vr \in v-r-set(eva)) \} \subseteq v-set-s-da(eva) \]

Note: This axiom defines domain and range of the function \( \text{val}(eva,v) \) and ensures that, for given eva, the range is at least \( v-0-set-s-da(eva) \).

(22) \[ v-r-set(eva) \subseteq \{ vr \mid \text{size}(vr) = \text{alloc-size}(eva) \} \]

Ref.: size 3-15(33)
alloc-size 7-11(26)

Note: This axiom defines domain and range of the function \( \text{rep}(eva,v) \).

6 9. DATA, OPERATIONS AND CONVERSIONS
ABSTRACT SYNTAX AND INTERPRETATION OF PL/I

(23) \[ vr \in vr-set(eva) \Rightarrow rep(eva, val(eva, vr)) = vr \]

Note: This axiom states that transition from a value representation to a value and back to a value representation leaves the given value representation unchanged. (The stronger consequence mentioned above (cf. (15)) concerns the opposite transition: from a value via a representation back to the value).

(24) \[ val(str-eva(base, j), vr) = \text{CONC} \begin{cases} \text{val(str-eva(base, 1), str-part(base, 1, 1)(vr))} \end{cases} \]

for: \((is-BIT \lor is-CHAR)(base)\)

Ref.: str-eva 7-14(45)
str-part 7-13(39)

Note: This axiom expresses the fact that unaligned strings are represented linearly in storage.

(25) \[ \text{is-pic-s-da(eva) \& is-UNAL-s-dens(eva)} \Rightarrow \]
\[ \text{val(eva, vr)} = \text{val(str-eva(str-base-s-da(eva), str-lengths-s-da(eva)), vr)} \]

Ref.: str-eva 7-14(45)
str-base 9-41(163)
str-length 9-41(164)

Note: This axiom is an analogue of axiom (25) for the case of unaligned pictures.

(26) \[ \text{value}(eva, vr) = \]
\[ \text{is-pic-s-da(eva)} \Rightarrow \text{pic-val(s-da(eva), val(eva, vr))} \]
\[ T \Rightarrow \text{val(eva, vr)} \]

Ref.: pic-val 9-61(240)

Before representing a value, the instruction test-rep(eva, v) tests whether SIZE or STRZ conditions have to be raised. If it is known that no condition can occur, then the function rep(eva, v) can be used directly.
(27) \[ \text{test-rep}(\text{eva}, v) = \]

\[ (\text{is-arithm} \& \& \text{is-CPLX-s-mode} \& \text{s-da}(\text{eva}) \& \]
\[ (\text{is-size-cond}(\text{eda-real}_1, \text{real}(v)) \vee \text{is-size-cond}(\text{eda-real}_1, \text{imag}(v))) \vee \]
\[ \text{is-REAL-s-mode} \& \text{s-da}(\text{eva}) \& \text{is-size-cond}(\text{s-da}(\text{eva}), \text{real}(v)) \rightarrow \]
\[ \text{call-cond}(\text{SIZE}) \]
\[ \text{is-string} \& \text{s-da}(\text{eva}) \& \text{is-str-size-cond}(\text{s-da}(\text{eva}), v) \rightarrow \]
\[ \text{pass-rep}(\text{eva}, \text{adjust-val}(\text{s-da}(\text{eva}), v)); \]
\[ \text{call-cond}(\text{STR2}) \]
\[ \text{is-pics} \& \text{s-da}(\text{eva}) \rightarrow \text{rep-pie}(\text{eva}, v) \]
\[ T \rightarrow \text{PASS:rep}(\text{eva}, \text{adjust-val}(\text{s-da}(\text{eva}), v)) \]

where:
\[ \text{eda-real}_1 = p(\text{s-da}(\text{eva}); \langle \text{s-mode}: \text{REAL} \rangle) \]

Ref.:
\[ \text{call-cond} 10-18(54) \]
\[ \text{rep-pie} 9-51(204) \]

(28) \[ \text{is-size-cond}(\text{eda}, v) = \]
\[ \text{is-FIX-s-scale}(\text{eda}) \rightarrow \text{abs}(v) \cdot \text{base-eda}_1 \pm \text{eda}_1 \geq \text{base-eda}_1 \pm p-eda_1 \]
\[ \text{is-FLT-s-scale}(\text{eda}) \rightarrow v \neq 0 \& \& - (\text{min-filt-eda}_1 \leq \text{abs}(v) \leq \text{max-filt-eda}_1) \]

for: \( \text{is-prc:arithm} \& \& \text{is-REAL-s-mode}(\text{eda}) \& \text{is-real-val}(v) \)

(29) \[ \text{is-str-size-cond}(\text{eda}, v) = \]
\[ \text{s-length}(\text{eda}) < \text{length}(v) \]

for: \( \text{is-string}(\text{eda}) \& (\text{is-char-val-list} \vee \text{is-bit-val-list})(v) \)

(30) \[ \text{adjust-val}(\text{eda}, v) = \]
\[ \text{is-REAL-s-mode}(\text{eda}) \rightarrow \text{real}(v) \]
\[ \text{is-string}(\text{eda}) \rightarrow \text{adjust-string}(\text{s-base}(\text{eda}), \text{result-length}(\text{eda}, v), v) \]
\[ T \rightarrow v \]

(31) \[ \text{result-length}(\text{eda}, v) = \]
\[ \text{is-ses-varying}(\text{eda}) \rightarrow \text{s-length}(\text{eda}) \]
\[ T \rightarrow \text{length}(v) \]

for: \( \text{is-string}(\text{eda}) \& (\text{is-char-val-list} \vee \text{is-bit-val-list})(v) \)
9.1.4 OPERANDS

The first formula defines an operand as an object that has two parts, an evaluated aggregate attribute and a value representation. The rest is an auxiliary instruction for producing operands from their parts, two functions for transforming an operand into its value or conversely, and two functions and two predicates producing and characterizing integer operands.

(34) \text{is-op} =
\langle s\text{-eva}:\text{is-ev}\text{a}, \\
\langle e\text{-vr}:\text{is-vr} \rangle \rangle

Ref.: is-vr 3-15(32)

Note: Actually only operands will be produced whose eda-parts satisfy the predicate is-correct-ed\text{a}.

(35) \text{mk-op}(eva, vr) =
P\text{ASS}: \mu_{0} (\langle s\text{-eva}:eva, \langle s\text{-vr}:vr \rangle \rangle)

(36) \text{cp-val}(op) =
\text{value}(eva\text{-cp}_{1}, vr\text{-op}_{1})

for: is-area\text{-s-da}(eva\text{-cp}_{1})

(37) \text{cp-da}(op) =
\text{s-da}\text{-s-eva}(op)

Note: Since rep and not test-rep appears, the function can only be used in cases where no SIZE or STRZ conditions can arise.
(38) \[ \text{val-op}(eda, v) = \]
\[ p_n(\langle s\text{-eva:aggr-scalar}(eda)\rangle, \langle s\text{-vr:rep}(aggr-scalar(eda), \text{adjust-val}(v))\rangle) \]
\[ \text{for:} - \text{is-pic}(eda) \]
\[ \text{Ref.: aggr-scalar 8-23(64)} \]

(39) \[ \text{intg-op}(i) = \]
\[ \text{val-op}(\text{INTG-EDA}, i) \]

(40) \[ \text{bintg-op}(i) = \]
\[ \text{val-op}(\text{BINTG-EDA}, i) \]

(41) \[ \text{is-intg-op}(x) = \]
\[ (\text{is-op} \& \text{is-intg-s-da})(x) \]
\[ \text{Ref.: is-intg 9-11(51)} \]

(42) \[ \text{is-bintg-op}(x) = \]
\[ (\text{is-op} \& \text{is-bintg-op-s-da})(x) \]

9.1.5 OPERATORS

An operator is either an infix operator, or a prefix operator, or the object
\text{C-EXP} introduced by the instruction \text{eval-expr-2} to treat a special case of
exponentiation.

(43) \[ \text{is-operator} = \]
\[ \text{is-infix-opr} \lor \text{is-prefix-opr} \lor \text{is-C-EXP} \]

(44) \[ \text{is-arithmetic-opr} = \]
\[ \text{is-ADD} \lor \text{is-SUBTR} \lor \text{is-MULT} \lor \text{is-DIV} \lor \text{is-EXP} \lor \text{is-C-EXP} \lor \text{is-PLUS} \lor \text{is-MINUS} \]

(45) \[ \text{is-ccomp-opr} = \]
\[ \text{is-GT} \lor \text{is-GE} \lor \text{is-EQ} \lor \text{is-LE} \lor \text{is-LT} \lor \text{is-NE} \]

(46) \[ \text{is-bit-opr} = \]
\[ \text{is-AND} \lor \text{is-OR} \lor \text{is-NOT} \]

9.2 AUXILIARY DEFINITIONS ON DATA ATTRIBUTES

The definitions given in this section are used in chapter 9 and in other
chapters. In 9.2.1, integer data attributes, functions producing integer or
string data attributes of given precision, scale or length, and a few incomplete
data attributes are defined. The functions listed in 9.2.2 yield implementation-defined limits associated with arithmetic data attributes.

9.2.1 SPECIAL DATA ATTRIBUTES AND CLASSES OF DATA ATTRIBUTES

\[(47) \textit{intg-eda}(n) = \mu_0(<\textit{mode:REAL},<\textit{base:DEC},<\textit{scale:FIX},<\textit{prec:n},<\textit{scale-f:0}>)\]

\[(48) \textit{bintg-eda}(n) = \mu_0(<\textit{mode:REAL},<\textit{base:BIN},<\textit{scale:FIX},<\textit{prec:n},<\textit{scale-f:0}>)\]

\[(49) \textit{INTG-EDA} = \textit{intg-eda}(<\textit{DEF-PREC-DEC})\]

\[(50) \textit{BINTG-EDA} = \textit{bintg-eda}(<\textit{DEF-PREC-EIN})\]

Note: The elementary objects \textit{DEF-PREC-DEC} and \textit{DEF-PREC-BIN} are implementation-defined integers, the default precisions for decimal or binary arithmetic data attributes.

\[(51) \textit{is-intg}(eda) = (\exists n) (eda = \textit{intg-eda}(n))\]

\[(52) \textit{is-bintg}(eda) = (\exists n) (eda = \textit{bintg-eda}(n))\]

\[(53) \textit{AR-EDA} = \mu_0(<\textit{mode:*},<\textit{base:*},<\textit{scale:*},<\textit{prec:*}>)\]

\[(54) \textit{FLT-EDA} = \mu(\textit{AR-EDA};<\textit{scale:FLT}>)\]

\[(55) \textit{STRING-EDA} = \mu_0(<\textit{base:*},<\textit{length:*},<\textit{varying:*}>)\]

\[(56) \textit{CHAR-EDA} = \mu(\textit{STRING-EDA};<\textit{base:CHAR})\]

\[(57) \textit{BIT-EDA} = \mu(\textit{STRING-EDA};<\textit{base:BIT})\]

9. DATA, OPERATIONS AND CONVERSIONS
Note: The last four definitions introduce special "incomplete data attributes" (cf. 9.5.3).

(58) char-eda(n) =
\[ \mu_0(\langle s\text{-base:CHAR}, s\text{-length:n} \rangle) \]

(59) bit-eda(n) =
\[ \mu_0(\langle s\text{-base:BIT}, s\text{-length:n} \rangle) \]

(60) char-eda-var(n) =
\[ \mu(\text{STRING-EDA}; \langle s\text{-base:CHAR}, s\text{-length:n}, s\text{-varying:VAR} \rangle) \]

9.2.2 IMPLEMENTATION-DEFINED LIMITS ASSOCIATED WITH ARITHMETIC DATA ATTRIBUTES

(61) max-prec(eda) =
\[ \text{max-prec-1}(s\text{-base}(eda), s\text{-scale}(eda)) \]
for: is-prop-arithm(eda)

Note: The definition expresses that max-prec(eda), the maximum precision associated with eda, depends only on the base and scale of eda. The four possible values of max-prec-1 are implementation-defined integers.

(62) max-flt(eda) =
\[ \text{max-flt-1}(s\text{-base}(eda)) \]
for: (is-prop-arithm & is-FLT\*s-scale) (eda)

Note: max-flt-1 yields an implementation-defined numeric value, an upper limit for the magnitude of numeric values that can be exactly represented with eda.

(63) min-flt(eda) =
\[ \text{min-flt-1}(s\text{-base}(eda)) \]
for: (is-prop-arithm & is-FLT\*s-scale) (eda)

Note: min-flt-1 yields an implementation-defined numeric value, a lower limit for the magnitude of numeric values that can be exactly represented with eda.

9.3 EVALUATION OF INFIX EXPRESSIONS

The instruction eval-infix-expr has as arguments the two evaluated operands and the operator of the infix expression; in the case of exponentiation with constant second argument, the original operator EXP will have been replaced by C-EXP (cf. eval-exp 8.2.3).

12 9. DATA, OPERATIONS AND CONVERSIONS
First, the operands are converted to the type and density required by the operator. Then, the result operand is computed.

The value representation part of the result operand is obtained via a result value. The instruction `infix-val` which computes this result value is defined in 9.3.1, the functions `eda-infix` and `eda-c-exp` which compute the result attributes are defined in 9.3.2.

\[(\text{eval-infix-exp}(\text{op-1}, \text{op-2}, \text{opr})) = \]

\[
\text{infix-cx}(c-\text{op-1}, c-\text{op-2}, \text{opr});
\]

\[
c-\text{op-1} = \text{convert-1}(\text{aggr-scalar}(\text{eda-tg}_1), \text{op-1}),
\]

\[
c-\text{op-2} = \text{convert-1}(\text{aggr-scalar}(\text{eda-tg}_2), \text{op-2}).
\]

where:

\[
\begin{align*}
\text{eda-tg}_1 &= \text{is-c-exp}(\text{opr}) \to \text{c-exp-target-1}(\text{mk-arith}(\text{eda-op}_1), \text{eda-cp}_2, \text{v-op}_2) \\
T &= \text{target-1}(\text{eda-op}_1, \text{eda-op}_2, \text{opr})
\end{align*}
\]

\[
\text{eda-tg}_2 = \text{target-2}(\text{eda-op}_1, \text{eda-op}_2, \text{opr}).
\]

Ref.: `convert-1` 9-29(119)

`agg-scalar` 8-23(64)

`mk-arith` 9-39(156)

\[(\text{infix-op}(\text{op-1}, \text{op-2}, \text{opr})) = \]

\[
\text{pass-val-op}(\text{eda-res}_1, \text{v});
\]

\[
infix-val(\text{eda-op}_1, \text{eda-op}_2, \text{v-op}_1, \text{v-op}_2, \text{opr}).
\]

where:

\[
\begin{align*}
\text{eda-res}_1 &= (\text{is-c-exp}(\text{opr}) \to \text{res-c-exp-eda}(\text{eda-op}_1, \text{eda-op}_2, \text{v-op}_2), \\
T &= \text{res-eda}(\text{eda-op}_1, \text{eda-op}_2, \text{opr})
\end{align*}
\]

Ref.: `eval-op` 8-9(32)

9.3.1 VALUE OF THE RESULT

The instruction `infix-val` computes the result value from the values of the converted operands, the data attributes of the converted operands, and the operator. The instruction makes case-distinctions according to the type of the operator. For string operators, comparison of strings, labels and files, and certain cases of fixed arithmetic operations and comparisons, the result can be defined exactly. For the remaining arithmetic cases and for pointer comparison, the result is implementation-dependent.
ABSTRACT SYNTAX AND INTERPRETATION OF PL/I

30 April 1969

(66) \( \text{infix-val}(\text{eda}-1, \text{eda}-2, v-1, v-2, \text{opr}) = \)

\[ \begin{align*}
\text{is-arithm-opr}(\text{opr}) & \rightarrow \\
\text{test-fl}(\text{eda}-\text{res}_1, v); \\
\text{v} & \text{:\ test-infix-num}(\text{eda}-1, \text{eda}-2, v-1, v-2, \text{opr}) \\
\text{is-comp-opr}(\text{opr}) & \rightarrow \text{PASS:truth-to-bit}(\text{is-true-comp}(\text{eda}-1, \text{eda}-2, v-1, v-2, \text{opr}, \text{DN})) \\
\text{is-bit-opr}(\text{opr}) & \rightarrow \\
\text{PASS:infix-bit}(\text{adjust-string}(\text{BIT}, \text{lgth-max}_1, v-1), \\
\text{adjust-string}(\text{BIT}, \text{lgth-max}_1, v-2), \text{opr}) \\
\text{is-CAT}(\text{opr}) & \rightarrow \text{PASS}:v-1-v-2
\end{align*} \]

where:
\[ \begin{align*}
\text{eda}-\text{res}_1 & = (\text{is-C-BXP}(\text{opr}) \rightarrow \text{res-C-exp-eda}(\text{eda}-1, \text{eda}-2, v-2), \\
T & \rightarrow \text{res-eda}(\text{eda}-1, \text{eda}-2, \text{opr})) \\
\text{lgth-max}_1 & = \text{max}(\text{length}(v-1), \text{length}(v-2))
\end{align*} \]

Ref. : \( \text{is-arithm-opr} \ 9-10(44) \)
\( \text{is-comp-opr} \ 9-10(45) \)
\( \text{is-bit-opr} \ 9-10(46) \)
\( \text{adjust-string} \ 9-8(32) \)

9.3.1.1 Arithmetic operations

The instruction \( \text{test-infix-num} \) tests for certain condition situations and then applies \( \text{infix-num} \) to calculate the result. The instruction \( \text{test-fl} \) tests for overflow and underflow.

Metavariables

\( \begin{align*}
\text{eda}, & \text{ is-prop-arith} \\
\text{eda}-1, \text{eda}-2 & \\
v-1, v-2, v & \text{ is-sum-val} \\
\text{opr} & \text{ is-arithm-opr}
\end{align*} \)

(67) \( \text{test-infix-num}(\text{eda}-1, \text{eda}-2, v-1, v-2, \text{opr}) = \)

\[ \begin{align*}
\text{is-0}(\text{cond-inf}_1) & \rightarrow \text{PASS:infix-num}(\text{eda}-1, \text{eda}-2, v-1, v-2, \text{opr}) \\
T & \rightarrow \text{call-cond}(\text{cond-inf}_1)
\end{align*} \]

where:
\[ \text{cond-inf}_1 = \text{cond-infix}(\text{eda}-1, \text{eda}-2, v-1, v-2, \text{opr}) \]

Ref. : \( \text{call-cond} \ 10-18(54) \)

9. DATA, OPERATIONS AND CONVERSIONS
(69) \[ \text{cond-infix}(eda-1, eda-2, v-1, v-2, opr) = \]
\[ \begin{align*}
\text{is-DIV}(opr) & \land v-2 = 0 \rightarrow \text{ZDIV} \\
\text{(is-EXP} & \land \text{is-C-EXP})(opr) \land \\
(v-1 = 0 & \land \text{imag}(v-2) = 0 & \text{real}(v-2) > 0) \lor \\
\text{(is-BAL\_s-mode}(eda-1) & v-1 < 0 \land \\
\text{is-intg} \lor \text{is-bal\_y}(eda-2)) \rightarrow \text{ERROR} \\
T \rightarrow N
\end{align*} \]

Ref.: \text{is-intg} 9-11(51) \\
\text{is-bal\_y} 9-11(52)

(69) \[ \text{infix-num}(eda-1, eda-2, v-1, v-2, opr) = \]

Note: This function is characterized by axiom (70) which follows.

(70) \[ \text{is-n}(\text{cond-infix}) \land \]
\[ \begin{align*}
\text{(is-FIX\_s-scale}(eda-1) & \land \text{is-FIX\_s-scale}(eda-2) \land \\
\text{is-DIV}(opr) \land \\
v-1 \in v-0\text{-set}(eda-1) \land v-2 \in v-0\text{-set}(eda-2) \lor \\
(v-1 = 0 \land v-2 = 0) \Rightarrow \\
\text{infix-num}(eda-1, eda-2, v-1, v-2, opr) = \text{acc-infix-num}(v-1, v-2, opr)
\end{align*} \]

Ref.: \text{v-0\text{-set}} 9-5(17)

Note: This axiom defines infix-num for the FIXED case (except division), provided that the argument values are exactly representable with their data attributes, and for certain limit cases of exponentiation. For all other cases, infix-num is implementation-defined (and is assumed to be an approximation of the exact arithmetic operation; this approximation may depend also on the data attributes).

(71) \[ \text{acc-infix-num}(v-1, v-2, opr) = \]
\[ \begin{align*}
\text{is-ADD}(opr) & \rightarrow v-1 + v-2 \\
\text{is-SUBTR}(opr) & \rightarrow v-1 - v-2 \\
\text{is-MULT}(opr) & \rightarrow v-1 \cdot v-2 \\
\text{(is-EXP} & \land \text{is-C-EXP})(opr) & \rightarrow v-1 \uparrow v-2
\end{align*} \]
ABSTRACT SYNTAX AND INTERPRETATION OF PL/I

(72) \[ \text{test-fl}(eda,v) = \]
\begin{align*}
& \text{is-CPLX-s-mode}(eda) \rightarrow \]
& \text{pass-cplx}(v_{-re},v_{-im}) ; \]
& v_{-re} : \text{test-fl}(eda_{-real}, \text{real}(v)) , \]
& v_{-im} : \text{test-fl}(eda_{-real}, \text{imag}(v)) \]
& \text{is-FIX-s-scale}(eda) \& \text{is-ofl-cond}(eda,v) \rightarrow \text{call-cond}(FOLF) \]
& \text{is-FLT-s-scale}(eda) \& \text{is-ofl-cond}(eda,v) \rightarrow \text{call-cond}(OFL) \]
& \text{is-FLT-s-scale}(eda) \& \text{is-ufl-cond}(eda,v) \rightarrow \]
& \text{pass}(0) ; \]
& \text{call-cond}(UFL) \]
& T \rightarrow \text{PASS} : v \]
\end{align*}

Ref.: call-cond 10-18(54)

(73) \[ \text{is-ofl-cond}(eda,v) = \]
\begin{align*}
& \text{abs}(v \cdot \text{base-eda}_1 + q \cdot da_1) \geq \text{base-eda}_1 + w \cdot eda_1 \]
\end{align*}

for: (is-REAL-s-mode \& is-FIX-s-scale)(eda) \& is-real-val(v)

Ref.: is-real-val 9-3(4)

(74) \[ \text{is-ofl-cond}(eda,v) = \]
\begin{align*}
& \text{abs}(v) \geq \text{max-flt-eda}_1 \]
\end{align*}

for: (is-REAL-s-mode \& is-FLT-s-scale)(eda) \& is-real-val(v)

Ref.: is-real-val 9-3(4)

(75) \[ \text{is-ufl-cond}(eda,v) = \]
\begin{align*}
& 0 < \text{abs}(v) < \text{min-flt-eda}_1 \]
\end{align*}

for: (is-REAL-s-mode \& is-FLT-s-scale)(eda) \& is-real-val(v)

Ref.: is-real-val 9-3(4)

9.3.1.2 Comparison operations

First, the interpretation of the operator \# is reduced to the interpretation of EQ, and the interpretation of GE, LE, LT to the interpretation of GT. Then, a case distinction according to the type of attributes is made. The resulting truth value T or F is transformed into bit string of length one.
ABSTRACT SYNTAX AND INTERPRETATION OF PL/I

9.3.1.2.1 Numeric comparison

Metavariables

eda-1, eda-2 is-prop-arithm
v-1, v-2 is-num-val
opr is-EQ \lor is-GT
(78) \( \text{is-true-num-comp(eda-1, eda-2, v-1, v-2, opr)} = \)

Note: This function is characterized by axiom (79) which follows.

(79) \( \text{is-FIX\text{-}scale(eda-1)} \land \text{is-FIX\text{-}scale(eda-2)} \land v-1 \in v\text{-}0\text{-}set(eda-1) \land v-2 \in v\text{-}0\text{-}set(eda-2) \Rightarrow \)
\( \text{is-true-num-comp(eda-1, eda-2, v-1, v-2, opr)} = \text{is-true-acc\text{-}num\text{-}comp(v-1, v-2, opr)} \)

Ref.: v-0\text{-}set 9-5(17)

Note: This axiom defines is-true-num-comp for the FIXED case, provided that the argument values are exactly representable with their data attributes. In all other cases, is-true-num-comp is implementation-defined and is assumed to be an approximation of the exact mathematical comparison operation.

(80) \( \text{is-true-acc\text{-}num\text{-}comp(v-1, v-2, opr)} = \)
\( \text{(is\text{-}EQ(opr) \text{-} v-1 = v-2,} \)
\( \text{is\text{-}GT(opr) \text{-} v-1 > v-2) } \)

9.3.1.2.2 String comparison

Character strings are compared according to an implementation-defined collating sequence, bit strings according to their interpretation as binary integers.

(81) \( \text{is-true\text{-}string\text{-}comp(base, v-1, v-2, opr)} = \)
\( \text{is\text{-}EQ(opr) \text{-} v-1 = v-2} \)
\( \text{is\text{-}<>(v-1) \text{-} F} \)
\( \text{head}(v-1) = \text{head}(v-2) \Rightarrow \text{is-true\text{-}string\text{-}comp(base, tail(v-1), tail(v-2), opr)} \)
\( \text{is\text{-}CHAR(base) \Rightarrow collatehead(v-1) > collatehead(v-2)} \)
\( \text{is\text{-}EIT(base) \Rightarrow single\text{-}bit\text{-}num\text{-}head(v-1) > single\text{-}bit\text{-}num\text{-}head(v-2)} \)

for: \( \text{(is\text{-}CHAR(base) \land is\text{-}char\text{-}val\text{-}list(v-1) \land is\text{-}char\text{-}val\text{-}list(v-2) \land is\text{-}EIT(base) \land is\text{-}bit\text{-}val\text{-}list(v-1) \land is\text{-}bit\text{-}val\text{-}list(v-2) \land length(v-1) = length(v-2) \land (is\text{-}EQ \lor is\text{-}GT)(opr)} \)

Ref.: single-bit-num 9-34(137)
\( \text{is\text{-}char\text{-}val 9-3(6)} \)
\( \text{is\text{-}bit\text{-}val 9-4(11)} \)

(82) \( \text{collate(v)} = \)

for: is\text{-}char\text{-}val(v)

Ref.: is\text{-}char\text{-}val 9-3(6)
Note: The function collat associates with the character value \( v \) an integer \( i \) according to the implementation-defined collating sequence.

9.3.1.2.3 Pointer comparison

Metavariables

\( p-1, p-2 \) is-ptr-val

The predicate

\[ (83) \quad \text{is-equal-ptr}(p-1, p-2) = \]

is characterized by axiom (84), which guarantees an implementation-independent result only if the two pointer values are either identical or independent (cf. 3.4.1).

\[ (84) \quad (p-1 = p-2 \Rightarrow \text{is-equal-ptr}(p-1, p-2)) \land \\
    (p-1 \neq p-2 \land \text{is-NPTR}(p-1) \lor \text{is-NPTR}(p-2) \lor \text{is-indep}(p-1, p-2)) \Rightarrow \\
    \neg\text{is-equal-ptr}(p-1, p-2) \]

Ref.: is-indep 3-16(43)

9.3.1.3 Bit string operations

\[ (85) \quad \text{infix-bit}(v-1, v-2, opr) = \]

\[ \text{length}(v-1) \]

\[ \text{LIST} \quad \text{single-infix-bit}(\text{elem}(i, v-1), \text{elem}(i, v-2), opr) \]

\[ \text{for:is-bit-val-list}(v-1) \land \text{is-bit-val-list}(v-2) \land \text{length}(v-1) = \text{length}(v-2) \land \]

\[ \text{is-bit-cpr}(opr) \]

Ref.: is-bit-val 9-8(11)

is-bit-cpr 9-10(46)
9.3.2 DATA ATTRIBUTES OF THE RESULT

On the one hand, the data attributes of the result of an infix expression are defined by the function eda-infix. This function is a sub-function of the function aggr-infix-expr and treats the scalar case; the latter function is a subfunction of the instruction aggr-expr (cf. 8.2.5) which is used to determine the attributes of an expression independent of its value. On the other hand, only the subfunctions target-1, target-2 and res-eda of eda-infix are used in the evaluation of infix expressions. The functions target-1 and target-2 calculate the target attributes to which the operands have to be converted and are defined in 9.3.2.1. The function res-eda calculates the result attribute from the attributes of the converted operands and is defined in 9.3.2.2.

The case of the operator C-EXP (exponentiation with an integer constant as second operand) needs a special treatment because the result attribute depends also on the value of the second operand. The corresponding function eda-c-expr, its sub-functions c-exp-target-1 and res-c-exp-eda, and the predicate is-c-exponent which is used by eval-expr-2 to test for this situation (cf. 8.2.3) is defined in 9.3.2.3.

9.3.2.1 Target attributes for conversion of operands

Except for some cases of exponentiation, the target attributes for the first and second operand are the same. Precision or length of arithmetic or string target attributes respectively, are set to *, because they will be determined by the convert-1-instruction from the source attributes eva-1, eva-2.

Ref.: complete-eva 9-37(145)
target-1(eda-1, eda-2, opr) =
    is-EXP(opr) \rightarrow \mu(\text{target}(eda-1, eda-2, opr);<\text{scale:FLT}>)
    T \rightarrow \text{target}(eda-1, eda-2, opr)

(89) target-2(eda-1, eda-2, opr) =
    (is-EXP v is-C-EXP)(opr) \rightarrow
    ((is-intg v is-bintg)(eda-2) \rightarrow
        \mu(eda-2;<\text{base:pref-base}(eda-1, eda-2),<\text{prec:>,<\text{scale-f:}>})
        T \rightarrow \mu(\text{target}(eda-1, eda-2, opr);<\text{scale:FLT}>)
    )
    T \rightarrow \text{target}(eda-1, eda-2, opr)

Ref.: is-intg 9-11(51)
      is-bintg 9-11(52)

(90) target(eda-1, eda-2, opr) =
    is-arithm-opr(opr) v is-comp-opr(opr) \&
    (is-num-type(eda-1) v is-num-type(eda-2)) \rightarrow
    pref-ar-eda(mk-arithm(eda-1), mk-arithm(eda-2))
    is-comp-opr(opr) \& (is-string-type(eda-1) v is-string-type(eda-2)) \rightarrow
    pref-str-eda(mk-string(eda-1), mk-string(eda-2))
    is-comp-opr(opr) \rightarrow eda-1
    is-CAT(opr) \& is-BITEs-base(eda-1) \& is-BITEs-base(eda-2) \rightarrow
    pref-str-eda(eda-1, eda-2)
    is-CAT(opr) \rightarrow pref-str-eda(mk-char-string(eda-1), mk-char-string(eda-2))
    is-bit-opr(opr) \rightarrow EIT-EDA

Ref.: is-arithm-opr 9-10(44)
      is-comp-opr 9-10(45)
      mk-arithm 9-39(156)
      mk-string 9-39(157)
      mk-char-string 9-39(158)
      is-bit-opr 9-10(46)

(91) is-num-type =
    is-arith \lor is-num-pic

Ref.: is-num-pic 9-45(173)

(92) is-string-type =
    is-string \lor is-char-pic
9.3.2.1.1 Higher arithmetic characteristics

Metavariable

eda-1, eda-2 is-prop-arith

(93) pref-ar-eda(eda-1, eda-2) =

\[ \mu_0 (\text{s-mode: pref-mode(eda-1, eda-2)}, \text{s-base: pref-base(eda-1, eda-2)}, \text{s-scale: pref-scale(eda-1, eda-2)}, \text{s-prec:*}) \]

(94) pref-mode(eda-1, eda-2) =

is-REAL*mode(eda-1) & is-REAL*mode(eda-2) \rightarrow \text{REAL}

T \rightarrow \text{CPLX}

(95) pref-base(eda-1, eda-2) =

is-DEC*base(eda-1) & is-DEC*base(eda-2) \rightarrow \text{DEC}

T \rightarrow \text{BIN}

(96) pref-scale(eda-1, eda-2) =

is-FIX*scale(eda-1) & is-FIX*scale(eda-2) \rightarrow \text{FIX}

T \rightarrow \text{FLT}

9.3.2.1.2 Higher string characteristics

Metavariable

eda-1, eda-2 is-string

(97) pref-str-eda(eda-1, eda-2) =

\[ \mu_0 (\text{s-base: pref-str-base(eda-1, eda-2)}, \text{s-varying: pref-varying(eda-1, eda-2)}, \text{s-length:*}) \]

(98) pref-str-base(eda-1, eda-2) =

is-BIT*base(eda-1) & is-BIT*base(eda-2) \rightarrow \text{BIT}

T \rightarrow \text{CHAR}

22 9. DATA, OPERATIONS AND CONVERSIONS
(99) \[ \text{pref-varying}(eda-1, eda-2) = \]
\[ \text{is-\#s-varying}(eda-1) \land \text{is-\#s-varying}(eda-2) \land T \leftrightarrow \text{VAR} \]

9.3.2.2 Result attributes

For comparison operations, the data attribute of the result is bit string of length one, for the other operations, it is the attribute of the converted first operand with the appropriate result precision or length inserted.

If the function \text{res-eda} is used for determining aggregate attributes of expressions before the actual evaluation of expressions (cf. \text{aggr-expr 8.2.5}) the length will be \#.

Metavariable

\( \text{lgth-1, lgth-2} \) is-extent

Abbreviations

For \text{is-prop-arithm}(eda-1) \land \text{is-prop-arithm}(eda-2):

- \( M_1 = \text{max-prec}(eda-1) \) maximum precision permitted for given base and scale
- \( P_1 = \text{s-prec}(eda-1) \) precision of converted first operand
- \( Q_1 = \text{s-scale-f}(eda-1) \) scale-factor of converted first operand
- \( P_2 = \text{s-prec}(eda-2) \) precision of converted second operand
- \( Q_2 = \text{s-scale-f}(eda-2) \) scale-factor of converted second operand

For \text{is-string}(eda-1) \land \text{is-string}(eda-2):

- \( \text{lgth}_1 = \text{s-length}(eda-1) \) (maximum) length of converted first operand
- \( \text{lgth}_2 = \text{s-length}(eda-2) \) (maximum) length of converted second operand
(100) res-eda(eda-1,eda-2,opr) =
    is-arith-opr(opr) —>
    (is-EXP(opr) & is-FIXes-scale(eda-2) —> eda-1, is-FLY*es-scale(eda-1) —> μ(eda-1;<s-prec:max(p1,p2)>, is-FIXes-scale(eda-1) —>
    μ(eda-1;<s-prec:min(N1, res-prec(eda-1,eda-2,opr))>, <s-scale-f:res-scale-f(eda-1,eda-2,opr)>))
    is-comp-opr(opr) —> bit-eda(1)
    is-bit-opr(opr) —> μ(eda-1;<s-length:max-1(lgth1,lgth2)>)
    is-CMT(opr) —> μ(eda-1;<s-length:length-sum(lgth1,lgth2)>)

Ref.: is-arith-opr 9-10(44)
    is-comp-opr 9-10(45)
    bit-eda 9-12(59)
    is-bit-opr 9-10(46)

(101) max-1(lgth-1,lgth-2) =
    is-intg-val(lgth-1) & is-intg-val(lgth-2) —> max(lgth-1,lgth-2)
    T —> *

Ref.: is-intg-val 9-3(5)

(102) length-sum(lgth-1,lgth-2) =
    is-intg-val(lgth-1) & is-intg-val(lgth-2) —> lgth-1 + lgth-2
    T —> *

Ref.: is-intg-val 9-3(5)

(103) res-prec(eda-1,eda-2,opr) =
    (is-ADD v is-SUBST v is-CMT)(opr) —> max(p1, q1, p2, q2) + max(q1, q2) + 1
    is-MULT(opr) —> p1 × p2 + 1
    is-DIV(opr) —> N1

for:is-prop-arithm(eda-1) & is-prop-arithm(eda-2) &
    (is-ADD v is-SUBST v is-MULT v is-DIV)(opr)
9.3.2.3 The special case of integer constant exponentiation

The predicate `is-c-exponent` tests for exponentiation with an integer constant as second operand. In this case, the function `eda-c-exp` is used to calculate the result attribute of the infix expression; this function depends on the value of the second operand. If the converted first operand is not `FIXED`, or if the resulting precision would be too large, the case is treated like the general exponentiation.

Metavariable

\[ t \text{ is-expr} \]

\[ \text{(105)} \quad \text{is-c-exponent}(t) = \]

\[ \text{is-infix-expr}(t) \& \text{is-EXP-opr}(t) \& \text{is-intg-values-ops-op-2}(t) \& \]

\[ 0 \leq s-ops-op-2(t) \& (\text{is-intg} \lor \text{is-bintg})\text{-ops-ops-op-2}(t) \]

Ref.:  
\[ \text{is-intg-val} \ 9-3(5) \]
\[ \text{is-intg} \ 9-11(51) \]
\[ \text{is-bintg} \ 9-11(52) \]

\[ \text{(106)} \quad \text{eda-c-exp}(eda-1, \text{const}) = \]

\[ \text{res-c-exp-eda(complete-tg(eda-tg_1, eda-1), complete-tg(eda-tg_2, eda-const_1),} \]
\[ v-const_2) \]

where:
\[ \text{eda-tg_1} = \text{c-exp-target-1}(\text{mk-arithm}(eda-1), \text{eda-const_1}, v-const_1) \]
\[ \text{eda-tg_2} = \text{target-2}(\text{eda-1, eda-const_1}, \text{C-EXP}) \]
\[ \text{eda-const_1} = \text{s-da}(\text{const}) \]
\[ v-const_1 = \text{s-v}(\text{const}) \]

Ref.:  
\[ \text{complete-tg} \ 9-37(149) \]
\[ \text{mk-arithm} \ 9-39(156) \]
(107) \[ c \cdot \text{exp-target-1}(\text{eda-1}, \text{eda-2}, v-2) = \]
\[ \text{is-FIXcs-scale}(\text{eda-1}) \land 0 < (p_1 + 1) \cdot v-2 - 1 \leq N_1 \implies \text{pref-ar-eda}(\text{eda-1}, \text{eda-2}) \]
\[ T = \text{target-1}(\text{eda-1}, \text{eda-2}, \text{EXP}) \]

where:
\[ N_1 = \max-prec(c \cdot \text{eda}_1) \]
\[ p_1 = s \cdot \text{prec}(c \cdot \text{eda}_1) \]
\[ c \cdot \text{eda}_1 = \text{complete-tg}(\text{pref-ar-eda}(\text{eda-1}, \text{eda-2}), \text{eda-1}) \]

for is-prop-arithmetic(eda-1) \& is-prop-arithmetic(eda-2) \& is-intg-val(v-2)

Ref.: max-prec 9-12(61)
complete-tg 9-37(149)
is-intg-val 9-3(5)

(108) \[ \text{res-}c \cdot \text{exp-eda}(\text{eda-1}, \text{eda-2}, v-2) = \]
\[ \text{is-FIXcs-scale}(\text{eda-1}) \land \text{is-FIXcs-scale}(\text{eda-2}) \implies \]
\[ p(\text{eda-1}; s \cdot \text{prec}; (p_1 + 1) \cdot v-2 - 1) \cdot \text{scale-f_gr} + v-2 \]
\[ T = \text{res-eda}(\text{eda-1}, \text{eda-2}, \text{EXP}) \]

for is-prop-arithmetic(eda-1) \& is-prop-arithmetic(eda-2) \& is-intg-val(v-2)

Ref.: is-intg-val 9-3(5)

2.9 Evaluation of Prefix Expressions

Prefix expressions are evaluated by the instruction \textit{eval-prefix-exp} that has as arguments an operand and an operator and yields as result an operand. The value of the result is calculated by the instruction \textit{prefix-val} defined in 9.4.1, the data attribute of the result by the function \textit{eda-prefix} defined in 9.4.2.

(109) \[
\text{eval-prefix-exp}(\text{op}, \text{opr}) = \]
\[
\text{prefix-op}(c \cdot \text{op}, \text{opr}) ; \]
\[
c \cdot \text{op} \cdot \text{prefix-1}(aggr-scalar(\text{eda-tg}_1), \text{op}) \]

where:
\[ \text{eda-tg}_1 = \text{prefix-target}(c \cdot \text{op}_1, \text{opr}) \]

Ref.: \textit{aggr-scalar} 9-23(64)

(110) \[
\text{prefix-op}(\text{op}, \text{opr}) = \]
\[
\text{pass-val-op}(c \cdot \text{op}_1, v) ; \]
\[
v \cdot \text{prefix-val}(\text{eda-op}_1, v \cdot \text{op}_1, \text{opr}) \]

Ref.: \textit{val-op} 9-9(38)
9.4.1 VALUE OF THE RESULT

In the case of an arithmetic operator, a test for overflow or underflow is made. This is necessary in case of the operator MINUS, because asymmetric representations of negative numbers may be used (cf. the representation of two-complement pictures 9.6.3.2).

(111) \[
\text{prefix-val(eda, v, opr) =}
\]
\[
(\text{is-PLUS } \lor \text{ is-MINUS })(opr) \rightarrow \text{test-fl(eda, prefix-num(eda, v, opr))}
\]
\[
\text{is-NOT(opr)} \rightarrow \text{PASS:prefix-not(v)}
\]
Ref.: test-fl 9-15(72)

9.4.1.1 Arithmetic prefix operations

Metavarsiables

eda \quad \text{is-prop-arithm}

v \quad \text{is-num-val}

opr \quad \text{is-PLUS } \lor \text{ is-MINUS}

(112) \[
\text{prefix-num(eda, v, opr) =}
\]
\[
\text{Note: This function is characterized by axiom (113) which follows.}
\]
(113) \[
(\text{is-PLUS(opr)} \lor \text{is-MINUS(opr)} \land \text{is-FIXED-scale(eda)}) \land v \in \text{-}0\text{-set(eda)} =
\]
\[
\text{prefix-num(eda, v, opr) = acc-prefix-num(v, opr)}
\]
Ref.: \text{-}0\text{-set 9-5(17)}

Note: This axiom gives a definition of prefix-num in case opr is PLUS and v is exactly representable with eda, or opr is MINUS, eda is FIXED and v is exactly representable with eda. In the remaining cases, prefix-num is an implementation-defined approximation of the mathematical operation.

(114) \[
\text{acc-prefix-num(v, opr) =}
\]
\[
(\text{is-PLUS(opr)} \rightarrow v,
\]
\[
\text{is-MINUS(opr)} \rightarrow -v)
\]

9. DATA, OPERATIONS AND CONVERSIONS 27
9.6.1.2 Negation

(115) \[ \text{prefix-not}(v) = \]

\[
= \text{length}(v) \]

\[
- \text{LIST} \text{ single-prefix-not(elem(i,v))} \]

\[
\text{for:} \text{is-bit-val-list}(v) \]

Ref.: \text{is-bit-val} 9-4(11)

(116) \[ \text{single-prefix-not}(v) = \]

\[
\text{is-0-BIT}(v) \rightarrow 1-BIT, \]

\[
\text{is-1-BIT}(v) \rightarrow 0-BIT \]

\[
\text{for:} \text{is-bit-val}(v) \]

Ref.: \text{is-bit-val} 9-4(11)

9.4.2 DATA ATTRIBUTES OF THE RESULT

(117) \( \text{eda-prefix}(eda,opr) = \)

\[
\text{complete-tg(prefix-target(eda,opr),eda)} \]

Ref.: \text{complete-tg} 9-37(149)

(118) \[ \text{prefix-target}(eda,opr) = \]

\[
\text{is-PLUS v is-MINUS}(opr) \rightarrow \text{AE-EDA} \]

\[
\text{is-NOT}(opr) \rightarrow \text{BIT-EDA} \]

9.5 CONVERTER

This chapter describes the instruction \( \text{convert-1(eva-ty,op)} \) which converts an operand op to a target attribute eva-ty. The result of the instruction is again an operand. Except in the case of areas, the conversion is done in three steps: going from a value representation to a value, converting the value and going from the converted value back to a representation. Of these, the first and third step have been described already (cf. 9.1); the second step is performed by the instruction \( \text{value-conv} \) defined in 9.5.1.

The data attribute part of the target attribute eva-ty may be incomplete, and first, it is completed. The syntax of incomplete data attributes and the function \( \text{complete-eva} \) are defined in 9.5.3.

In the case of areas the instruction \( \text{area-conv} \) is invoked (cf. 9.5.2).
Metavariable

ev-tg \{(s-da:is-inc-eda),(s-dens:is-AL v is-UNAL)\}

(119) \textit{convert-1}(\textit{evatg},op) =

\textit{convert-2}(\text{complete-evatg},s-evatg,op),op)

for:\textit{is-inc-eda\#s-da(\textit{evatg})}

(120) \textit{convert-2}(\textit{evatg},op) =

\textit{is-area\#s-da(\textit{evatg}) \rightarrow area-conv(\textit{evatg},op)}

\textbf{T \rightarrow}

\textit{mk-op}(\textit{evatg},vr);

\textit{vr}:\textit{test-rep}(\textit{evatg},v);

v:\textit{value-conv}(s-da(\textit{evatg}),eda-op,op-val(op))

Ref.: \textit{mk-op} 9-9(35)

\textit{test-rep} 9-7(27)

\textit{op-val} 9-9(36)

9.5.1 \textbf{VALUE CONVERSION}

If eda-tg and eda are of the same type, the result of \textit{value-conv}(eda-tg,eda,v)
is the unchanged value v. Otherwise, the conversion must be between the types
numeric, character, and bit, and different instructions or functions are invoked
according to the six possible combinations. These instructions or functions
depend only on v, with the following two exceptions:

(a) in conversion from numeric to string, the source attribute is needed,

(b) in conversion from character to bit, the target attribute is needed,
because only as many characters as necessary are converted (and hence can
raise the \texttt{CONVERSION} condition).
### 9.5.1.1 Numeric to Character Conversion

If the source attribute is numeric picture, then this is essentially the operation of representing a numeric value in pictured form (cf. 9.6.3.5). If the source attribute is arithmetic, then again a picture attribute is constructed from the arithmetic attribute, but this case differs from the ordinary picture case with respect to the treatment of complex values and of scale factors.

(123) \text{num-char-conv}(eda,v) =

\begin{align*}
(\text{is-arithm} & \& \text{is-CPLX-s-mode})(eda) \rightarrow \\
\text{pass-cplx-string}(v-re,v-im); & \\
v-re: \text{real-char-conv}(eda-real_1, \text{real}(v)), & \\
v-im: \text{real-char-conv}(eda-real_1, \text{imag}(v)) & \\
\text{is-BEAL-s-mode}(eda) \rightarrow \text{real-char-conv}(eda,v) & \\
\text{is-num-pics}(eda) \rightarrow \text{num-pic-char-conv}(eda,v) & \\
\text{for:is-num-type}(eda) & \& \text{is-num-val}(v)
\end{align*}

Ref.:  
\begin{align*}
\text{is-num-pic} & 9-45(173) \\
\text{num-pic-char-conv} & 9-50(236) \\
\text{is-num-type} & 9-21(91) \\
\text{is-num-val} & 9-3(3)
\end{align*}
(124) \[
\text{cplx-string}(v-\text{re},v-\text{im}) = v-\text{re}^\text{sign-and-left-adjust}(v-\text{im}^\langle 1\text{-CHAR} \rangle)
\]
for: is-char-val-list(v-re) & is-char-val-list(v-im)

Ref.: is-char-val 9-3(6)

(125) \[
\text{sign-and-left-adjust}(v) =
\begin{align*}
\text{is-ELANK} & \text{elem}(1,v) & \text{is-ELANK} & \text{is-PLUS} & \text{is-MINUS} & \text{is-ELANK}(2,v) \\
\text{is-ELANK} & \text{elem}(1,v) & \text{is-PLUS} & \text{tail}(v) & \text{sign-and-left-adjust}(\text{tail}(v)^{< 1\text{-CHAR}})
\end{align*}
\]

T \rightarrow v
for: is-char-val-list(v)

Ref.: is-char-val 9-3(6)

Note: The function moves leading blanks to the end and, if no sign was present, signs the string with PLUS.

(126) \[
\text{real-char-conv}(\text{eda},v) =
\begin{align*}
\text{is-ELANK} & \text{base}(\text{eda}) \Rightarrow \text{real-char-conv}(\text{complete-tag}(\mu(\text{ARM-FDA};\langle \text{base};\text{DEC} \rangle),\text{eda}),v) \\
\text{is-FIX} & \text{is-scale}(\text{eda}) \Rightarrow 0 \leq q-\text{eda} \leq p-\text{eda} \\
\text{NUM-pic-char-conv}(\text{out-fix-pic}(p-\text{eda} + 3,q-\text{eda}),v) \\
\text{is-INT} & \text{is-scale}(\text{eda}) \Rightarrow \text{BASE-CONV-scale-f}(v-\text{str},q-\text{eda}) \\
\text{v-str} & \text{NUM-pic-char-conv}(\text{out-fix-pic}(p-\text{eda} + 1,0),v) + 10 \times q-\text{eda} \\
\text{is-INT} & \text{is-scale}(\text{eda}) \Rightarrow \text{NUM-pic-char-conv}(\text{out-flt-pic}(q-\text{eda} + 2),v)
\end{align*}
\]
for: (is-real-arith & is-FLOAT-mode)(\text{eda}) & is-real-val(v)

Ref.: NUM-pic-char-conv 9-60(236)
out-fix-pic 9-60(237)
out-flt-pic 9-60(239)
is-real-val 9-3(4)
(127) \( 	ext{conv-scale-f}(v, m) = v < f - \text{CHAR} > < \text{sign-char}(-m) > \text{num-char-list}(\text{lgth-n}_i, \text{abs}(m)) \)

where:
\( \text{lgth-n}_i = (\text{li})(10^i + i) \leq \text{abs}(m) < 10^i \)

for: is-char-val-list(v)

Ref.: sign-char 9-5(212)
      is-char-val 9-3(6)

(128) \( \text{num-char-list}(n, v) = \)

\( n = 0 \rightarrow <> \)
\( T \rightarrow \text{num-char-list}(n - 1, \text{trunc}(v / 10)) < \text{single-num-char}(\text{mod}(v, 10)) > \)

for: is-intg-val(v) & v \geq 0

Ref.: is-intg-val 9-3(5)

(129) \( \text{single-num-char}(v) = \)

\( v = 0 \rightarrow 0\text{-CHAR} \)
\( v = 1 \rightarrow 1\text{-CHAR} \)
\( v = 2 \rightarrow 2\text{-CHAR} \)
\( v = 3 \rightarrow 3\text{-CHAR} \)
\( v = 4 \rightarrow 4\text{-CHAR} \)
\( v = 5 \rightarrow 5\text{-CHAR} \)
\( v = 6 \rightarrow 6\text{-CHAR} \)
\( v = 7 \rightarrow 7\text{-CHAR} \)
\( v = 8 \rightarrow 8\text{-CHAR} \)
\( v = 9 \rightarrow 9\text{-CHAR} \)

for: is-intg-val(v) & 0 \leq v \leq 9

Ref.: is-intg-val 9-3(5)

9.5.1.2 Numeric to bit conversion

The numeric value is first converted to binary integer of a precision that is deduced from the source data attribute. Then, the absolute value of the binary integer is interpreted as a bit string.
30 April 1969

ABSTRACT SYNTAX AND INTERPRETATION OF PL/I

(130) \texttt{num-bit-conv}(eda,v) =
\begin{align*}
\text{lghth-eda} & \leq 0 \rightarrow \text{PASS}::<>
\end{align*}
\begin{align*}
\tau \rightarrow \\
\text{bintg-bit-conv(lghth-eda, vr)} ; \\
v r \text{-test-rep(bintg-eda(lghth-eda), v)}
\end{align*}

where:
\begin{align*}
lghth-eda &= \text{bit-length(}\text{mk-arithm(eda)}\text{)}
\end{align*}
for:is-num-type(eda) \& is-num-val(v)

Ref.:
\begin{align*}
\text{test-rep} 9-7(27) \\
\text{bintg-eda} 9-11(48) \\
is-num-type 9-21(91) \\
is-num-val 9-3(3)
\end{align*}

(131) \texttt{bintg-bit-conv}(n,vr) =
\begin{align*}
\text{PASS}:\text{num-bit-list}(n, \text{ahs}=\text{value(bintg-eda(n), vr)})
\end{align*}

Ref.:
\begin{align*}
\text{value} 9-7(26) \\
\text{bintg-eda} 9-11(48)
\end{align*}

(132) \texttt{num-bit-list}(n,v) =
\begin{align*}
n &= 0 \rightarrow <> \\
\tau \rightarrow \text{num-bit-list}(n - 1, \text{trunc}(v / 2)) \langle \text{single-num-bit(modulo(v, 2))} \rangle
\end{align*}

for:is-real-val(v) \& n \geq 0

Ref.:
\begin{align*}
is-real-val 9-3(4)
\end{align*}

(133) \texttt{single-num-bit}(v) =
\begin{align*}
\langle v = 0 \rightarrow 0\text{-BIT}, \\
v = 1 \rightarrow 1\text{-BIT} \rangle
\end{align*}

for:v = 0 \lor v = 1

9.5.1.3 Character to numeric conversion

The instruction \texttt{char-num-conv}(v) is defined in chapter 11, because it uses methods for parsing strings that are developed there (cf. 11.6.1).

9.5.1.4 Character to bit conversion

The character string is parsed from left to right. If a character is found that cannot be converted then the CONVERSION condition is raised. On return from the on-unit, conversion is retried with the corrected string.


\[ \text{char-bit-conv}(\text{eda-tg}, v) = \]

\[ \begin{align*}
&\text{for:} \quad \text{(is-string} \land \text{is-BIT-s-base)}(\text{eda-tg}) \land \text{is-char-val-list}(v) \\
&\text{Ref.:} \quad \text{call-conv-cond 10-24(70)} \quad \text{is-char-val 9-3(6)}
\end{align*} \]

\[ \text{(135) single-char-bit}(v) = \]

\[ \begin{align*}
&(\text{is-0-CHAR}(v) \rightarrow 0\text{-BIT},) \\
&(\text{is-1-CHAR}(v) \rightarrow 1\text{-BIT}) \\
&\text{for:} \quad (\text{is-0-CHAR} \lor \text{is-1-CHAR}) (v)
\end{align*} \]

\[ 9.5.1.5 \text{ Bit to numeric conversion} \]

\[ \text{(136) bit-num-conv}(v) = \]

\[ \begin{align*}
&\sum_{i=1}^{\text{length}(v)} \text{single-bit-num}(\text{elem}(i, v)) \cdot 2^{(\text{length}(v) - i)} \\
&\text{for:} \quad \text{is-bit-val-list}(v)
\end{align*} \]

\[ \text{Ref.:} \quad \text{is-bit-val 9-4(11)} \]

\[ \text{(137) single-bit-num}(v) = \]

\[ \begin{align*}
&(\text{is-0-BIT}(v) \rightarrow 0, ) \\
&(\text{is-1-BIT}(v) \rightarrow 1) \\
&\text{for:} \quad \text{is-bit-val}(v)
\end{align*} \]

\[ \text{Ref.:} \quad \text{is-bit-val 9-4(11)} \]
9.5.1.6 Bit to character conversion

\( \text{bit-char-conv}(v) = \)

\[
\text{length}(v) \\
\text{LIST} \ 	ext{single-bit-char} (\text{elem}(i,v)) \\
i=1
\]

for:is-bit-val-list(v)

Ref.: is-bit-val 9-4(11)

\( \text{single-bit-char}(v) = \)

\[
\{ \text{is-0-BIT}(v) \rightarrow 0\text{-CHAR}, \\
\text{is-1-BIT}(v) \rightarrow 1\text{-CHAR} \}
\]

for:is-bit-val(v)

Ref.: is-bit-val 9-4(11)

9.5.2 AREA CONVERSION

Area conversion is accomplished by the instruction \( \text{area-conv}(\text{eva}, \text{op}) \), where \( \text{op} \) is the operand to be converted and \( \text{eva} \) is the target area attribute. An area operand is constructed whose \( \text{vr} \)-part has the size which corresponds to \( \text{eva} \), the same allocation state as the \( \text{vr} \)-part of \( \text{op} \), and which in the parts identified by the allocation state is identical with the \( \text{vr} \)-part of \( \text{op} \).

Before the actual conversion a test is made whether the conversion is possible. The conversion is possible if by a sequence of allocations a value representation of the size corresponding to \( \text{eva} \) can be given the allocation state of the \( \text{vr} \)-part of \( \text{op} \). If the conversion is not possible the AREA condition is raised.

\( \text{area-conv}(\text{eva-tg}, \text{op}) = \)

\[
\text{sc-area-cond}(z_1, \text{alloc-states-s-vr(op)}) \rightarrow \\
\text{PASS} : p_3(<s\text{-eva:eva-tg}, <s\text{-vr:area-assign}(z_1, s\text{-vr(op)})>) \\
\text{T } \rightarrow \\
\text{error} : \\
\text{call-cond}(\text{AREA})
\]

where:

\[
z_1 = \text{alloc-size(\text{eva-tg})}
\]

for:is-area-s-da(\text{eva-tg})

Ref.: alloc-state 3-16(53) \\
call-cond 10-18(54) \\
alloc-size 7-11(26)
(141) no-area-cond\(z, allst\) =

\[
\begin{align*}
\{ & \text{is-ptr-val-list}(\text{list}) \land \\
& \text{allst} = \{ p \mid p = \text{elem}(i, \text{list}) \land 1 \leq i \leq \text{length}(\text{list}) \} \land \\
& \text{size}(\text{vr}) = z \land \\
& \text{allocatable}(\text{list}, \text{vr}) \}
\end{align*}
\]

for is\(\text{-}\text{size}(z) \land \text{is-ptr-val-set}(\text{allst})\)

Ref.: size 3-15(33)
      is-size 3-15(34)
      is-ptr-val 3-15(36)

(142) allocatable\(\text{list}, \text{vr}\) =

\[
\begin{align*}
\text{is-\(<\text{>}(\text{list})} \rightarrow T \\
T \rightarrow \\
\text{is-free-space}(\text{head}(\text{list}), \text{vr}, \text{AREA}) \land \\
\text{allocatable}(\text{tail}(\text{list}), \text{el-alloc}(\text{head}(\text{list}), \text{vr}))
\end{align*}
\]

for is\text{-}\text{ptr-val-list}(\text{list})

Ref.: is-free-space 3-17(56)
      el-alloc 3-17(55)
      is-ptr-val 3-15(36)

(143) area-assign\(z, vr\) =

for is\text{-}\text{size}(z)

Ref.: is-size 3-15(34)

Note: This implementation-defined function gives a value representation. Its properties are defined by the following assertion.

(144) alloc-state\text{-}area-assign\(z, vr\) = alloc-state\(vr\) \&

\[
\{ p \mid p \in \text{alloc-state}(\text{vr}) \land \\
& \text{p}\text{-area-assign}(z, vr) = p(\text{vr}) \} \land \\
& \text{size}\text{-area-assign}(z, vr) = z
\]

for is\text{-}\text{size}(z) \land \text{is-ptr-val}(p)

Ref.: alloc-state 3-16(53)
      size 3-15(33)
      is-size 3-15(34)
      is-ptr-val 3-15(36)

9.5.3 COMPLETION OF TARGIT ATTRIBUTES

The predicate is-inc-eda characterizes a class of "incomplete data attributes" that differs from the class characterized by is-correct-eda (cf. 9.7) in that any component of an arithmetic or string attribute may be replaced by \(*\). The function
complete-eva(eva-tg,eva) completes the data attribute component eda-tg of eva-tg by means of its subfunction complete-tg and the source attribute eda. If eda-tg and eda are of different types, then eda is first transformed to the appropriate type by one of the functions mk-arithm(eda), mk-string(eda), mk-char-string(eda), mk-bit-string(eda).

Metavariable

eda-tg is-inc-eda target data attribute

(145) complete-eva(eva-tg,eva) =
\( \mu(eva-tg; \langle s-da:complete-tg(s-da(eva-tg),s-da(eva)) \rangle) \)

(146) is-inc-eda =
\( \text{is-inc-arithm} \times \text{is-inc-string} \times \text{is-plc} \times \text{is-area} \times \text{is-ENTRY} \times \text{is-LABEL} \times \text{is-CFSET} \times \text{is-TH} \times \text{is-FILE} \times \text{is-TASK} \times \text{is-EVENT} \)

(147) is-inc-arithm =
\( \langle \langle s-mode:is-REAL \times \text{is-CFLX} \times \text{is-}^\ast \rangle, \langle s-base:is-BIN \times \text{is-DEC} \times \text{is-}^\ast \rangle, \langle s-scale:is-FIX \times \text{is-TIT} \times \text{is-}^\ast \rangle, \langle s-prec:is-intg-val \times \text{is-}^\ast \times \text{is-}^\ast \rangle, \langle s-scale-f:is-intg-val \times \text{is-}^\ast \rangle \rangle \)

Ref.: is-intg-val 9-3(5)

(148) is-inc-string =
\( \langle \langle s-base:is-BIT \times \text{is-CHAR} \times \text{is-}^\ast \rangle, \langle s-varying:is-VAR \times \text{is-}^\ast \times \text{is-}^\ast \rangle, \langle s-length:is-intg-val \times \text{is-}^\ast \rangle \rangle \)

Ref.: is-intg-val 9-3(5)

(149) complete-tg(eda-tg,eda) =
\( \text{is-inc-arithm}(eda-tg) \rightarrow \text{complete-ar}(eda-tg,\text{mk-arithm}(eda)) \)
\( \text{is-inc-string}(eda-tg) \rightarrow \text{complete-str}(eda-tg,\langle \text{is-CHAR} \times \text{base}(eda-tg) \rightarrow \text{mk-char-string}(eda) \rangle, \text{is-BIT} \times \text{base}(eda-tg) \rightarrow \text{mk-bit-string}(eda), T \rightarrow \text{mk-string}(eda)) \)
\( \text{is-area}(eda-tg) \rightarrow \text{complete-area}(eda-tg,eda) \)
\( T \rightarrow \text{eval-da-1}(eda-tg) \)

cont'd
Abstract Syntax and Interpretation of PL/I

Ref.: eval-da-1 6-10(26)

9.5.3.1 Completion of arithmetic attributes

Notavariables

eda-tg is-isc-arith
eda is-arith

150) complete-ar(eda-tg,eda) =
    complete-prec(ccomplete-m-b-s(eda-tg,eda),eda)

151) complete-m-b-s(eda-tg,eda) =
    \mu(eda-tg;\langle s:e(eda) \rangle | s \in \{s-mode,s-base,s-scale\} \& is-**s(eda-tg))

152) complete-prec(eda-tg,eda) =
    is-**s-prec(eda-tg) \& is-FIX\*s-scale(eda-tg) \rightarrow
    \mu(eda-tg;\langle s-prec:min(N-tg_1,\text{result-prec}_1)\rangle,
    \langle s-scale-f:sign(q-eda_1) \cdot \text{ceil(abs(q-eda_1) \cdot f-prec_1)}\rangle)
    \rightarrow
    \mu(eda-tg;\langle s-prec:min(N-tg_1,\text{ceil}(p-eda_1 \cdot f-prec_1))\rangle)
    \rightarrow
    eda-tg

where:
    N-tg_1 = \text{max-prec}(eda-tg)
    f-prec_1 = \text{prec-fact}(eda-tg,eda)
    \text{result-prec}_1 = (f-prec_1 = 1 \rightarrow p-eda_1,\n    T \rightarrow 1 + \text{ceil}(p-eda_1 \cdot f-prec_1))

Ref.: max-prec 9-12(61)

153) prec-fact(eda-tg,eda) =
    s-base(eda) = s-base(eda-tg) \rightarrow 1
    \rightarrow 1 / 3 \cdot 32
    is-EIN\*s-base(eda) \rightarrow 3 \cdot 32
    is-TIC\*s-base(eda) \rightarrow 3 \cdot 32
2.5.3.2 Completion of string attributes

\[(154)\] \(\text{complete-str}(eda-tg,eda) =\)

\(\mu(eda-tg; \langle \varepsilon; (eda) \rangle \mid \text{is-string}(eda-tg))\)

for: is-inc-string(eda-tg) & is-string(eda)

2.5.3.3 Completion of area attributes

\[(155)\] \(\text{complete-area}(eda-tg,eda) =\)

\(-is\text{-area}(eda) \rightarrow \text{error}\)

\(-is\text{-area-size}(eda-tg) \rightarrow eda\)

T \rightarrow eda-tg

for: is-area(eda-tg)

2.5.3.4 Transformation into an arithmetic attribute

\[(156)\] \(\text{mk-arithm}(eda) =\)

\(-is\text{-arithm}(eda) \rightarrow eda\)

\(-is\text{-num-pic}(eda) \rightarrow \text{mk-arithm-pic}(eda)\)

\(-is\text{-string-type}(eda) \rightarrow\)

\(\{\text{str-base}(eda) = \text{CHAR} \rightarrow \text{intg-eda}(\text{max-prec}(\text{INTG}-EDA))\},\)

\(\text{str-base}(eda) = \text{BIT} \rightarrow \text{bitg-eda}(\text{max-prec}(\text{BITG}-EDA))\)

T \rightarrow \text{error}

Ref.: \(\text{is-num-pic} 9-45(173)\)

\(\text{mk-arithm-pic} 9-63(268)\)

\(\text{is-string-type} 9-21(92)\)

\(\text{intg-eda} 9-11(87)\)

\(\text{max-prec} 9-12(61)\)

\(\text{bitg-eda} 9-17(48)\)

2.5.3.5 Transformation into a string attribute

\[(157)\] \(\text{mk-string}(eda) =\)

\(-\{\text{is-arithm} \times \text{is-string} \times \text{is-pic}(eda) \rightarrow \text{error}\}\)

\(\text{str-base}(eda) = \text{CHAR} \rightarrow \text{mk-char-string}(eda)\)

\(\text{str-base}(eda) = \text{BIT} \rightarrow \text{mk-bit-string}(eda)\)
**Abstract Syntax and Interpretation of PL/I**

30 April 1969

(158) \( \text{mk-char-string}(\text{eda}) = \)

\( \text{is-arithm}(\text{eda}) \rightarrow \mu (\langle \text{s-base:CHAR}, \langle \text{s-length:char-length}(\text{eda}) \rangle \rangle) \)

\((\text{is-string} \lor \text{is-pic})(\text{eda}) \rightarrow \mu (\text{mk-str-1}(\text{eda}); \langle \text{s-base:CHAR} \rangle)\)

\( T \rightarrow \text{error} \)

(159) \( \text{char-length}(\text{eda}) = \)

\( \text{is-BIN}=\text{base}(\text{eda}) \rightarrow \text{char-length}(\text{complete-tg}(\mu (\langle \text{AR-ED}, \langle \text{s-base:DEC} \rangle \rangle), \text{eda})) \)

\( \text{is-CPLX}=\text{mode}(\text{eda}) \rightarrow 2 \cdot \text{char-length}(\text{eda-real}_{1}) \)

\( \text{is-FLT}=\text{scale}(\text{eda}) \rightarrow \text{p-eda}_{1} + 4 + \text{EXP-SIZE} \)

\( \text{is-FIX}=\text{scale}(\text{eda}) \rightarrow \)

\((0 \leq q_{-}\text{eda}_{1} \leq \text{p-eda}_{1} \rightarrow \text{p-eda}_{1} + 3, \)

\( T \rightarrow \text{p-eda}_{1} + 3 + \text{lgth-q}_{1} \)

where:

\( \text{lgth-q}_{1} = (2i)(10 \uparrow (i - 1)) \leq \text{abs}(q_{-}\text{eda}_{1}) < 10 \uparrow i) \)

for:is-prop-arithm(eda)

Note: cf. numeric to character conversion 9.5.1.1 and 9.6.3.5.

(160) \( \text{mk-bit-string}(\text{eda}) = \)

\( \text{is-num-type}(\text{eda}) \rightarrow \mu (\langle \text{s-base:BIT}, \langle \text{s-length:bit-length}(\text{mk-arithm}(\text{eda})) \rangle \rangle) \)

\( \text{is-string-type}(\text{eda}) \rightarrow \mu (\text{mk-str-1}(\text{eda}); \langle \text{s-base:BIT} \rangle) \)

\( T \rightarrow \text{error} \)

Ref.: is-num-type 9-21(91)

is-string-type 9-21(92)

(161) \( \text{bit-length}(\text{eda}) = \)

\( \max (0, \min (N_{-}\text{tg}_{1}, \text{ceil}(\text{lgth-eda}_{1} \cdot \text{f-prec}_{1}))) \)

where:

\( N_{-}\text{tg}_{1} = \max -\text{prec}(\text{BINWC-ED!}) \)

\( \text{lgth-eda}_{1} = \text{is-FIX}=\text{scale}(\text{eda}) \rightarrow \text{p-eda}_{1} - \text{q-eda}_{1}, \)

\( T \rightarrow \text{p-eda}_{1} \)

\( \text{f-prec}_{1} = \text{prec-fact}(\text{BINWC-ED!}, \text{eda}) \)

for:is-prop-arithm(eda)

Ref.: \( \max -\text{prec} 9-12(61) \)

40 9. DATA, OPERATIONS AND CONVERSIONS
(162) \[ \text{mk-str-1(eda)} = \]
\[
\begin{align*}
\text{is-string(eda)} & \rightarrow \text{eda} \\
\text{is-pic(eda)} & \rightarrow \mu_{o}(<\text{s-base: str-base(eda)},<\text{s-length: str-length(eda)}/>)
\end{align*}
\]

for: (is-string \(\lor\) is-pic) (eda)

(163) \[ \text{str-base(eda)} = \]
\[
\begin{align*}
\text{is-string(eda)} & \rightarrow \text{s-base(eda)} \\
\text{(is-arithm \& is-BIN-s-base)(eda)} & \lor \text{is-bin-pic(eda)} \rightarrow \text{BIT} \\
\text{T} & \rightarrow \text{CHAR}
\end{align*}
\]

for: (is-prop-arithm \(\lor\) is-string \(\lor\) is-pic) (eda)

(164) \[ \text{str-length(eda)} = \]
\[
\begin{align*}
\text{is-string(eda)} & \rightarrow \text{s-length(eda)} \\
\text{is-pic(eda)} & \rightarrow \text{pic-length(eda)}
\end{align*}
\]

for: (is-string \(\lor\) is-pic) (eda)

Ref.: pic-length 9-64(250)

5.6 PICTURES

This chapter defines functions, instructions and predicates that are needed for the evaluation, representation, conversion and checking of pictures. The notion of "explicit" picture and certain auxiliary predicates and functions are defined in 9.6.1, the predicate is-correct-pic(eda) in 9.6.2, the instructions rep-pi£(eva,v) and num-pic-char-conv(eda,v) in 9.6.3, the function pic-val(eda,vr) and the predicate is-pic-match(eda,v) in 9.6.4, finally certain functions for the transformation of picture attributes into other attributes in 9.6.5.

Metavariables

<table>
<thead>
<tr>
<th>Var</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>eda</td>
<td>explicit picture attribute (cf. 9.6.1.1)</td>
</tr>
<tr>
<td>sf</td>
<td>subfield selector</td>
</tr>
<tr>
<td>spec</td>
<td>single picture specification, i.e., character appearing in a picture attribute</td>
</tr>
<tr>
<td>pv</td>
<td>value in pictured form (cf. 9.6.3)</td>
</tr>
<tr>
<td>nl</td>
<td>list of numbers</td>
</tr>
<tr>
<td>sgn</td>
<td>sign of a numeric value</td>
</tr>
</tbody>
</table>
Abbreviations

sf-mt = s-mt-field
sf-exp = s-exp-field

sg: sg e {PLUS, MINUS} sign character

lgth-eda = pic-length*ppic(eda) length of string represented with a picture attribute

for is-xnum-pic(eda):

lgth-sf = length*sf*ppic(eda) length of subfield
lgth-mt = length*sf-mt*ppic(eda) length of mantissa subfield
u-eda = s-st-unit(eda) unit position of mantissa subfield
g-eda = is-g-s-scale-f(eda) -- 0
T -> s-scale-f(eda) explicit scale factor of eda

for is-xnum-pic(eda) & is-CPLX<s-mode(eda) :

eda-real = μ(eda;<s-mode:REAL>) real data attribute corresponding to a complex eda

for (is-xdec-pic v is-xsterling-pic) (eda):

d-i = dig-pos(eda,sf,i) i-th digit position of a subfield
d-md = dig-pos(eda,sf,nd-sf) last digit position of a subfield
The class of "explicit" picture attributes is introduced in 9.6.1.1. Predicates for the classification of picture attributes and picture specifications (i.e., characters in the picture attribute) are listed in 9.6.1.2.

9.6.1.1 Explicit picture attributes

The "explicit" picture attribute corresponding to a picture attribute is this picture attribute itself, if the latter is character picture, binary picture, or decimal or sterling picture without leading zero suppression. In the remaining cases, it is the unsuppressed picture, i.e., the picture attribute with the suppression positions replaced by the ordinary digit specification 9-CHAR, with certain components added that contain the suppression information more explicitly. These components are s-dr-beg, s-dr-end, s-dr-spec (drift begin, drift end, drifting specifications) for each subfield that originally contained suppression specifications. The reason for introducing the transformation into this explicit form is that both checking of picture attributes and representing a value in pictured form become simpler.

9.6.1.1.1 Syntax of explicit picture attributes

The syntax of explicit picture attributes is defined by the predicate is-xpic.
(165) \textit{is-xpic(x)} =
\begin{align*}
\text{is-correct-pic(x)} & \vee \text{(is-dec-pic} \vee \text{is-sterling-pic)} \cdot \text{ppic(x)} & \& \\
\text{et} & \text{(is-intg-val} \cdot \text{s-dr-beg} \& \text{is-intg-val} \cdot \text{s-dr-end} & \\
\text{sf} & \text{sf-set(x)} & \\
\text{is-dr-spec} \cdot \text{s-dr-spec} & \cdot \text{sf(x)}
\end{align*}

Ref.: is-intg-val 9-3(5)

(166) \textit{ppic(x)} =
\begin{align*}
\delta(x; \{ s \in \text{dr-set} \& \text{sf} \in \text{sf-set}(x) \})
\end{align*}

Note: \textit{ppic(x)} is the "pure picture part" of \( x \), i.e., with the drifting information deleted.

(167) \textit{dr-set} =
\begin{align*}
\{ \text{s-dr-beg, s-dr-end, s-dr-spec} \}
\end{align*}

(168) \textit{sf-set(x)} =
\begin{align*}
\{ \text{sf} \in \{ \text{sf-st}_1, \text{sf-exp}_1 \} \& \neg \text{is-csf(x)} \}
\end{align*}

Note: \textit{sf-set(x)} is the set of selectors that select occupied subfields from \( x \).
For fixed-point pictures, this set consists of the single selector \( \text{sf-st}_1 \),
for floating-point pictures, of the two selectors \( \text{sf-st}_1, \text{sf-exp}_1 \).

9.6.1.2 Transformation of an explicit picture into a picture

The picture attribute (or "implicit" picture attribute) corresponding to an explicit picture attribute \( \text{eda} \) is obtained by the function \( \text{mpic(eda)} \).

(169) \textit{mpic(eda)} =
\begin{align*}
\text{is-char-pic} \vee \text{is-bin-pic}(\text{eda}) \rightarrow \text{eda} \\
\text{T} \rightarrow \text{mpic-1(eda)}
\end{align*}

(170) \textit{mpic-1(eda)} =
\begin{align*}
\text{is-fix-pic(eda)} \rightarrow \text{msubf(eda,sf-st}_1) \\
\text{is-flt-pic(eda)} \rightarrow \text{msubf(msubf(eda,sf-st}_1,\text{sf-exp}_1)}
\end{align*}

for: (is-xdec-pic \vee is-xsterling-pic)(eda)
\(\text{nsubf}(eda, sf) = \)
\[
\text{is-w(spec-dr)} \rightarrow \text{eda}
\]
\[
T \rightarrow \mu(\text{ppic}(eda): \left\{ \text{elem}(d-i) \cdot \text{spec-dr} \right\}, 1 \leq i \leq n-d \cdot sf \wedge \text{beg-dr} \leq d-i \leq \text{end-dr} \wedge \neg \text{is-Y-CHAR} \cdot \text{elem}(d-i, sf(eda)))
\]

for: (is-xdec-pic \lor \text{is-xsterling-pic})(eda)

Note: The digit positions of the subfield between \text{beg-dr} and \text{end-dr} that are different from \text{Y-CHAR} are replaced by \text{spec-dr} and the explicit drifting information is deleted.

9.6.1.3 Transformation of a picture into an explicit picture

The explicit picture attribute corresponding to a picture attribute \text{eda} is obtained by the function \text{xpic}(eda). The definition, in all but the trivial cases, is implicit and interferes with that of \text{is-correct-pic}(eda) (cf. 9.6.2).

\(\text{xpic}(eda) = \)
\[
\text{(is-char-pic \lor \text{is-bin-pic})(eda)} \rightarrow \text{eda}
\]
\[
T \rightarrow \text{leda-1}(\text{is-correct-pic-1(eda-1)} \wedge \text{eda} = \text{spic-1(eda-1)})
\]

for: \text{is-correct-pic}(eda)

9.6.1.2 Classification of pictures and picture specifications

Some predicates classifying (implicit and explicit) pictures, predicates classifying picture specifications (i.e., characters appearing in picture attributes) and some functions for the treatment of digit positions are defined.

9.6.1.2.1 Predicates classifying pictures

\(\text{is-num-pic} = \)
\[
\text{is-bin-pic} \lor \text{is-dec-pic} \lor \text{is-sterling-pic}
\]

\(\text{is-xnum-pic} = \)
\[
\text{is-bin-pic} \lor \text{is-xdec-pic} \lor \text{is-xsterling-pic}
\]

\(\text{is-xdec-pic}(eda) = \)
\[
\text{is-dec-pic} \cdot \text{ppic}(eda)
\]

\(\text{is-xsterling-pic}(eda) = \)
\[
\text{is-sterling-pic} \cdot \text{ppic}(eda)
\]
(177) \text{is-fix-pic}(eda) = \\
\text{is-o-sf-exp}(eda) \land \text{is-o-s-exp-sep}(eda) \\
\text{for:is-xnum-pic}(eda)

(178) \text{is-flt-pic}(eda) = \\
\neg \text{is-o-sf-exp}(eda) \land \text{is-o-s-scale-f}(eda) \\
\text{for:is-xnum-pic}(eda)

2.6.1.2.2. Predicates classifying picture specifications

The following predicates serve to classify characters occurring within picture attributes. Where the classification is context dependent, the arguments of the predicates have to be positions, i.e., selectors rather than characters.

Metavariable

eda \quad \text{is-xnum-pic} \quad \text{explicit numeric picture attribute}

(179) \text{is-pic-spec} = \\
\text{is-char-spec} \lor \text{is-bin-spec} \lor \text{is-sterling-spec}

(180) \text{is-char-ed-spec} = \\
\text{is-BLANK} \lor \text{is-ASTER} \lor \text{is-SLASH} \lor \text{is-COMMA} \lor \text{is-POINT}

(181) \text{is-dig-spec}(eda,sf,i) = \\
\text{is-dec-dig-spec}(eda,sf,i) \lor (\text{is-8-CHAR} \lor \text{is-7-CHAR} \lor \text{is-6-CHAR}) \cdot \text{elem}(i,sf(eda))

(182) \text{is-dec-dig-spec}(eda,sf,i) = \\
(is-9-CHAR \lor is-Y-CHAR \lor \text{is-T-CHAR} \lor \text{is-I-CHAR}) \cdot \text{elem}(i,sf(eda)) \lor \\
\text{is-r-char}(eda,sf,i)

(183) \text{is-r-char}(eda,sf,i) = \\
\text{is-R-CHAR} \cdot \text{elem}(i,sf(eda)) \land \neg \text{is-C-CHAR} \cdot \text{elem}(i-1,sf(eda))

(184) \text{is-sign-spec}(eda,sf,i) = \\
(is-SIGN \lor is-PLUS \lor \text{is-MINUS} \lor \text{is-T-CHAR} \lor \text{is-I-CHAR}) \cdot \text{elem}(i,sf(eda)) \lor \\
\text{is-r-char}(eda,sf,i)
(185) \texttt{is-trail-sign(eda,sf,i) =}
\texttt{is-C-CHAR\cdot\text{elem}(i,sf(eda)) \& is-R-CHAR\cdot\text{elem}(i+1,sf(eda)) \lor is-D-CHAR\cdot\text{elem}(i,sf(eda)) \& is-B-CHAR\cdot\text{elem}(i+1,sf(eda))}

(186) \texttt{is-sign-pos(eda,sf,i) =}
\texttt{(2i.(s (\text{elem}(i))=sf \& is-sign-spec(eda,sf,i)) \lor sf = sf-mt_k \& (3sf^{-1},i).s = (\text{elem}(i))\cdot sf^{-1} \& is-trail-sign(eda,sf^{-1},i))}

\textbf{Note:} A trailing sign CR or EB is interpreted as a sign position of the mantissa subfield, even if it appears within the exponent subfield.

(187) \texttt{is-dr-spec =}
\texttt{is-S-CHAR \lor is-ASTER \lor is-dr-ed-spec}

(188) \texttt{is-dr-ed-spec =}
\texttt{is-SIGN \lor is-PLUS \lor is-MINUS \lor is-DOLLAR}

(189) \texttt{is-ed-spec(eda,sf,i) =}
\texttt{(is-BLANK \lor is-POINT \lor is-COMMA \lor is-SLASH \lor is-S-CHAR\cdot\text{elem}(i,sf(eda)) \lor is-d-char(eda,sf,i)}

(190) \texttt{is-d-char(eda,sf,i) =}
\texttt{is-D-CHAR\cdot\text{elem}(i,sf(eda)) \& \neg is-B-CHAR\cdot\text{elem}(i+1,sf(eda))}

9.6.1.2.3 Functions computing and counting digit positions

dig-pos(eda,sf,i) yields \(d_i\), the \(i\)-th digit position of the subfield \(sf(eda)\);
n-of-d-subf(eda,sf) yields the number of digit positions in the subfield \(sf(eda)\);
n-of-d(eda,sf,m,n) the number of those positions between \(m\) and \(n\);
dig-index(eda,sf,n) yields the index of the rightmost digit position that precedes \(n\) equals position \(n\).

(191) \texttt{dig-pos(eda,sf,i) =}
\texttt{i = 0 \rightarrow 0}
\texttt{T \rightarrow}
\texttt{(\neg (n > dig-pos(eda,sf,i - 1) \& (n = lqth-sf_k + 1 \lor is-dig-spec(eda,sf,n)) \&
(\forall j).dig-pos(eda,sf,i - 1) < j < n = \neg is-dig-spec(eda,sf,j)))}

for: \texttt{is-xdec-pic \lor is-xsterling-pic (eda)} \& \texttt{i \geq 0}
(192) \( \text{n-of-d-ub}(\text{eda}, \text{sf}) = \)
\( \text{is-bin-pic}(\text{eda}) \rightarrow \)

\( (\text{is-sign-mag-pic}(\text{eda}) \& \neg \text{is-SIGN-elem}(1, sf(\text{eda})) \rightarrow \text{lgth-sf}_1, \)

\( T \rightarrow \text{lgth-sf}_1 - 1) \)

\( T \rightarrow \text{n-of-d}(\text{eda}, \text{sf}, 1, \text{lgth-sf}_1) \)

for: \( \text{is-xnum-pic}(\text{eda}) \)

(193) \( \text{n-of-d}(\text{eda}, \text{sf}, m, n) = \)

\( m > n \rightarrow 0 \)

\( \text{is-dig-spec}(\text{eda}, \text{sf}, m) \rightarrow ? + \text{n-of-d}(\text{eda}, \text{sf}, m + 1, n) \)

\( T \rightarrow \text{n-of-d}(\text{eda}, \text{sf}, m + 1, n) \)

for: \( \text{is-xdec-pic} \lor \text{is-sterling-pic}(\text{eda}) \)

(194) \( \text{dig-index}(\text{eda}, \text{sf}, n) = \)

\( (\text{Li})(\text{dig-pos}(\text{eda}, \text{sf}, i) \leq n < \text{dig-pos}(\text{eda}, \text{sf}, i + 1)) \)

for: \( \text{is-xdec-pic} \lor \text{is-sterling-pic}(\text{eda}) \)

9.6.2 CHECKING PICTURE ATTRIBUTES

The class \( \text{is-pie} \) defined by the abstract syntax is much wider than the class of actually permitted picture attributes; the latter is characterized by the predicate \( \text{is-correct-pie} \).

(195) \( \text{is-correct-pie}(\text{eda}) = \)

\( \text{is-char-pic}(\text{eda}) \lor \)

\( (\text{is-bin-pic}(\text{eda}) \& (\text{is-sign-mag-pic} \lor \text{is-2-compl-pic} \lor \text{is-1-compl-pic})(\text{eda}) \& \)

\( \text{is-correct-pie-1}(\text{eda}) \lor \)

\( ((\text{is-dec-pic} \lor \text{is-sterling-pic})(\text{eda}) \& \)

\( (\exists \text{eda-1})(\text{is-correct-pie-1}(\text{eda-1}) \& \text{eda} = \text{spic-1}(\text{eda-1}))) \)

for: \( \text{is-pie}(\text{eda}) \)

(196) \( \text{is-correct-pie-1}(\text{eda}) = \)

\( (\text{is-fix-pic} \lor \text{is-flt-pic})(\text{eda}) \& (\text{is-0}(u-\text{eda}) \lor 0 \leq u-\text{eda} \leq \text{lgth-mt}_1) \& \)

\( \text{sf} \in \text{sf-set}(\text{eda}) \)

for: \( \text{is-xnum-pic}(\text{eda}) \)
9.6.2.1 Three classes of binary pictures

With binary pictures the classes of sign-magnitude pictures, two-complement pictures and one-complement pictures are distinguished; these classes correspond to different kinds of representing values in pictured form.

Metavariable

\( \text{eda} \quad \text{is-bir-pic} \)

(197) \( \text{is-sign-mag-pic}(\text{eda}) = \)

\[ \text{Et} \quad (\text{is-7-CHAR-list} \lor \text{is-SIGN-head} \land \text{is-7-CHAR-list-tail}) \circ \text{sf}(\text{eda}) \]

\( \text{sf} \in \text{sf-set}(\text{eda}) \)

(198) \( \text{is-2-compl-pic}(\text{eda}) = \)

\[ \text{Et} \quad \text{is-2-CHAR-list} \circ \text{sf}(\text{eda}) \]

\( \text{sf} \in \text{sf-set}(\text{eda}) \)

(199) \( \text{is-1-compl-pic}(\text{eda}) = \)

\[ \text{Et} \quad \text{is-3-CHAR-list} \circ \text{sf}(\text{eda}) \]

\( \text{sf} \in \text{sf-set}(\text{eda}) \)

9.6.2.2 Proper subfields

(200) \( \text{is-correct-subf}(\text{eds}, \text{sf}) = \)

\[
\{ \text{is-xdec-pic} \lor \text{is-asterling-pic}(\text{eda}) \} \lor
\{ \text{is-6-CHAR} \lor \text{is-ASTER}\circ \text{elem}(i, \text{sf}(\text{eda})) \} \land
\neg (\exists i, j) (i \neq j \land \text{is-DOLLAR}\circ \text{elem}(i, \text{sf}(\text{eda})) \land \text{is-DOLLAR}\circ \text{elem}(j, \text{sf}(\text{eda})) \land
\text{is-trail-sig}(\text{eda}, \text{sf}, i) \lor i > d-n1 \land
\text{is-flt-pic}(\text{eda}) \lor \text{sf} = \text{sf-exp}1) \} \land
\text{is-xdec-pic}(\text{eda}) \land
\text{is-correct-drift}(\text{eda}, \text{sf}) \}
\]

for: \( \text{is-xnum-pic}(\text{eda}) \)
9.6.2.3 Additional restrictions for sterling subfields

Metavariables

eda       is-xsterling-pic
sf         \text{sf = sf-st}_4 \quad \text{sterling pictures have no exponent field}

(201) \text{is-correct-sterling-subf}(eda,sf) =

\begin{align*}
& (0 \leq \text{end-st}_4 < \text{end-pd}_4 < \text{end-sh}_4 < \text{end-ipc}_4 < \text{lgth-sf}_4) \land \\
& \text{Ed} (\text{is-dr-ed-spec} \cdot \text{elem}(i,sf(eda))) \land \\
& (i > \text{end-st}_4 \equiv \text{elem}(i,sf(eda)) \equiv \text{spec-dr}_4) \land \\
& (\text{is-dec-dig-spec}(eda,sf,i) \lor \text{is-COMMA} \cdot \text{elem}(i,sf(eda)))) \land \\
& (3n) (\text{end-pd}_4 < n < \text{end-sh}_4) \land \\
& n-1 \text{Ed} (\text{is-ELANK} \lor \text{is-POINT} \lor \text{is-SLASH} \cdot \text{elem}(i,sf(eda))) \land \\
& i=\text{end-pd}_4+1 \text{Ed} (\text{is-correct-sh-pc}(eda,sf,n,\text{end-sh}_4)) \land \\
& (3n) (\text{end-sh}_4 < n < \text{end-ipc}_4) \land \\
& n-1 \text{Ed} (\text{is-ELANK} \lor \text{is-POINT} \lor \text{is-SLASH} \lor \text{is-S-CHAR} \cdot \text{elem}(i,sf(eda))) \land \\
& i=\text{end-ipc}_4+1 \text{Ed} (\text{is-dec-dig-spec}(eda,sf,i) \lor i = \text{end-ipc}_4 + 1 \lor \\
& i=\text{end-ipc}_4+1 \text{Ed} (\text{is-SLASH} \cdot \text{elem}(i,sf(eda)))) \land \\
& \text{lgth-sf}_4 \text{Ed} (\text{is-ELANK} \lor \text{is-D-CHAR} \lor \text{is-dr-ed-spec}) \cdot \text{elem}(i,sf(eda)) \land \\
& i=\text{end-ipc}_4+1 \text{Ed} (\text{is-trail-sign}(eda,sf,i) \lor \text{is-trail-sign}(eda,sf, i - 1))
\end{align*}

(202) \text{is-correct-sh-pc}(eda,sf,n,n) =

\begin{align*}
& (n = n \land \\
& (n = \text{end-sh}_4 \land \text{is-}8-\text{-CHAR} \cdot \text{elem}(n,sf(eda)) \land n = \text{end-ipc}_4 \land \\
& (\text{is-7-CHAR} \lor \text{is-6-CHAR} \cdot \text{elem}(n,sf(eda)))) \lor \\
& (m = n - 1 \land \text{is-dec-dig-spec}(eda,sf,m) \land \text{is-dec-dig-spec}(eda,sf,n) \land \\
& (\text{is-}8-\text{-CHAR} \cdot \text{elem}(n,sf(eda)) \lor \text{is-7-CHAR} \cdot \text{elem}(m,sf(eda))))
\end{align*}

9.6.2.4 Correct drifting

The following rules are expressed:

(a) Drifting must not begin after the first digit position; if drifting extends beyond \(u-eda_4\), then it must include all digit positions.

(b) If the drifting specification is a drifting editing specification, then at least two drifting specifications must have appeared in the non-explicit form of the subfield.

50 9. DATA, OPERATIONS AND CONVERSIONS
In the case of a sterling picture, if the drifting specification is a drifting editing specification or an asterisk, drifting must not extend beyond end-pd1.

Trailing editing specifications and Y-CHAR belong to the drifting part of the subfield.

Only certain kinds of specifications may be contained in the drifting part.

```plaintext
(203) is-correct-drift(eda,sf) =
     1 ≤ beg-dr1 ≤ dig-pos(eda,sf,1) ≤ end-dr1 ≤ lth-sf1 &
     (sf = sf-at_1 & is-dr(spec-dr_1) ∧ end-dr2 > u-eda = end-dr1 ≥ d-pd4) &
     (is-dr-ed-spec(spec-dr_1) ≥ beg-dr4 ≤ end-dr4) &
     (is-sterling-pic(eda) & (is-dr-ed-spec v is-ASTER)(spec-dr_4) =
       end-dr4 ≤ end-pd4) & (is-ed-spec(sf,eda,beg-dr_4) &
       (is-Y-CHAR-elem(elem(end-dr_1 + 1,sf(eda)))) &
       (vi) (beg-dr_1 ≤ i ≤ end-dr_1 ≤ (is-dr-ed-spec(spec-dr_1) & i = beg-dr_1 →
         elem(i,sf(eda))) = spec-dr_4,
T → is-ed-spec(eda,sf,i) v (is-CHAR v is-Y-CHAR)·elem(i,sf,(eda)))
```

9.6.3 REPRESENTATION IN PICTURED FORM

In this chapter, the instruction `rep-pic(eva,v)` is defined which represents a value v with an evaluated aggregate attribute whose data attribute component is a picture attribute. Numeric pictures are treated in 9.6.3.1 to 9.6.3.3, character pictures in 9.6.3.4. The instruction `num-pic-char-conv(eda,v)` for numeric picture to character conversion, which is closely related to `rep-pic`, is defined in 9.6.3.5.

**Pictured values.** For all kinds of pictures, `rep-pic` essentially computes a value, namely a list of character or bit values, and only in a last step, this value is transformed into a value representation. For decimal and sterling pictures, the essential step is the transformation of a numeric value into a "pictured value". A pictured value pv has the same structure as an (explicit) picture attribute, but any character in the picture attribute may be replaced by a character that represents a corresponding part of the numeric value to be represented. (For example, 0-CHAR may be replaced by any digit, or SIGN may be replaced by PLUS or MINUS or BLANK.) Only finally, the pictured value will be "linearized" to a string value.

(204) `rep-pic(eva,v) =`

```plaintext
`pass-rpic(eva,v-1);`
`v-1:rep-pic(eva,v)`
```

for: `is-num-pics-da(eva) & is-num-val(v) v is-char-pics-da(eva) & is-char-val-list(v)`

Ref.: Rep 9-6 (15)
is-num-val 9-3(3)
is-char-val 9-3(6)
9.6.3.1 Representation of numeric pictures

(205) \texttt{rep-pic-1}(\texttt{eda},v) =
    \texttt{is-char-pic}(\texttt{eda}) \rightarrow\n    \texttt{test-char-pic}(\texttt{eda},\\texttt{edit-char-pic}(\texttt{s-field(eda)},\texttt{adjust-string(CHAR,1gth-eda,v)}))\n    T \rightarrow \texttt{rep-num-pic-1}(\texttt{xpic(eda)},v)

for: \texttt{is-num-pic(eda)} & \texttt{is-num-val(v)} v \texttt{is-char-pic(eda)} & \texttt{is-char-val-list(v)}

Ref.: \texttt{adjust-string 9-8(32)} \texttt{is-num-val 9-3(3)} \texttt{is-char-val 9-3(6)}

Note: In case of a numeric picture, \texttt{eda} is transformed into explicit form.

9.6.3.1 Representation of numeric pictures

(206) \texttt{rep-num-pic-1}(\texttt{eda},v) =
    \texttt{is-CPLX-model(eda)} \rightarrow\n    \texttt{conc}(v-re, v-im);\n    v-re: \texttt{rep-real-pic-1}(\texttt{eda-real1},\texttt{real(v)}),\n    v-im: \texttt{rep-real-pic-1}(\texttt{eda-real1},\texttt{imag(v)})\n    T \rightarrow \texttt{rep-real-pic-1}(\texttt{eda},\texttt{real(v)})

for: \texttt{is-xnum-pic(eda)} & \texttt{is-num-val(v)}

Ref.: \texttt{is-num-val 9-3(3)}

(207) \texttt{conc}(a,b) =
    \texttt{PASS:a'b}

for: \texttt{is-list(a)} & \texttt{is-list(b)}
(208) rep-real-pic-1(eda,v) =
    is-fix-pic(eda) →
    pass-lin-fix(eda,v-1);
    v-1:rep-subf(eda,sf-mt1,trunc(v • base-eda1 + (g-eda1 - q-eda1)))
    is-flt-pic(eda) →
    pass-lin-flt(eda,v-1,v-2);
    v-1:rep-subf(eda,sf-mt1,trunc(v • base-eda1 + (q-eda1 - e-eda1))),
    v-2:rep-subf(eda,sf-exp1,e-eda1)

where:
   e-eda =
   (Ln)(base-eda1 • (p-eda1 - 1)) ≤ abs(v • base-eda1 + (g-eda1 - n)) <
   base-eda1 • p-eda1

for:is-xnum-pic(eda) & is-real-val(v)

Ref.: is-real-val 9-3(4)

Note: For floating-point pictures, v is split into mantissa and exponent in such
a way that the highest significant digit of v will appear in the first
digit position of the picture. In all cases, the values passed to rep-subf
are integers.

9.6.3.2 Representation of subfields

Metavariabiles

eda     is-xnum-pic
v       is-intg-val

(209) rep-subf(eda,sf,v) =
    is-subf-size-cond(eda,sf,v) → call-cond(SIZE)
    T → PASS:rep-valid-subf(eda,sf,v)

Ref.: call-cond 10-18(54)

Note: The SIZE condition is raised if v is too large to be held by the subfield.

(210) is-subf-size-cond(eda,sf,v) =
    is-xsterling-pic(eda) → abs(v) ≥ 240 • 10 • (nd-pd1 • q-eda1)
    T →
    abs(v) ≥ base-eda1 • nd-sf1 &
    (is-2-compl-pic(eda) = v ≠ -(base-eda1 • nd-sf1))
Note: Except for sterling pictures, base-eda \( n_d - s_f \) measures the capacity of the subfield. For two-complement representation, the negative limit value is representable.

\[(211) \quad \text{rep-valid-subf}(eda, sf, v) = \]
\[\text{is-bin-pic}(eda) \land \text{is-sign-mag-pic}(eda) \rightarrow \]
\[\text{rep-bin-sign}(eda, sf, \text{sign}(v)) \land \text{num-bit-list}(nd-sf_1, abs(v)) \]
\[\text{is-bin-pic}(eda) \rightarrow \text{num-bit-list}(lgth-sf_1, \text{compl-val}(eda, sf, v)) \]
\[T \rightarrow \]
\[\text{drift-pv}(eda, sf, i-s_4, \text{rep-sign}(eda, sf, sg-v_4, \text{rep-num-list}(eda, sf, sg-v_4, nl-v_4))) \]

where:
\[sg-v_4 = \text{sign-char}(v)\]
\[nl-v_4 = \text{num-bit-list}(eda, sf, abs(v))\]
\[i-s_4 = \{(i = nd-sf_1 + 1 \land 1 \leq i \leq nd-sf_1 \land \text{elem}(i, nl-v_4) \neq 0) \land \]
\[(V_j)(1 \leq j < i \Rightarrow \text{elem}(j, nl-v_4) = 0)\]

Ref.: num-bit-list 9-33(132)

Note: \( i-s_4 \) is the index of the first non-zero element of \( nl-v_4 \).

\[(212) \quad \text{sign-char}(v) = \]
\[(v \geq 0 \rightarrow \text{PLUS}, \quad T \rightarrow \text{MINUS})\]

9.6.3.2.1 Binary subfields

Metavarsiable

\[eda \quad \text{is-bin-pic}\]

\[(213) \quad \text{rep-bin-sign}(eda, sf, sgn) = \]
\[-\text{is-SIGN-head-sf}(eda) \rightarrow <>\]
\[T \rightarrow\]
\[(sgn = 1 \rightarrow <0\text{-BIT}>, \quad sgn = -1 \rightarrow <1\text{-BIT}>)\]

for:\text{is-sign-mag-pic}(eda)

54 9. Data, Operations and Conversions
(214) \( \text{comp-val}(\text{eda}, \text{sf}, v) = \)
\[
\begin{align*}
\text{is-2-comp-pic}(\text{eda}) & \rightarrow 2 \times \text{lgth-sf} - \text{abs}(v) \\
\text{is-1-comp-pic}(\text{eda}) & \rightarrow (2 \times \text{lgth-sf} - 1) - \text{abs}(v)
\end{align*}
\]
\(\text{for:}(\text{is-2-comp-pic} \lor \text{is-1-comp-pic})(\text{eda})\)

9.6.3.2.2 Decimal and sterling subfields

The function \( \text{mk-num-list}(\text{eda}, \text{sf}, v) \) decomposes the number \( v \) into a number list. The function \( \text{rep-num-list}(\text{eda}, \text{sf}, \text{sg}, \text{nl}) \) replaces the digit positions of the subfield \( \text{sf}(\text{eda}) \) by characters representing the elements of the number list \( \text{nl} \). These representing characters are computed by \( \text{rep-digit}(\text{spec}, \text{sg}, \text{v}) \); the sign-character \( \text{sg} \) is needed if \( \text{spec} \) specifies overpunching; the subfunctions \( \text{overpunch}(\text{sg}, \text{v}) \) and \( \text{rep-sterling}(\text{spec}, \text{v}) \) are implementation-defined, but obey suitable axioms guaranteeing their one-to-one-ness. The function \( \text{drift-v}(\text{eda}, \text{sf}, \text{i}, \text{pv}) \) performs zero-suppression and drifting on a subfield of a pictured value, depending on the index \( \text{i} \) of the first significant digit position.

Metavariable

\( \text{eda} \quad \text{is-xdec-pic} \lor \text{is-sterling-pic} \)

9.6.3.2.2.1 Decomposition into a number list

(215) \( \text{mk-num-list}(\text{eda}, \text{sf}, v) = \)
\[
\begin{align*}
\text{is-xdec-pc}(\text{eda}) & \rightarrow \text{mk-num-list-1}(\text{nd-sf}, v) \\
\text{is-sterling-pc}(\text{eda}) & \rightarrow \\
\text{mk-num-list-1}(\text{nd-pd}, \text{trunc}(v / 240)) \\
\text{mk-sterl-subl}(\text{eda}, \text{sf}, \text{end-sh}, \text{modulo}(\text{trunc}(v / 12), 20)) \\
\text{mk-sterl-subl}(\text{eda}, \text{sf}, \text{end-lp1}, \text{modulo}(v, 12)) \\
\text{mk-num-list-1}(\text{g-eda}, \text{modulo}(v, 10 + \text{g-eda}))
\end{align*}
\]

where:
\( v_1 = \text{trunc}(v / 10 + \text{g-eda}) \)

(216) \( \text{mk-num-list-1}(n, v) = \)
\[
\begin{align*}
\text{n} & = 0 \rightarrow \langle \rangle \\
\text{T} & \rightarrow \text{mk-num-list-1}(\text{n} - 1, \text{trunc}(v / 10))^{\langle \text{modulo}(v, 10) \rangle}
\end{align*}
\]

form \( \geq 0 \)
Abstract Syntax and Interpretation of PL/I

(217) \( mk\text{-sterl}\text{-subl}(eda, sf, n, v) = \)
\[
(is-6\text{-CHAR} \land is-7\text{-CHAR} \land is-8\text{-CHAR}) \cdot elem(n, sf(eda)) \rightarrow <v>
\]
\[ T \rightarrow mk\text{-num}\text{-list}\text{-1}(2, v) \]
for: is-sterling\text{-pic}(eda) \land n \in \{\text{end-sh}, \text{end-ipc}\} \land sf = sf\#a

9.6.3.2.2 Representation of number lists

(218) \( rep\text{-num}\text{-list}(eda, sf, sq, nl) = \)
\[
\mu(eda; \{<elem(d-i_1) \cdot sf > \cdot rep\text{-digit}(elem(d-i_1, sf(eda)), sq, elem(i, nl)) > | 1 \leq i \leq nd-sf_a)\})
\]

(219) \( rep\text{-digit}(spec, sq, n) = \)
\[
is-7\text{-CHAR}(spec) \land n = 0 \rightarrow BLANK
\]
\[
is-7\text{-CHAR}(spec) \lor is-7\text{-CHAR}(spec) \land is\text{-PLUS}(sq) \lor is-7\text{-CHAR}(spec) \land is\text{-MINDUS}(sq) \rightarrow
\]
\[
overpunch(sq, n)
\]
\[
(is-6\text{-CHAR} \land is-7\text{-CHAR} \land is-8\text{-CHAR})(spec) \rightarrow rep\text{-sterl}\text{-dig}(spec, n)
\]
\[ T \rightarrow single\text{-num}\text{-char}(n) \]

Ref.: single-num-char 9-32(129)

The functions

(220) \( overpunch(sq, n) = \)

(221) \( rep\text{-sterl}\text{-dig}(spec, n) = \)

are implementation-defined and are characterized by axioms (222) and (223) respectively, which follow.

(222) \( 0 \leq n-1 \leq 9 \land 0 \leq n-2 \leq 9 \Rightarrow is\text{-char}\text{-val}\cdot overpunch(sq-1, n-1) \land
\overline{overpunch(sq-1, n-1)} \neq single\text{-num}\text{-char}(n-2) \land
\overline{(sq-1 \neq sq-2 \lor n-1 \neq n-2)} \cdot overpunch(sq-1, n-1) \neq overpunch(sq-2, n-2) \)

Ref.: is-char-val 9-3(6)
single-num-char 9-32(129)

(223) \( is\text{-sterl}\text{-dig}\text{-match}(spec, n-1) \land is\text{-sterl}\text{-dig}\text{-match}(spec, n-2) \Rightarrow
\)
\( is\text{-char}\text{-val}\cdot rep\text{-sterl}\text{-dig}(spec, n-1) \land
\overline{(n-1 \neq n-2)} \cdot rep\text{-sterl}\text{-dig}(spec, n-1) \neq rep\text{-sterl}\text{-dig}(spec, n-2) \)

Ref.: is-char-val 9-3(6)
Abstract Syntax and Interpretation of PL/I

9.5.3.2.2.3 Representation of the sign

(224) \text{is-sterl-dig-match}(\text{spec}, n) = \text{is-8-CHAR}(\text{spec}) \land 0 \leq n \leq 19 \lor \text{(is-7-CHAR} \lor \text{is-6-CHAR})(\text{spec}) \land 0 \leq n \leq 11

9.6.3.2.2.4 Zero suppression and drifting

(225) \text{rep-sign}(\text{eda}, \text{sf}, \text{sg, pv}) =
\neg (\exists e) (\text{is-sign-pos}(\text{eda}, \text{sf}, e)) \rightarrow \text{pv}
(\exists j)(e^{-s}_1 = (\text{elem}(j)) \land \text{sf} \land \text{is-sign-spec}(\text{eda}, \text{sf}, i)) =
\neg (\text{is-SIGN} = e^{-s}_1(\text{eda}) \rightarrow \mu(\text{pv}; <e^{-s}_1; \text{sg}>),
\neg (\text{is-PLUS} \lor \text{is-MINUS})*e^{-s}_1(\text{eda}) \land e^{-s}_1(\text{eda}) \land \text{sg} \rightarrow \mu(\text{pv}; <e^{-s}_1; \text{BLANK}>),
T \rightarrow \text{pv})
T \rightarrow \text{(is-PLUS}(sg) \rightarrow \mu(\text{pv}; <(\text{elem}(j)) \land \text{sf} = e^{-s}_1; \text{BLANK}> \mid j \in [i^{-s}_2, i^{-s}_1 + 1]),
T \rightarrow \text{pv})

where:
\begin{align*}
e^{-s}_1 &= (\exists e) (\text{is-sign-pos}(\text{eda}, \text{sf}, e)) \\
\text{sf} = e^{-s}_1 &= (\text{sf} = 1) (\exists i)(e^{-s}_1 = (\text{elem}(i)) \land \text{sf} = 1) \\
i^{-s}_1 &= (\exists i)(e^{-s}_1 = (\text{elem}(i)) \land \text{sf} = 1)
\end{align*}

Note: The third alternative treats the trailing signs CR and DB (cf. \text{is-sign-pos} 9.6.1.2.2.2 (186)). The case of the sign specifications T-CHAR, I-CHAR, F-CHAR (overpunching) has already been treated by \text{rep-digit} (cf. 9.6.3.2.2.2 (219)).

9.6.3.2.2.4 Zero suppression and drifting

(226) \text{drift-pv}(\text{eda}, \text{sf}, i, \text{pv}) = \text{is-}\overline{U}(\text{spec-dr}_1) \rightarrow \text{pv}
T \rightarrow
\mu(\text{pv}; <(\text{elem}(j)) \land \text{sf} = \text{dr-spec-j}_1 \mid \text{beg-dr}_1 \leq j \leq \text{dyn-end-dr}_1 \land \neg (\text{is-STERLING-PIC}(\text{eda}) \land \text{end-pd}_1 < j < \text{end-ipc}_1 \land \text{is-ed-spec}(\text{eda}, \text{sf}, j)>)

where:
\begin{align*}
\text{dyn-end-dr}_1 &= \\
\text{sf} &= (\text{sf} \neq 1 \land \text{is-}\overline{U}(\text{u-eda}_1) \land u-eda_1 < \text{dig-pos}(\text{eda}, \text{sf}, i) - 1 \land \text{end-dr}_1 \rightarrow \text{u-eda}_1, T \rightarrow \min(\text{dig-pos}(\text{eda}, \text{sf}, i) - 1, \text{end-dr}_1)) \\
\text{dr-spec-j}_1 &= (\text{is-ASTER}(\text{spec-dr}_1) \rightarrow \text{ASTER,} \text{is-dr-ed-spec}(\text{spec-dr}_1) \land j = \text{dyn-end-dr}_1 \rightarrow \text{elem}(\text{beg-dr}_1, \text{sf}(\text{pv})
t \rightarrow \text{BLANK})
\end{align*}

Note: Drifting positions of pv are replaced by BLANK or ASTER, except the last drifting position in case the drifting specification is a drifting editing specification. Since i is the index of the first non-zero digit, \text{dig-pos}(\text{eda}, \text{sf}, i) - 1 is an upper limit for positions that may be
9.6.3.3 Composition of subfields and linearization

Metavariable

eda is-xnum-pic

(227) lin-fix(eda,pv) =
    is-bin-pic(eda) → pv
    T ← sf-nt×ppic(pv)
    for:is-fix-pic(eda)

Note: for is-bin-pic(eda), pv is a bit-val-list

(228) lin-flt(eda,pv-1,pv-2) =
    is-bin-pic(eda) → pv-1,pv-2
    T ← lin-util-merge-subf(eda,ppic(pv-1),ppic(pv-2))
    for:is-flt-pic(eda)

(229) merge-subf(eda,pv-1,pv-2) =
    μ(pv-1:
      [(elem(i))·sf-exp1:elem(i,sf-exp1(pv-2)) | 1 ≤ i ≤ length·sf-exp1(pv-2) & 
       ¬is-trail-sign(eda,sf-exp1,i) & ¬is-trail-sign(eda,sf-exp1,i-1))]

Note: cf. the note to is-sign-pos 9.6.1.2.2 (186).

(230) lin-util-1(pv) =
    sf-nt1(pv)·exp-sep(pv)·sf-exp1(pv)

(231) exp-sep(pv) =
    is-ω·s-exp-sep(pv) → <>
    T ← <s-exp-sep(pv)>

9.6.3.4 Representation of character pictures

The function edit-char-pic(spec-list,v) merges the character string editing characters, which may occur in the specification list, with the character value list v in the appropriate way.

58 9. DATA, OPERATIONS AND CONVERSIONS
The instruction **test-char-pic(eda, v)** checks this modified character string value v against the character picture attribute eda. If v matches eda, the unchanged value is returned; otherwise, the CONVERSION condition is raised and the representation is retried.

(232) edit-char-pic(spec-list, v) =
    is-<>(spec-list) → <>
    is-char-ed-spec<head(spec-list) —
        <head(spec-list)>"edit-char-pic(tail(spec-list), v)
    T → <head(v)>"edit-char-pic(tail(spec-list), tail(v))

for: is-char-spec-list(spec-list) & is-char-val-list(v)

Ref.: is-char-val 9-3(6)

(233) test-char-pic(eda, v) =
    is-char-pic-match(eda, v) → PASS:
    T →
        rep-pic-1(eda, corr-v);
        corr-v:call-conv-cond(v, i0)

where:
    i0 =
        (i1){1 ≤ i ≤ length-eda, 
        is-valid-single-char(elem(i, s-field(eda)), elem(i, v)) &
        (vj)(1 ≤ j < i > is-valid-single-char(elem(j, s-field(eda)), elem(j, v)))}

for: is-char-pic(eda) & is-char-val-list(v)

Ref.: call-conv-cond 10-24(70)
      is-char-val 9-3(6)

(234) is-char-pic-match(eda, v) =
    is-th-eda,
    Et is-valid-single-char(elem(i, s-field(eda)), elem(i, v))
    i:=1

for: is-char-pic(eda) & is-char-val-list(v)

Ref.: is-char-val 9-3(6)
(235) \textit{is-valid-single-char} (spec, v) =
\[ \text{is-\textit{X-CHAR}} (\text{spec}) \lor \text{is-\textit{A-CHAR}} (\text{spec}) \land (\text{is-letter} \lor \text{is-BLANK})(v) \lor \text{is-\textit{9-CHAR}} (\text{spec}) \land (\text{is-digit} \lor \text{is-BLANK})(v) \lor \text{is-char-ed-spec} (\text{spec}) \]
\textit{for:} is-char-spec (spec) \& is-char-val (v)

Ref.: is-letter 9-4(9)
is-digit 9-4(9)
is-char-val 9-3(6)

9.6.3.5 \textit{Conversion from numeric picture to character string}

The definitions collected here are used in numeric to character conversion (cf. 9.5.1.7).

(236) \textit{num-pic-char-conv} (eda, v) =
\[ \text{is-bin-pic} (eda) \rightarrow \]
\[ \text{pass-bit-char-conv} (v-1); \]
\[ v-1 : \text{rep-pic-1} (eda, v) \]
\[ T \rightarrow \text{rep-pic-1} (eda, v) \]
\textit{for:} is-num-pic(eda) \& is-num-val(v)

Ref.: bit-char-conv 9-35(138)
is-num-val 9-3(3)

(237) \textit{out-fix-pic} (m, n) =
\[ \text{pic} \times \text{out-fix-pic} (m, n) \]

(238) \textit{xout-fix-pic} (m, n) =
\[ \mu_0 (\text{\leq\textit{mode:REAL}}, \text{\leq\textit{mt}_1: \mu (\text{\leq\textit{spec-list-mt}_1} ; \text{\leq\textit{s-dr-beg}_1}, \text{\leq\textit{s-dr-end}_1}, \text{\leq\textit{dr-end}_1}, \text{\leq\textit{mt-unit}_1} ); \text{\leq\textit{dr-end}_1}, \text{\leq\textit{mt-unit}_1})} \]
\textit{where:}
\[ \text{spec-list-mt}_1 = \text{\leq\textit{LIST}} \quad (i = 1 \rightarrow \text{\leq\textit{MINUS}}, \]
\[ n \neq 0 \land i = n \rightarrow \text{\leq\textit{POINT}}, \]
\[ T \rightarrow \text{\leq\textit{9-CHAR}} \]
\[ \text{dr-end}_1 = (n = 0 \rightarrow m - 1, \]
\[ T \rightarrow m - n - 2) \]
\[ \text{mt-unit}_1 = (n = 0 \rightarrow Q, \]
\[ T \rightarrow m - n - 1) \]

Note: \textit{xout-fix-pic (m,n)} is a fixed-point picture eda with \textit{lgth-mt}_1 = m, g-eda1 = n, and a drifting MINUS.
(239) \texttt{cut-fit-pic(v)} = \\
\text{where:} \\
\text{\texttt{spec-list-nt}} = \text{LIST} (i = 1 \rightarrow \text{MINUS}, \\
i = 3 \rightarrow \text{DCINT}, \\
T \rightarrow 9\text{-CHAR}) \\
\text{\texttt{spec-list-exp}} = \text{LIST} (i = 1 \rightarrow \text{SIGN}, \\
i = 1 \rightarrow \text{9-CHAR}) \\

9.6.4 VALUE OF A PICTURE

The function \texttt{pic-val(eda,v)} yields a numeric value for numeric pictures, a character string value for character pictures. It per- forms the inverse steps of \texttt{reg-pic}, and the greater part of its definition is given implicitly. In order that \texttt{pic-val(eda,v)} be defined, \texttt{eda} and the string value \texttt{v} must match, i.e., they must satisfy the predicate \texttt{is-pic-match(eda,v)} defined in 9.6.4.2.

\textbf{Metavariablen}

\texttt{v} \hspace{1cm} \texttt{is-char-val-list} \hspace{0.2cm} \texttt{v} \hspace{0.2cm} \texttt{is-fit-val-list}

\textbf{Abbreviations}

\texttt{for: (is-xnum-pic & is-CPLX*s-mode)(eda)}:

\texttt{v-real} = \text{LIST} \hspace{0.2cm} \text{elem}(i,\texttt{v}) \hspace{1cm} \text{first half of string representing a complex value}

\texttt{v-imag} = \text{LIST} \hspace{0.2cm} \text{elem}(i,\texttt{v}) \hspace{1cm} \text{second half of string representing a complex value}
(240) \[ \text{pic-val(eda,v) =} \]
\[ \quad \text{is-char-pic(eda) \rightarrow} \]
\[ \quad (\text{is-char-pic-match(eda,v) \rightarrow v}, \]
\[ \quad T \rightarrow \text{error}) \]
\[ T \rightarrow \text{num-pic-val(xpic(eda),v)} \]

\texttt{fcr:is-correct-pic(eda) \& (is-char-val-list v \& is-bit-val-list)(v)}

\textbf{Ref.:} \quad \text{is-char-val 9-3(6)}
\quad \text{is-bit-val 9-4(11)}

\textbf{Note:} In case of numeric picture, eda is transformed into explicit form.

\textbf{9.6.4.1 Value of a numeric picture}

\textbf{Metavariables}

\texttt{eda} \quad \text{is-xnum-pic}
\texttt{x} \quad \text{is-real-val}

(241) \[ \text{num-pic-val(eda,v) =} \]
\[ \quad \text{is-CPLX-is-mode(eda) \rightarrow} \]
\[ \quad \text{cplx(num-pic-val(eda-real\_x,v-real\_x),num-pic-val(eda-real\_x,v-imag\_x))} \]
\[ T \rightarrow \text{num-pic-val(xpic(eda),v)} \]

\texttt{fcr:is-fix-pic(eda) \& (is-fix-match(eda,m,v), \& is-norm-subf(eda,sf-mt\_x,m) \&}
\[ x = m . \text{base-eda} \_1 \uparrow (q-eda\_1 - q-eda\_1))} \]

\texttt{fcr:is-flt-pic(eda) \& (is-fix-match(eda,m,v), \& is-norm-subf(eda,sf-mt\_x,m) \&}
\[ x = m . \text{base-eda} \_1 \uparrow (n - q-eda\_1))} \]

\textbf{Note:} cf. the definitions of \texttt{rep-num-pic-1} 2.6.3.1 (206) and \texttt{rep-real-pic-1} 2.6.3.1 (208). The conditions \texttt{is-fix-match} and \texttt{is-flt-match} guarantee that \texttt{x} is an inverse of \texttt{v}, i.e., that \texttt{v} represents \texttt{x}. The conditions \texttt{is-norm-subf} additionally guarantee that \texttt{x} is unique.

(242) \[ \text{is-fix-match(eda,m,v) =} \]
\[ \quad \text{-is-subf-size-cond(eda,sf-mt\_x,m) \&}
\[ v = \text{lir-fix(eda,rep-valid-subf(eda,sf-mt\_x,m))}\]

\texttt{for:is-fix-pic(eda)}
(243) \textit{is-flt-match}(eda, n, v) = \\
\quad \sim \textit{is-subf-size-cond}(eda, sf-\textit{mt}, m) \land \sim \textit{is-subf-size-cond}(eda, sf-\text{exp}, n) \land \\
\quad v = \textit{lin-flt}(eda, \text{rep-valid-subf}(eda, sf-\textit{mt}, m), \text{rep-valid-subf}(eda, sf-\text{exp}, n))

\text{for: is-flt-pic}(eda)

(244) \textit{is-norm-subf}(eda, sf, m) = \\
\quad \sim \textit{is-signed-subf}(eda, sf) \lor m \leq 0

(245) \textit{is-signed-subf}(eda, sf) = \\
\quad \textit{is-2-compl-pic}(eda) \lor \textit{is-1-compl-pic}(eda) \lor (3e)(\textit{is-signed-pos}(eda, sf, v))

9.6.9.2 Testing of string values against picture attributes

The predicate \textit{is-pic-match}(eda, v) can be applied to arbitrary pictures \textit{is-correct-pic}(eda). It combines the predicates \textit{is-char-pic-match}, \textit{is-fix-match}, \textit{is-flt-match} used by the function \textit{pic-val}.

(246) \textit{is-pic-match}(eda, v) = \\
\quad \textit{is-char-pic}(eda) \land \textit{is-char-pic-match}(eda, v) \\
\quad \land \textit{is-num-pic-match}(xpic(eda), v)

\text{for: is-correct-pic}(eda)

(247) \textit{is-num-pic-match}(eda, v) = \\
\quad \textit{is-CELL*mode}(eda) \longrightarrow \\
\quad \textit{is-num-pic-match}(eda-\text{real}, v-\text{real}) \land \textit{is-num-pic-match}(eda-\text{real}, v-\text{imag})

\text{for: is-2-compl-pic}(eda)

\text{for: is-flt-pic}(eda) \rightarrow (3e)(\textit{is-fix-match}(eda, m, v))

\text{for: is-xnum-pic}(eda)

9.6.5 Functions on picture attributes

The function \textit{mk-arith-pic}(eda) transforms a numeric picture attribute eda into arithmetic attribute. The function \textit{mk-str-1}(eda) (cf. 9.5.3.5 (162)) transforms a string class attribute into a string attribute, only its subfunction \textit{pic-length}(eda) remains to be defined.
(248) \texttt{tk-arith-pic(eda) =}
\[ p_0(\langle s\text{-}mode: s\text{-}mode(eda)\rangle, \langle s\text{-}base: base_1 \rangle, \langle s\text{-}scale: scale_1 \rangle, \]
\[ \langle s\text{-}prec: pic\text{-}prec(eda)\rangle, \langle s\text{-}scale\text{-}f: scale\text{-}f_1 \rangle) \]

\textbf{where:}
\begin{align*}
base_1 &= (\text{is-bin-pc}(eda) \rightarrow \text{BIN}, \\
&\quad T \rightarrow \text{DEC}) \\
scale_1 &= (\text{is-flt-pc}(eda) \rightarrow \text{FLT}, \\
&\quad T \rightarrow \text{FIX}) \\
scale\text{-}f_1 &= (\text{is-flt-pc}(eda) \rightarrow \text{U}, \\
&\quad T \rightarrow g\text{-}eda_1 - g\text{-}eda_1) \\
\end{align*}

(249) \texttt{pic\text{-}prec(eda) =}
\[ \text{is-xsterling-pc}(eda) \rightarrow nd\text{-}pd_1 + 3 + g\text{-}eda_1 \]
\[ T \rightarrow nd\text{-}mt_1 \]

\textbf{Note:} For sterling pictures, the shillings field and integral pence field are counted as if containing together three digit positions. Hence, in general, \( p\text{-}eda_1 \neq nd\text{-}mt_1 \) for sterling pictures.

(250) \texttt{pic\text{-}length(eda) =}
\[ \text{is-char-pc}(eda) \rightarrow \text{length}\text{-}s\text{-}field(eda) \]
\[ \text{is\text{-}CFLX\text{-}s\text{-}mode}(eda) \rightarrow 2 \cdot \text{pic\text{-}length(eda\text{-}real_1)} \]
\[ \text{is\text{-}fix\text{-}pc}(eda) \rightarrow \text{length}\text{-}sf\text{-}mt_1(eda) \]
\[ \text{is\text{-}flt\text{-}pc}(eda) \rightarrow \text{length}\text{-}lin\text{-}flt\text{-}1(eda) \]

\textbf{for:is\text{-}correct-pc(eda)
10. ATTENTIONS AND CONDITIONS

This chapter defines the parts of PL/I which are common in dealing with conditions and attentions. Due to the heterogeneous purposes served by the various types of conditions - computational, input/output, program-checkout, list-processing, programmer named, system-action, and attention - the situations which result in a condition raising partially are described in the respective chapters of this document.

While some conditions always are enabled, others under control of condition prefixes in the program can be enabled or disabled, as described in chapter 10.1.1 and still others - the attentions - may be enabled or disabled only by statements, as treated in 10.1.2.

For each condition the language specifies a specific condition action - the PL/I standard system action - which is performed if a condition is raised and an interrupt of the program occurs. This action can be modified for each condition by executing on and revert statements as described in chapter 10.2.

The activation of attentions is either done asynchronously or by an access statement and result in raising the attention condition, which is defined in chapter 10.3.

The raising and execution of a condition action has some analogy to the execution of a call to a parameterless procedure. The raising may also be done by a signal statement. This is defined in chapter 10.4.

Chapter 10.5 defines the updating of the condition builtin function part of CS used for interpreting condition builtin functions in an cm-unit.

Features

<table>
<thead>
<tr>
<th>Feature</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>an</td>
<td>attention directory</td>
</tr>
<tr>
<td>at</td>
<td>attribute directory</td>
</tr>
<tr>
<td>ca</td>
<td>condition action</td>
</tr>
<tr>
<td>cbif</td>
<td>components needed for updating the condition builtin function part</td>
</tr>
<tr>
<td>cd1</td>
<td>evaluated condition list</td>
</tr>
<tr>
<td>ccnd</td>
<td>evaluated condition</td>
</tr>
<tr>
<td>cs</td>
<td>condition state</td>
</tr>
<tr>
<td>dm</td>
<td>denotation directory</td>
</tr>
<tr>
<td>en</td>
<td>attention enabling state</td>
</tr>
<tr>
<td>ev</td>
<td>attention environment directory</td>
</tr>
<tr>
<td>gen</td>
<td>generation</td>
</tr>
<tr>
<td>list</td>
<td>list</td>
</tr>
<tr>
<td>onl, ncl</td>
<td>prefix condition list</td>
</tr>
</tbody>
</table>
ABSTRACT SYNTAX AND INTERPRETATION OF PL/I

0.1 ENABLING AND DISABLING

0.1.1 ENABLING AND DISABLING OF CONDITIONS

Several PL/I on-conditions can be enabled or disabled under control of condition prefixes. Condition prefixes control the enabling and disabling of conditions in a static scope. As identifiers appear in some condition prefixes, a dynamic interpretation of prefixes is necessary to ensure unambiguity of reference. The interpretation of condition prefixes establishes two components of the condition state C_S, the block prefix part and the statement prefix part.

At the beginning of the program interpretation a standard enabling status exists in the initial state of C_S.

10. ATTENTIONS AND CONDITIONS
During the interpretation of PL/I statements the enabling status as defined for the block can be modified by explicit statement condition prefixes.

The enabling and disabling of attentions is controlled for each task by the enable or disable statements, respectively. Information necessary for all tasks is stored in the global directory AN under the specific attention identification, whereas the attentions a task has enabled or associated, with or without the event option, are kept in the task local state component IN.
Metavariables

cd1 is-eattn-cond-list evaluated attention condition list
cond is-eattn-cond evaluated attention condition

10.1.2.1 Enable statement

The interpretation distinguishes between enabling with or without event option and thereby makes entries into the different components of $E_N$. The attention directory $A_N$ is updated with every enable statement.

When all the attentions of the enable statement have been tried to enable, then the inspection of the $e$-wait-list component of $E_N$ decides whether the task has to wait or may continue. Events for which all attentions are successfully enabled are immediately deleted.

When the enabling mode is changed to asynchronous by the enable statement and the info-list of the relevant attention in $A_N$ is not empty, an asynchronous interrupt is immediately executed.

The attentions as described in the abstract syntax of an enable or disable statement are evaluated and satisfy the predicate is-eattn-cond. The identification part of this evaluated condition is new comparable with the identification part of the incoming attention which describes the attention occurrence outside the PL/I machine.

(7) \[ \text{int-enable-st}(t) = \]

\[ \text{int-enable-1}(e\text{-list}(t)) \]

for:is-enable-st(t)
(8) \[ \text{int-enable-1}(\text{enable-list}) = \]
\[ -\text{is-0*event-head(enable-list)} \rightarrow \]
\[ \text{int-enable-1}(\text{tail(enable-list)}); \]
\[ \text{int-enable-2}(\text{tn,head(enable-list),cdl}); \]
\[ \text{attach-event}(\text{gen,tn}); \]
\[ \text{cdl:pass-eval-attn-list(idrl,EN,FV)}; \]
\[ \text{gen:eval-ref-qen(s-event-head(enable-list))}; \]
\[ \text{tn:un-name} \]
\[ -\text{is-<>(enable-list)} \rightarrow \]
\[ \text{int-enable-1}(\text{tail(enable-list)}); \]
\[ \text{int-enable-2}(\text{tn,head(enable-list),cdl}); \]
\[ \text{cdl:pass-eval-attn-list(idrl)} \]
\[ \text{is-<>(enable-list) \& -is-<>(s-wait-list(FW))} \rightarrow \text{enab-wait} \]
\[ T \rightarrow \text{null} \]

where:
\[ \text{idrl} = s\text{-attn-list}\_s\text{-cond}\_head(\text{enable-list}) \]

for:is-enable-list(\text{enable-list})

Ref.: \[ \text{eval-ref-qen 8-28(62)} \]
\[ \text{un-name 3-10(20)} \]

(9) \[ \text{int-enable-2}(\text{tn,enable,cdl}) = \]
\[ \text{is-0(attn)} \lor \text{s-task(attn)} = \text{TN} \rightarrow \]
\[ \text{int-enable-2}(\text{tn,enable,tail(cdl)}); \]
\[ \text{enable(\text{enable,head(cdl)})} \]
\[ -\text{-is-<>(cdl)} \rightarrow \]
\[ \text{int-enable-2}(\text{tn,enable,tail(cdl)}); \]
\[ \text{associate(tn,enable,head(cdl)}) \]
\[ \text{is-<>(cdl)} \& \text{-is-0(tn)} \& (\text{-is-<>t\_t(FV) \lor is-0\_t(FW)}) \rightarrow \text{delete-task-event(tn,NORMAL)} \]
\[ T \rightarrow \text{null} \]

where:
\[ \text{attn} = s\text{-ident(head(cdl))} \]

for:is-n(tn) \lor is-0(tn),is-enable(\text{enable})

Ref.: \[ \text{delete-task-event 5-8(28)} \]
(10) \textbf{enable}(enable, \text{cond}) = \\
\begin{align*}
\text{is-\text{enable}}(\text{attn}_1) & \rightarrow \\
\text{s-en}:\mu(\text{FIN};\text{s-enab-list}:\text{s-enab-list}(\text{FIN}) \cdot \text{cond}) \\
\text{s-spec}(\text{enable}) & = \text{s-spec}(\text{attn}_1) \rightarrow \text{null} \\
\text{is-ACC:s-spec}(\text{attn}_1) & \& \text{is-ASYN:s-spec}(\text{enable}) & \& \text{is-\text{<} >>s-info}(\text{attn}_1) & \rightarrow \\
\text{s-an}:\mu(\text{FIN};\text{s-spec-ident}_1:\text{s-spec}(\text{enable})) \\
\text{s-en}:\mu(\text{FIN};\text{tn}:\text{prep-attn-1}(\text{FIN}, \text{cond})) \\
\text{T} & \rightarrow \text{s-en}:\mu(\text{FIN};\text{s-spec-ident}_1:\text{s-spec}(\text{enable})) \\
\end{align*}

where:
\begin{align*}
\text{id} & \text{ent}_1 = \text{s-ident}(\text{cond}) \\
\text{attn}_1 & = \text{id} \text{ent}_1(\text{FIN}) \\
\end{align*}

for: \text{is-\text{enable}}(enable)

\text{Ref.}: \quad \text{prep-attn-1 10-13}(39)

(11) \textbf{associate}(\text{tn}, \text{enable}, \text{cond}) = \\
\begin{align*}
\text{is-0}(\text{tn}) & \& \text{is-\text{enab-list}}(\text{FIN}) & \text{tn} \neq \text{FIN} & \rightarrow \\
\text{s-an}:\mu(\text{FIN};\text{id} \text{ent}_1: \text{id} \text{ent}_0(\text{s-assoc-list}:\text{s-assoc-list}(\text{FIN}) \cdot \text{cond}_1), (\text{tn}:\text{tn}(\text{FIN}) \cdot \text{cond})) \\
\text{s-en}:\mu(\text{FIN};\text{s-assoc-list}:\text{s-assoc-list}(\text{FIN}) \cdot \text{cond}_1), (\text{tn}:\text{tn}(\text{FIN}) \cdot \text{cond})) \\
\end{align*}

\begin{align*}
\text{is-0}(\text{tn}) & \& \text{is-\text{enab-list}}(\text{FIN}) & \text{tn} \neq \text{FIN} & \rightarrow \\
\text{s-an}:\mu(\text{FIN};\text{id} \text{ent}_1: \text{id} \text{ent}_0(\text{s-assoc-list}:\text{s-assoc-list}(\text{FIN}) \cdot \text{cond}_1), (\text{tn}:\text{tn}(\text{FIN}) \cdot \text{cond})) \\
\text{s-en}:\mu(\text{FIN};\text{s-assoc-list}:\text{s-assoc-list}(\text{FIN}) \cdot \text{cond}_1), (\text{tn}:\text{tn}(\text{FIN}) \cdot \text{cond})) \\
\end{align*}

\begin{align*}
\text{T} & \rightarrow \text{s-en}:\mu(\text{FIN};\text{s-assoc-list}:\text{insert-list}(\text{s-assoc-list}(\text{FIN}), \text{cond}_1, 0)), (\text{tn}:\text{tn}(\text{FIN}) \cdot \text{cond})) \\
\end{align*}

where:
\begin{align*}
\text{id} & \text{ent}_1 = \text{s-ident}(\text{cond}) \\
\text{attn}_1 & = \text{id} \text{ent}_1(\text{FIN}) \\
\text{cond}_1 & = \mu_0(\text{s-ident}(\text{cond}), \text{s-spec}(\text{enable}), \text{tn}:\text{tn}(\text{FIN})) \\
\end{align*}

for: \text{is-\text{0}}, \text{is-\text{0}}(\text{tn}), \text{is-\text{enable}}(enable)

(12) \textbf{insert-list}(\text{list}, \text{cond}, \text{opt}) = \\
\begin{align*}
\text{s-ident}(\text{head}(\text{list})) & = \text{s-ident}(\text{cond}) \rightarrow \text{merge-cond}(\text{cond}, 0, \text{opt}) \cdot \text{tail}(\text{list}) \\
\text{T} & \rightarrow \text{insert-1}(\text{head}(\text{list}), \text{tail}(\text{list}), \text{cond}, \text{opt}) \\
\end{align*}

for: \text{is-\text{opt}}(\text{opt})
(13) \[
\text{insert-1}(\text{list-1}, \text{list-2}, \text{cond}, \text{opt}) = \\
\ s\text{-ident}(\text{head}(\text{list-2})) = \ s\text{-ident}(\text{cond}) \rightarrow \\
\text{list-1}^*\text{merge-cond}(\text{cond}, \text{head}(\text{list-2}), \text{opt}) \text{tail}(\text{list-2}) \\
\neg\text{is-<>}(\text{list-2}) \rightarrow \text{insert-1}(\text{list-1}^*<\text{head}(\text{list-2})>, \text{tail}(\text{list-2}), \text{cond}, \text{opt}) \\
T \rightarrow \text{list-1} \\
\] 
\text{for:is-cpt}(\text{cpt})

(14) \[
\text{merge-cond}(\text{cond-1}, \text{cond-2}, \text{opt}) = \\
\text{is-<>}(\text{cpt}) \rightarrow <> \\
\text{is-<>s-tn}(\text{cond-2}) \rightarrow \text{<cond-1>} \\
T \rightarrow \\
<\mu_0(\langle s\text{-ident}:s\text{-ident}(\text{cond-1})>, s\text{-spec}:s\text{-spec}(\text{cond-1})>, \\
\langle s\text{-tn}:s\text{-tn}(\text{cond-1}) \cup s\text{-tn}(\text{cond-2})\rangle) \\
\] 
\text{for:is-cpt}(\text{opt})

(15) \[
\text{eval-attn-list}(\text{idrl}, \text{dn}, \text{ev}) = \\
\text{is-<>}(\text{idrl}) \rightarrow <> \\
T \rightarrow \langle \text{eval-attn}(\text{head}(\text{idrl}), \text{dn}, \text{ev})\rangle \text{eval-attn-list}(\text{tail}(\text{idrl}), \text{dn}, \text{ev}) \\
\] 
\text{for:is-id-ref-list}(\text{idrl})

(16) \[
\text{eval-attn}(\text{idr}, \text{dn}, \text{ev}) = \\
\mu_0(\langle s\text{-ident}:\text{mk-ident}(s\text{-id} (\text{idr}), s\text{-n}(\text{idr}) (\text{dn}) (\text{ev}))\rangle) \\
\] 
\text{for:is-id-ref}(\text{idr})

(17) \[
\text{mk-ident}(\text{id}, \text{ea}) = \\
\] 
\text{for:is-id}(\text{id}), \text{is-ea}(\text{ea})

Ref.: \text{is-ea 71-21(58)}

Note: Implementation defined selector value, identifying an attention.

(18) \[
\langle \text{vid}, \text{ea}\rangle (\text{is-ident}\text{*mk-ident}(\text{id}, \text{ea})) \\
\] 
Ref.: \text{is-ident 3-18(69)}
10. ATTENTIONS AND CONDITIONS

10.1. Disable statement

The disable statement disables or disassociates the attentions from the task and possibly enables them in another task, thereby it may delete events or execute asynchronous interrupts. Again the attentions in the disable statement are evaluated as in the enable statement.

(22) int-disable-st(t) =
    int-disable-1(cal);
    cal:pass-eval-attn-list(s-attn-list*s-cond(t))

for:is-disable-st(t)
(23) \( \text{int-disable-1}(cdl) = \)
\[ \text{is-}\langle\rangle (cdl) \rightarrow \text{null} \]
\[ \text{s-task}(\text{attr}_1) = \text{tm} \rightarrow \]
\[ \text{int-disable-1}(\text{tail}(cdl)); \]
\[ \text{disable}(\text{head}(cdl)) \]
\[ (\exists \text{tn}) (\text{tn} \in \text{s-assoc}(\text{attr}_1) \land \text{tn} = \text{tm}) \rightarrow \]
\[ \text{int-disable-1}(\text{tail}(cdl)); \]
\[ \text{disassociate}(\text{head}(cdl)) \]
\[ \text{tn} \rightarrow \text{int-disable-1}(\text{tail}(cdl)) \]

where:
\[ \text{attr}_1 = \text{s-ident}(\text{head}(cdl)) \]

(24) \( \text{disable}(\text{ccnd}) = \)
\[ \text{is-}\langle\rangle * \text{s-assoc}(\text{attr}_1) \rightarrow \]
\[ \text{s-an}: \mu (\text{AN}; \text{attr}_1) \]
\[ \text{s-en}: \mu (\text{EN}; \langle\rangle ); \text{s-enab-list: insert-list}(\text{s-enab-list}(\text{FP}), \text{ccnd}, \ast)) \]
\[ \text{is-}\langle\rangle * \text{s-infc}(\text{attr}_1) \land \text{s-spec}(\text{cond}) = \text{SYN} \rightarrow \]
\[ \text{s-an}: \mu (\text{AN}; \text{attr}_1) \]
\[ \text{s-en}: \mu (\text{EN}; \langle\rangle ); \text{s-enab-list: insert-list}(\text{s-enab-list}(\text{FP}), \text{ccnd}, \ast)) \]
\[ \text{s-ta}: \text{activate-tasks}(\mu (\text{PA}; <\text{tn}_1: \text{prep-attr}\{\text{tn}_1, \text{ccnd}\}>)) \]
\[ \text{s-c}: \text{null}; \]
\[ \text{delete-task-event}(\text{tn}, \text{NORMAL}) \mid \text{tn} \in \text{s-tr}(\text{ccnd}) \land \text{is-}\langle\rangle * \text{tn}(\text{en}_1)) \]
\[ \text{tn} \rightarrow \]
\[ \text{s-an}: \mu (\text{AN}; \text{attr}_1) \]
\[ \text{s-en}: \mu (\text{EN}; \langle\rangle ); \text{s-enab-list: insert-list}(\text{s-enab-list}(\text{FP}), \text{ccnd}, \ast)) \]
\[ \text{s-ta}: \text{activate-tasks}(\mu (\text{PA}; <\text{s-en}\text{tn}_1: \text{en}_1>) \]
\[ \text{s-c}: \text{null}; \]
\[ \text{delete-task-event}(\text{tn}, \text{NORMAL}) \mid \text{tn} \in \text{s-tr}(\text{ccnd}) \land \text{is-}\langle\rangle * \text{tn}(\text{en}_1)) \]

where:
\[ \text{ident}_1 = \text{s-ident}(\text{cond}) \]
\[ \text{attr}_1 = \text{ident}_1 (\text{AN}) \]
\[ \text{tn}_1 = \text{head-order-set}(\text{s-assoc}(\text{attr}_1), \text{DISABLE}) \]
\[ \text{cond}_1 = \langle \text{tn}, \text{x} = \text{elem}(\text{i}, \text{s-assoc-list}(\text{s-en}\text{tn}_1 (\text{FP}))) \rangle \land \]
\[ \text{s-ident}(\text{x}) = \text{s-ident}(\text{cond}) \]
\[ \text{task}_1 = \mu (\text{tn}_1 (\text{PA}); <\text{s-en}\text{en}_1>) \]
\[ \text{en}_1 = \text{s-en}(\text{task}_1) \]
\[ \text{en}_2 = \mu (\text{en}_2); <\text{s-en}\text{en}_2> \rightarrow \text{ccnd}> \]
\[ <\text{s-assoc-list: insert-list}(\text{s-assoc-list}(\text{en}_2), \text{cond}, \ast)>, \]
\[ <\text{s-wait-list: insert-list}(\text{s-wait-list}(\text{en}_2), \text{ccnd}, \ast)>, \]
\[ <\text{tn}\text{insert-list}(\text{tn}(\text{en}_2), \text{cond}, \ast)>, \]
\[ \text{en}(\text{tn}(\text{en}_2), \text{cond}, \ast) > \mid (\text{tn} \in \text{s-tr}(\text{ccnd})) \]
\[ \text{an}_1 = \mu (\text{AN}; <\text{s-spec-ident}_1: \text{s-spec}(\text{cond}_1)> , <\text{s-task}\text{ident}_1: \text{tn}_1>, \]
\[ <\text{s-assoc-ident}_1: \text{s-assoc}(\text{attr}_1) - \{\text{tn}_1\}>) \]

Ref.: activate-tasks 5-12(42) 
prep-attr 1 11-14(39) 
delete-task-event 5-8(28) 
order-set 11-40(111)
10.2 CONDITION ACTION

Condition actions are the actions which have to be performed if in the interpretation of a PL/I program a condition situation occurs, the condition is raised, the condition is enabled and the normal flow of control is interrupted as specified for the specific raising. The action to be executed can either be the standard system action or the action specified by the execution of CR and revert statements.

10.2.1 STANDARD SYSTEM ACTION FOR CONDITIONS

If the interpretation of a program specifies that a condition action is to be executed and no on-unit has been established for the condition in the current CS, the system action defined by syst-cond-exec is executed.

(25) \text{disassociate}(\text{cond}) = \\
\text{s-addr}(<\text{assoc}\cdot\text{ident}_1;\text{assoc}\cdot\text{ident}_1(\text{AN}) - (\text{TP})); \\
\text{s-addr}(<\text{assoc}\cdot\text{list}\cdot\text{insert}\cdot\text{list}(\text{e-assoc}\cdot\text{list}(\text{AN}), \text{cond}, *))> \\

where: \\
\text{ident}_1 = \text{s-ident}(\text{cond})

10.2.1 STANDARD SYSTEM ACTION FOR CONDITIONS

If the interpretation of a program specifies that a condition action is to be executed and no on-unit has been established for the condition in the current CS, the system action defined by syst-cond-exec is executed.

(26) \text{syst-cond-exec}(\text{cond}, \text{cbif}) = \\
is-\text{UL}(\text{cond}) \lor \text{is-STRG}(\text{cond}) \lor \text{is-STRZ}(\text{cond}) \lor \text{is-NAM}(\text{ccnd}(\text{cond})) \lor \\
is-\text{proc}-\text{named}\cdot\text{cond}(\text{cond}) -- \\
\text{ccment}(\text{cond}, \text{cbif}) \\
is-\text{IND}(\text{ccnd}(\text{cond})) \lor \text{is-SIGNAL}(\text{ccnd}(\text{cond})) \lor \\
is-\text{INT}(\text{ccnd}(\text{cond})) \lor \text{is-FIND}(\text{ccnd}(\text{cond})) \lor \text{is-FIND}(\text{ccnd}(\text{cond})) \\
is-\text{FINISH}(\text{ccnd}(\text{cond})) -- \\
\text{null} \\
is-\text{IND}(\text{ccnd}(\text{cond})) -- \text{syst-endpage-exec}(s-f(\text{cond})) \\
is-\text{CCNV}(\text{ccnd}(\text{cond})) \lor \text{is-IO}(\text{ccnd}(\text{cond})) \lor \text{is-FD}(\text{ccnd}(\text{cond})) \lor \\
is-\text{SIZE}(\text{ccnd}(\text{cond})) \lor \text{is-ZERO}(\text{ccnd}(\text{cond})) \lor \text{is-AREA}(\text{ccnd}) -- \\
call-ccnd-1(\text{ERROR}, \text{cbif}); \\
\text{ccment}(\text{cond}, \text{cbif}) \\
is-\text{check}(\text{cond}) -- \text{syst-check-exec}(\text{cond}) \\
is-\text{ERROR}(\text{cond}) -- \text{syst-error-exec} \\

where: \\
\text{cbif} = \mu(\text{cbif};<\text{s-cond};\text{cond}>)

Ref.: 
is-\text{eattn}\cdot\text{cond} 3-19(72) 
\text{syst-endpage-exec} 11-123(332) 
call-ccnd-1 10-16(55) 
is-\text{check} 3-21(77) 
\text{syst-check-exec} 11-123(331) 

(27) \text{comment}(\text{cond}, \text{cbif}) = \\
\text{s-addr}(\text{AN}; <\text{comment}\cdot\text{ident}: \text{comment}(\text{AN}, \text{cond}(\text{cbif}))> \\

10. ATTENITIONS AND CONDITIONS
(28) \[ \text{comment-f(\text{cond,cbif,\xi})} = \]

Note: Implementation defined function, whose values satisfy the predicate is-comment.

(29) \( \neg \text{cond,cbif,\xi} \{ \text{is-comment(\text{comment-f(\text{cond,cbif,\xi})})} \} \)

Ref.: is-comment 3-28(103)

(30) \text{Syst-error-exec} =

Note: Implementation defined error system action.

10.2.2 INTERPRETATION OF CN STATEMENT

The interpretation of an on statement for a condition establishes the \text{on-unit} as the new action for that specific condition in the current condition status \(CS\).

(31) \text{int-on-st(t)} =

\[
\begin{align*}
\{ & \{ i \mid 1 \leq i \leq \text{length(s-ref-list}\text{*s-cond(t)}) \} \\
& \text{is-corr-check(elea}(i,\text{s-ref-list}\text{*s-cond(t)}),\text{AT}) \} \rightarrow \\
& \text{cn-establish}\text{-1}(\text{cdl},t); \\
& \text{cdl}\text{:pass-expand-rl(s-ref-list}s\text{-cond(t)},\text{AT}) \\
& \text{is-attn-cond(s-cond(t))} \rightarrow \\
& \text{cn-establish}\text{-1}(\text{cdl},t); \\
& \text{cdl}\text{:pass-eval-attn-list(s-attn-list}s\text{-cond(t)},\text{DN,LV}) \\
& \text{is-named-io-cond(s-cond(t))} \rightarrow \\
& \text{cn-establish}(\text{cond},t); \\
& \text{cond}\text{:eval-named-io-cond(s-cond(t))} \\
& \text{-is-check-cond(s-cond(t)) \&} \\
& \text{(is-progr-named-cond(s-cond(t)) = is-COND(attr_0))} \rightarrow \\
& \text{cn-establish(s-cond(t),t)} \\
& \text{T} \rightarrow \text{error}
\end{align*}
\]

where:

\(\text{attr}_0 = (s-n*s\text{-cond(t)})\) (at)

for: is-on-st(t)

Ref.: is-corr-check 10-2(3) 
expand-rl 10-22(66) 
eval-attn-list 10-7(15)
10. ATTENTIONS AND CONDITIONS
(36) \( \text{int-revert-st}(t) = \)
\[ (\forall i)(1 \leq i \leq \text{length}(s\text{-ref-list} \cdot s\text{-cond}(t)) \rightarrow \]
\( \text{is-corr-check}(\text{elem}(i, s\text{-ref-list} \cdot s\text{-cond}(t)), AT) \) \rightarrow \]
\( \text{revert-on-1}(cdl); \]
\( \text{cdl} : \text{pass-expand-rl}(s\text{-ref-list} \cdot s\text{-cond}(t), AT) \)
\( \text{is-attn-cond}(s\text{-cond}(t)) \rightarrow \]
\( \text{revert-on-1}(cdl); \]
\( \text{cdl} : \text{pass-eval-attn-list}(s\text{-attn-list} \cdot s\text{-cond}(t), DN, AT) \)
\( \text{is-named-io-cond}(s\text{-cond}(t)) \rightarrow \]
\( \text{revert-on}(\text{cond}); \]
\( \text{cond} : \text{eval-named-io-cond}(s\text{-cond}(t)) \)
\( \neg \text{is-check-cond}(s\text{-cond}(t)) \land \)
\( \neg \text{is-pg-named-cond}(s\text{-cond}(t)) \rightarrow \text{is-COND}(attr_0) \) \rightarrow \]
\( \text{revert-cond}(s\text{-cond}(t)) \)
\( T \rightarrow \text{error} \)

where:
\( attr_0 = (\text{e-n}\cdot s\text{-cond}(t))(\text{at}) \)

for: \( \text{is-revert-st}(t) \)

Ref.:
\( \text{is-corr-check} \ 10-2(3) \)
\( \text{expand-rl} \ 10-22(66) \)
\( \text{eval-attn-list} \ 10-7(15) \)

(37) \( \text{revert-on-1}(cdl) = \)
\[ \text{null; \{revert-on(elem(i, cdl)) | 1 \leq i \leq \text{length}(cdl)\}} \]

(38) \( \text{revert-on}(\text{cond}) = \)
\[ s\text{-cs} = \mu CS < (\text{cond-sel}(\text{cond}, AT)) \cdot s\text{-cap} : (\text{cond-sel}(\text{cond}, AT)) \cdot s\text{-cap} \cdot s\text{-cs}(E)> \]

for: \( \text{is-revert-st}(t) \)

Ref.:
\( \text{cond-sel} \ 3-19(74) \)
10.3 ATTENTION ACTIVATION

Metavariables

cdl is-attm-cond-list evaluated attention condition list
cond is-attm-cond evaluated attention condition

10.3.1 ASYNCHRONOUS INTERRUPT

In each step of the computation the attention directory AN is searched for an attention, whose information stack (info-list) was recently enlarged from outside the PL/I machine in the environment step and whose enabling mode is asynchronous. In this case prep-attm-1 is used to activate the asynchronous interrupt in the task the name of which is found in AN.

The selected task is dumped and in its control the attention call is installed. Before executing the condition call the head of the information list is assigned to previously allocated storage and the pointer to this storage is put into chf for later use in the on-unit.

Before the actual condition call is executed the enabling mode is changed from asynchronous to ACC-1, so that this attention call is not interruptable by an asynchronous interrupt of the same attention. After normal or abnormal return the enabling mode is rechanged to the original mode before the call, when it was not changed by the on-unit.

(39) prep-attm-1(task, cond) =
    µ(task;<s-d;d4>,<s-ba:μ>,<s-ei:μ>,<s-free-set: []>,<s-task-set: []>,<s-ci:μ>,<s-c:int-attm (cond))>

where:
    d4 = stack(s-ba(task), s-ei(task), s-cs(task), s-d(task), s-ci(task), s-c(task))

Ref.: stack 6-2(2)

(40) int-attm (cond) =
    epilogue;
    call-attm-cond (cond)

Ref.: epilogue 6-11(31)
ABSTRACT SYNTAX AND INTERPRETATION OF PL/I

(41) **call-attn-cond** (cond) =

```
free(b);
change-enab-1(cond);
call-cond-1(cond,char);
change-enab(cond);
char:pass-attn-char(gen,cond,\$);
install-info(gen,cond);
gen:allocate(b,eva,\$,DUMMY);
end-name
```

where:
```
eva = s-struct*head*s-info(s-ident(cond) (AN))
```

Ref.:  free 7-18(55)
call-cond-1 10-18(55)
allocate 7-11(25)
end-name 3-10(20)

(42) **install-info** (gen,cond) =

```
s-an: \$:(AN:<s-info*ident1:tail*s-info(attn1)>)
s-c:assign(gen,op0)
```

where:
```
ident1 = s-ident(cond)
attn1 = ident1(AN)
op0 = s-op*head*s-info(attn1)
```

Ref.:  assign 8-9(23)

(43) **change-enab** (cond) =

```
is-ASYN*s-spec(attn1) \rightarrow s-an: \$:(AN:<s-spec*ident1:ACC-T>)
T \rightarrow null
```

where:
```
ident1 = s-ident(cond)
attn1 = ident1(AN)
```

(44) **change-enab-1** (cond) =

```
is-ACC-1*s-spec(attn1) & -is-s-info(attn1) \rightarrow
s-an: \$:(AN:<s-spec*ident1:ASYN>)
s-cn: \$:(CN:prep-attn-1(TN(FP),cond))
is-ACC-1*s-spec(attn1) \rightarrow s-an: \$:(AN:<s-spec*ident1:ASYN>)
T \rightarrow null
```

where:
```
ident1 = s-ident(cond)
attn1 = ident1(AN)
```

10. ATTENTIONS AND CONDITIONS
10.3.2 ACCESS STATEMENT

The access statement makes available attention information for processing, either taking one attention of a specified list or an arbitrary one out of the list of enabled attentions in the task. If an attention is enabled in the task, but the attention stack is empty, then the else-unit is taken, or in its absence the task waits until information appears on the stack.

\[(\text{int-access-st}(t) =
\begin{align*}
\text{int-acc-1}(\text{cdl}, t) & ; \\
\text{cdl}: \text{pass-eval-attr-list}(\text{s-attr-list} & \text{s-cond}(t), \text{DN}, \text{AT})
\end{align*}\]

for: is-access-st(t)

Ref.: eval-attr-list 10-7(15)

\[(\text{int-acc-1}(\text{cdl}, t) =
\begin{align*}
\text{is-<>}(\text{cdl}) & \rightarrow \text{int-acc-2}(t) \\
\text{test-attr}(\text{cdl}, \text{AN}, \text{EN}, \text{TN}) & \rightarrow \text{int-acc-3}(\text{cdl}, t)
\end{align*}\]

for: is-access-st(t)

Ref.: is-access-st 3-4(6)

\[(\text{test-attr}(\text{cdl}, \text{an}, \text{en}, \text{tn}) =
\begin{align*}
\text{tn}_1 & = \text{tn} \& \text{s-spec(\text{attn}_1)} = \text{ASYN} \lor \text{s-spec(\text{cond}_1)} = \text{ASYN} \rightarrow \text{error} \\
\text{-is-<>} \rightarrow \text{test-attr(tail(\text{cdl}), an, en, tn)} \\
\text{T} & \rightarrow \text{T}
\end{align*}\]

where:
\begin{align*}
\text{attn}_1 & = \text{s-ident(head(cdl))} (\text{an}) \\
\text{tn}_1 & = \text{s-task(\text{attn}_1)} \\
\text{cond}_1 & = (\forall x)(x = \text{elem}(i, \text{s-assoc-list(\text{en}))} \& \text{s-ident(x)} = \text{s-ident\text{*head(cdl)}}
\end{align*}

for: is-tn(\text{tn})

Ref.: is-tn 3-4(6)
\[(49)\]
\[
\text{int-acc-2}(t) = \begin{cases}
  \neg \text{is-<>s-enab-list}(\text{EN}) \land \\
  (\exists i) (1 \leq i \leq \text{length}\text{-s-enab-list}(\text{EN}) \land s\text{-spec}\text{-elem}(i,\text{s-enab-list}(\text{EN})) = \text{ACC}) \rightarrow \\
  \text{int-acc-3}(\text{cdl}_1, t)
\end{cases}
\]
\[
T \rightarrow \text{error}
\]

where:
\[
\text{cdl}_1 = \text{order-set}\{(\text{elem}(i, \text{s-enab-list}(\text{EN})) \mid 1 \leq i \leq \text{length}\text{-s-enab-list}(\text{EN}) \land s\text{-spec}\text{-elem}(i, \text{s-enab-list}(\text{EN})) = \text{ACC}, \text{ACC}\}
\]

for:is\text{-access-st}(t)

Ref.: order-set 11-40(111)

\[(50)\]
\[
\text{int-acc-3}(\text{cdl}, t) = \begin{cases}
  \text{is-eattn-cond}\left(\text{select-cond}(\text{cdl}, \text{AN, TN})\right) \rightarrow \text{call-eattn-cond}\left(\text{select-cond}(\text{cdl}, \text{AN, TN})\right) \\
  \neg \text{is-<>s-else}(t) \rightarrow \text{stack-ci}(F, t)
\end{cases}
\]
\[
T \rightarrow \text{wait-acc}(\text{cdl})
\]

for:is\text{-access-st}(t)

Ref.: is-eattn-cond 3-19(72)
\text{stack-ci} 6-39(108)

Note: The instruction \text{stack-ci} interprets the else unit. It is necessary to use this instruction here to handle also goto statements within the on-unit correctly.

\[(51)\]
\[
\text{select-cond}(\text{cdl}, \text{ar, tn}) = \begin{cases}
  \text{s-task}(\text{attn}_1) = \text{tn} \land s\text{-spec}(\text{attn}_1) = \text{ACC} \land \neg \text{is-<>s-info}(\text{attn}_1) \rightarrow \text{head}(\text{cdl}) \\
  \neg \text{is-<>}(\text{tail}(\text{cdl})) \rightarrow \text{select-cond}(\text{tail}(\text{cdl}), \text{ar}, \text{tn})
\end{cases}
\]
\[
T \rightarrow \text{F}
\]

where:
\[
\text{attn}_1 = (s\text{-ident}\text{-head}(\text{cdl}))(\text{ar})
\]

for:is\text{-tn}(\text{tn})

Ref.: is\text{-tn} 3-4(6)
10. ATTENTIONS AND CONDITIONS

10.4 CONDITION ACTIVATION

If the interpretation of a program arrives in certain situations the language specifies that a condition is to be raised. Due to the heterogeneous sources of condition raising some of the condition raising actions will be described in the appropriate sections of the language definition and this chapter will give auxiliary instructions used for the condition activation.

10.4.1 INTERPRETATION OF THE CONDITION CALL

The instruction call-cond-1 is the general form of a condition call, which is used when a chif argument is required for updating the condition builtin function part of C\(2\). In the other cases the instruction call-cond is used.

The instruction call-cond-2 inspects the condition status for an appropriate condition action. If none is present or the on-unit is SYSTEM, syst-cond-exec is used to execute the system action, otherwise int-cond is used to execute the condition action.

(54) call-cond (cond) =
        call-cond-1 (cond, µ_0 (<s-oncode=s-chif:code (cond, E)>))
for: (is-prefix-cond a -is-check-cond v is-AREA v is-ERROR v is-FINISH) (cond)

Ref.: code 10-27(80)
(55) \[\text{call-condition-1}(\text{cond}, \text{cbif}) =\]
\[\neg \text{is-FOFL}(\text{cond}) \land \text{is-OFL}(\text{cond}) \land \text{is-SIZI}(\text{cond}) \land \text{is-SUBRG}(\text{cond}) \land \neg \text{is-Z~IV}(\text{cond}) \Rightarrow \text{error};\]
\[\text{call-condition-2}(\text{cond}, \text{cbif})\]
\[\neg \text{is-FOFL}(\text{cond}) \land \text{is-OFL}(\text{cond}) \land \text{is-SIZI}(\text{cond}) \land \text{is-SUBRG}(\text{cond}) \land \neg \text{is-Z~IV}(\text{cond}) \Rightarrow \text{error};\]
\[\text{call-condition-2}(\text{cond}, \text{cbif})\]
\[\text{null}\]
\[\text{is-ERROR}(\text{cond}) \Rightarrow \text{error}\]

Ref.: cond-sel 3-19(74)
      is-check 3-21(77)
      is-f-cond 3-21(78)
      is-eattn-cond 3-19(72)

(56) \[\text{call-condition-2}(\text{cond}, \text{cbif}) =\]
\[\text{is-sture-on-unit}((\text{cond-sel}(\text{cond}, \text{AT})) \land \text{s-cap}(\text{CS}) \Rightarrow \text{error};\]
\[\text{int-snap}((\text{cond-sel}(\text{cond}, \text{AT})) \land \text{s-cap}(\text{CS}), \text{cbif});\]
\[\text{int-snap}((\text{cond})\Rightarrow \text{error};\]
\[\text{syst-cond-exec}(\text{cond}, \text{cbif});\]
\[\text{int-snap}((\text{cond})\Rightarrow \text{error};\]

Ref.: cond-sel 3-19(74)
      syst-cond-exec 10-10(26)

(57) \[\text{int-snap}(\text{cond}) =\]
\[\text{is-0-snap}((\text{cond-sel}(\text{cond}, \text{AT})) \land \text{s-cap}(\text{CS}) \Rightarrow \text{null}\]
\[\text{null}\]
\[\text{null}\]

Ref.: cond-sel 3-19(74)

(58) \[\text{snap-action}(\text{cond}) =\]
Note: The snap action is implementation defined.

(59) \text{error-exec} =

Note: The action after normal return from an on-unit for an \text{ERROR} condition is implementation defined. In some cases the action is \text{error}.

10.4.2 \text{INTERPRETATION OF CONDITION ACTION}

A new block activation is established by the instruction \text{int-cond} and after updating the condition bif part, the statement of the on-unit is interpreted. The instruction \text{epilogue} terminates the block activation, making use of the epilogue information installed in \text{EL}.

(60) \text{int-cond}(ca,cbif) =

\text{is-corr-on-unit}(s-on-unit(ca)) \rightarrow

s-ba:*
\text{s-ei:} y_0 (<s-block-act:on>,<s-free-set:[]>,<s-task-set:[]>)
\text{s-cs:} \mu(CS;<s-bpp:s-bpp(ca)>)
\text{s-d:} stack(\#A,\#II,CS,\#D,CI,C)
\text{s-ci:} 0
\text{s-c:} \text{epilogue:}
\text{int-st(s-on-unit(ca)); update-chif(s-cond(ca),cbif)}

T \rightarrow \text{error}

Ref.: stack 6-2(2)
epilogue 6-11(31)
int-st 6-40(116)
update-chif 10-25(73)

(61) \text{is-corr-on-unit}(st) =

\text{is-\langle label-list(st) \rangle} \& \neg(\text{is-return-st(st) \& is-if-st(st) \& is-on-st(st) \& is-group(st)})

for:is-st(st)

10.4.3 \text{SIGNAL STATEMENT}

The execution of a signal statement for a condition causes the condition to be immediately raised.
abstract syntax and interpretation of PL/I

10.4.4 special condition activations

10.4.4.1 activation of check condition

The instruction call-check-cond is used to raise the check condition for a list of references; each of these references may in general be subscripted, but if the check condition is raised by a signal statement, the references are not subscripted. In both cases the order of the references is relevant.

The instruction call-check-cond for an existing and non-empty reference establishes an iterated condition call to an expanded reference list which contains references to all base elements of structures referenced in reflist.

\[\text{int-signal-st(t)} =\
(WI) (1 \leq i \leq \text{length(s-ref-list*s-cond(t)}) \Rightarrow\text{is-corr-check}(\text{elem(i,s-ref-list*s-cond(t)),AT}) \rightarrow\]

\[\text{iterate-call-cond(erl,SIGNAL);}\]
\[\text{erl:pass-expand-nil(s-ref-list*s-cond(t),AT)}\]
\[\text{is-attn-cond(s-cond(t)) & length(s-attn-list*s-cond(t)) = 1} \rightarrow\]

\[\text{call-cond-1(cond,cbif_{1});}\]
\[\text{cond:pass-eval-attn(head*s-attn-list*s-cond(t),2N,2V)}\]
\[\text{is-named-io-cond(s-cond(t))} \rightarrow\]

\[\text{call-cond-1(cond,\mu(chif_{1};<s-onfile:onf-1>))};\]
\[\text{onf-{1:pass-onf(cond,fp);}\]
\[\text{cond:eval-named-io-cond(s-cond(t))}\]

\[-\text{is-check-cond(s-cond(t)) & -is-attn-cond(s-cond(t))} \&\]
\[\text{(is-prcgr-named-cond(s-cond(t)) \& is-COND(attr_{0})} \rightarrow\]

\[\text{call-cond-1(s-cond(t),cbif_{1})}\]

\[T \rightarrow \text{error}\]

where:

\[\text{cbif_{1} = \mu(\text{<s-excode:code(s-cond(t),e)},<s-type:SIGNAL>)}\]
\[\text{attr}_{0} = (\text{s-n*s-cond(t)})\text{(st)}\]

for:is-signal-st(t)

Ref.:  
\[\text{is-corr-check 10-2(3)}\]
\[\text{eval-attn 10-7(16)}\]
\[\text{eval-named-io-cond 10-11(32)}\]
\[\text{code 10-27(80)}\]

\[\text{cnf} \text{(cond,fp)} =\]
\[\text{s-id(s-f(ccnd)(fp))}\]

for:is-fd(fp),is-f-cond(cond)

Ref.:  
\[\text{is-fd 3-24(86)}\]
\[\text{is-f-cond 3-21(78)}\]
(64) \( \text{call-check-cond}(r_l) = \)
\[
\text{is-<>}(r_l) \lor \text{is-S}(r_l) \rightarrow \text{null}
\]
\[
T \rightarrow\]
\[
\text{iterate-call-cond}(r_l, 0);\]
\[
\text{erl:pass-expand-rl}(r_l, AT)
\]
\[
\text{for:}(\text{is-ref-list} \lor \text{is-S})(r_l)
\]

(65) \( \text{iterate-call-cond}(r_l, \text{type}) = \)
\[
\text{is-<>}(r_l) \rightarrow \text{null}
\]
\[
T \rightarrow\]
\[
\text{iterate-call-cond}(\text{tail}(r_l), \text{type});\]
\[
\text{call-cond-1}(\text{head}(r_l), \text{cbif}_1)
\]
\[
\text{where:}\]
\[
\text{cbif}_1 = \mu_0(\langle s-code:s-code(\text{head}(r_l), t)\rangle, \langle s-type:s-type\rangle)
\]
\[
\text{for:}\text{is-ref-list}(r_l)
\]
\[
\text{Ref.: code 10-27(80)}
\]

(66) \( \text{expand-rl}(r_l, at) = \)
\[
\text{is-<>}(r_l) \rightarrow <>
\]
\[
T \rightarrow \text{expand-ref}(\text{head}(r_l), at) \cdot \text{expand-rl}(\text{tail}(r_l), at)
\]
\[
\text{for:}\text{is-ref-list}(r_l)
\]
10.4.4.2 Activation of conversion condition

If a conversion condition is not raised by a signal statement but through an actual conversion error, a specific action is activated which either allocates a dummy and passes this generation or passes the generation passed to this instruction to the onsource condition builtin function and in both cases passes the integer to the onchar condition builtin function. The value returned after interpreting the call may be modified through pseudovariables.
(70) \texttt{call-conv-cond(vg,i) =} 
\texttt{is-char-val-list(vg) \rightarrow} 
\texttt{pass-op-val(op); free(b); op:gen-op(gen);} 
\texttt{call-cond-1(CONV,cbif); cbif:pass-oneschif(gen,i,\xi);} 
\texttt{assign(gen,val-op(char-eda-var(ONS-LENGTH),vg));} 
\texttt{gen:allocate(b,aggr-scalar(char-eda-var(ONS-LENGTH)),DUMMY);} 
\texttt{s-id(vg) = mk-id(SUBSTR) \rightarrow} 
\texttt{pass-op-val(op); or:eval-onsource-substr(vg);} 
\texttt{call-cond-1(CONV,cbif); cbif:pass-oneschif(vg,i,\xi)} 
\texttt{for:is-char-val-list(vg) \lor is-gen(vg) \lor is-ps-gen(vg),is-intg-val(i)} 

\textbf{Ref.:} is-char-val 9-3(6) 
op-val 9-9(36) 
free 7-16(58) 
gen-op 8-22(59) 
assign 8-9(23) 
val-op 9-9(38) 
char-eda-var 9-12(60) 
allocate 7-11(25) 
aggr-scalar 8-23(64) 
p-name 3-10(20) 
eval-onsource-substr 12-62(149) 
is-gen 3-14(30) 
is-ps-gen 12-67(169) 
is-intg-val 9-3(5) 

(71) \texttt{cnscschif(g,i,\xi) =} 
\texttt{\nu_\mathcal{E}(<s-onsource:g>,<s-onchar:i>,<s-oncode:code(CONV,\xi)>)} 
\texttt{for:is-gen(g) \lor is-ps-gen(g),is-intg-val(i)} 

\textbf{Ref.:} code 10-27(80) 
is-gen 3-14(30) 
is-ps-gen 12-67(169) 
is-intg-val 9-3(5) 

\textbf{10.4.4.3 Activation of i/o conditions} 
Every i/o condition call is interpreted by the instruction \texttt{call-io-cond-1}, which possibly modifies the cbif argument used for interpreting condition builtin functions.

\textbf{24 10. ATTENTIONS AND CONDITIONS}
(72) \[ \text{call-if-cond-1(io-cond) =} \]
\[ \text{is-\&(io-cond) \& is-\&(\text{io})} \rightarrow \text{null} \]
\[ \text{is-f-ccmd(io-cond)} \rightarrow \]
\[ \text{call-cond-1(6(io-cond; s-chif), i(chif_{1}; <s-chif: id_{1}>; <s-oncode: code_{1}>>)} \]

where:
\[ \text{id}_{1} = \text{(is-nor-f(io-cond) \& s-id(s-f(io-cond)(\&))}, \]
\[ T \rightarrow \text{mk-id(SYSFININT)} \]
\[ \text{chif}_{1} = \text{s-chif(io-cond)} \]
\[ \text{code}_{1} = \text{(is-intg-val\& s-oncode(chif_{1}) \& s-oncode(chif_{1})}, \]
\[ T \rightarrow \text{code(io-cond,\&)} \]

for: \( \text{(is-f-cond \& is-\&(\text{io-cond})} \]

Ref.: \( \text{is-f-cond 3-21(78)} \]
\( \text{is-intg-val 9-3(5)} \]
\( \text{code 10-27(80)} \]

Note: \( \text{is-\&(\text{io})} \) is a criterion for a task epilogue in process.

10.5 CONDITION BUILTIN FUNCTION STATUS

Condition builtin functions change the value they return as a consequence to condition raising. The new value obtained remains unchanged in the dynamic descemence of the condition raising, i.e., in all blocks entered from an en-unit executed as consequence of the condition raising. If a condion is raised and only the system action is executed, no change of the condition-bif-values is required. The updating of condition-bif-values is defined by the instruction \text{update-chif} which is executed as part of the interpretation of a condition action. The value of a condition builtin function, where the corresponding component of the condition bif part is \&, is described in chapter 12 with the builtin function.

(73) \[ \text{update-chif(cond, chif) =} \]
\[ \text{is-ERROR(cond) \& is-\&s-cond(chif)} \rightarrow \text{update-chif-1(s-cond(chif), chif)} \]
\[ T \rightarrow \text{update-chif-1(cond, chif)} \]

Note: When the ERROR condition is raised as standard system action for a condition, the condition builtin functions return the same values as in an en-unit for the condition.

(74) \[ \text{update-chif-1(cond, chif) =} \]
\[ \text{s-cs:}\mu(\text{CS;} <s-chif: mk-chif(cond, chif, CS>)} \]
Abstract Syntax and Interpretation of PL/I

(75) \( \text{mk-cbif}(\text{ccnd}, \text{cbif}, \text{cs}) = \)
\[ \begin{align*}
& \text{is-f-ccnd}(\text{ccnd}) \rightarrow \text{mk-io-cbif}(\text{cond}, \text{cbif}, \text{cs}) \\
& \text{is-CONV}(\text{cond}) \rightarrow \text{mk-conv-cbif}(\text{cond}, \text{cbif}, \text{cs}) \\
& \text{is-eattn-ccnd}(\text{cond}) \rightarrow \text{mk-eattn-cbif}(\text{cond}, \text{cbif}, \text{cs}) \\
& T \rightarrow \text{mk-cbif-0}(\text{cond}, \text{cbif}, \text{cs})
\end{align*} \]

Ref.: \( \text{is-f-ccnd} \ 3-21(78) \)
\( \text{is-eattn-ccnd} \ 3-19(72) \)

(76) \( \text{mk-cbif-0}(\text{cond}, \text{cbif}, \text{cs}) = \)
\[ \begin{align*}
& \mu(\text{s-cbif}(\text{cs}); \text{<s-oncode:s-oncode(cbif)>}, \text{<s-onloc:s-entry>s-cbif(cbif)>}, \\
& \text{<s-abn-ret:s-abn-ret(cbif)>}, \text{<s-cond:0>})
\end{align*} \]

(77) \( \text{mk-io-cbif}(\text{cond}, \text{cbif}, \text{cs}) = \)
\[ \begin{align*}
& \text{is-NAME}(\text{cond}) \wedge \text{is-SIGNAL}(\text{type}) \rightarrow \mu(\text{cbif}; \text{<s-datafield:0>}) \\
& \text{is-NAME}(\text{cond}) \rightarrow \mu(\text{cbif}; \text{<s-datafield:s-datafield(cbif)>}) \\
& \text{is-EOV}(\text{cond}) \vee \text{is-EOV}(\text{cond}) \wedge \text{is-SIGNAL}(\text{type}) \rightarrow \mu(\text{cbif}; \text{<s-onident:0>}) \\
& \text{is-EOV}(\text{cond}) \wedge \text{is-EOV}(\text{cond}) \rightarrow \mu(\text{cbif}; \text{<s-onident:s-onident(cbif)>}) \\
& \text{is-PEND}(\text{cond}) \wedge \text{is-INDF}(\text{cond}) \wedge \text{is-SIGNAL}(\text{type}) \rightarrow \mu(\text{cbif}; \text{<s-onkey:0>}) \\
& \text{is-PEND}(\text{cond}) \wedge \text{is-INDF}(\text{cond}) \rightarrow \mu(\text{cbif}; \text{<s-onkey:s-onkey(cbif)>}) \\
& \text{is-ENDP}(\text{cond}) \vee \text{is-ENDP}(\text{cond}) \rightarrow \mu(\text{cbif}; \text{<s-onkey:s-onkey(cbif)>}, \text{<s-oncount:s-oncount(cbif)>}) \\
& T \rightarrow \mu(\text{cbif}; \text{<s-oncount:s-oncount(cbif)>})
\end{align*} \]

where:
\( \text{cbif} = \mu(\text{mk-cbif-0}(\text{cond}, \text{cbif}, \text{cs}); \text{<s-onfile:s-onfile(cbif)>}) \)
\( \text{cond} = \text{s-cond}(\text{cond}) \)
\( \text{type} = \text{s-type}(\text{cbif}) \)

for: \( \text{is-f-ccnd}(\text{cond}) \)

Ref.: \( \text{is-f-ccnd} \ 3-21(78) \)
ABSTRACT SYNTAX AND INTERPRETATION OF PL/I

(78) \( mk\text{-}conv\text{-}chif(\text{cond}, \text{chif}, \text{cs}) = \)

\[ \begin{align*}
&\text{is-SIGNAL\text{-}s\text{-}type(chif)} \rightarrow \mu(chif_0; \langle s\text{-}onsource:0, s\text{-}onchar:0 \rangle) \\
&\text{is-ON\text{-}s\text{-}onfile\text{-}def\text{-}s\text{-}chif(cs) \& is-ON\text{-}s\text{-}onkey(chif)} \rightarrow \\
&\mu(chif_0; \langle s\text{-}onfile:s\text{-}onfile\text{-}def\text{-}s\text{-}chif(cs), s\text{-}onkey:s\text{-}onkey(chif), s\text{-}onsource:s\text{-}onsource(chif), s\text{-}onchar:s\text{-}onchar(chif) \rangle) \\
&\text{is-ON\text{-}s\text{-}onfile\text{-}def\text{-}s\text{-}chif(cs)} \rightarrow \\
&\mu(chif_0; \langle s\text{-}onfile:s\text{-}onfile\text{-}def\text{-}s\text{-}chif(cs), s\text{-}onsource:s\text{-}onsource(chif), s\text{-}onchar:s\text{-}onchar(chif) \rangle) \\
&\text{T} \rightarrow \mu(chif_0; \langle s\text{-}onsource:s\text{-}onsource(chif), s\text{-}onchar:s\text{-}onchar(chif) \rangle)
\end{align*} \]

where:

\( chif_0 = mk\text{-}chif\text{-}0(\text{cond}, \text{chif}, \text{cs}) \)

for: is-CCNV(ccnd)

(79) \( mk\text{-}eattn\text{-}chif(ccnd, \text{chif}, \text{cs}) = \)

\[ \begin{align*}
&\text{is-SIGNAL\text{-}s\text{-}type(chif)} \rightarrow \mu(chif_0; \langle s\text{-}onattn:0 \rangle) \\
&T \rightarrow \mu(chif_0; \langle s\text{-}onattn:s\text{-}onattn(chif), s\text{-}ccnd:ccnd \rangle)
\end{align*} \]

where:

\( chif_0 = mk\text{-}chif\text{-}0(\text{cond}, \text{chif}, \text{cs}) \)

for: is-eattn\text{-}cond(ccnd)

Ref.: is-eattn\text{-}cond 3-79(72)

(80) \( \text{code}(ccnd, \xi) = \)

Note: Implementation defined function, whose values satisfy the axiom (81).

(81) \( (\text{cond}, \xi)(\text{is-intg\text{-}val\text{-}code}(\text{cond}, \xi) \& \text{code}(\text{cond}, \xi) \geq 0) \)

Ref.: is-intg\text{-}val 9-3(5)
This chapter defines all kind of data transmission to and from the external storage \( \mathbb{E} \), and to and from the message part \( \mathbb{M} \). Transmission involving \( \mathbb{E} \) is done by a file being related with a data set which consists of a single or more volumes, but only one volume at any time. Transmission to and from \( \mathbb{M} \) is either by display statements or by comments of standard system actions for on-conditions.

The chapter is divided into eight immediate sections, the last two of them being devoted to some special cases of stream transmission (11.7), and to the display transmission (11.8).

Section 11.1 defines data set mapping, i.e., the structuring of a data set as it appears if the data set is accessed by a particular file with a particular mapping parameter. This section also defines various functions and predicates which are fundamental for the following sections, in particular for 11.2 and 11.3. Most of the propositions over the domain and range of the defined functions is in the form of axioms.

Section 11.2 defines the five functions read, ignore, delete, rewrite, and write which are used for the basic access to data in \( \mathbb{E} \). Hence this section is strongly related with record transmission (11.5), and the sections on elementary stream I/O (11.6.3.3.1 and 11.6.4.3.1).

The first two sections are based upon the existence of a file without describing how it is built. Section 11.3 defines the initiation and termination of a file, including data set switching and data set label processing. Additional initiating actions occurring before I/O statements, such as options evaluation and attaching of I/O events, are defined in section 11.4. Moreover, section 11.4 contains all activities of record I/O excluding data transmission and buffer handling.

Section 11.5 defines record transmission. The sub-division of this section is made according to the different record I/O statements which corresponds more or less to the basic accessing functions (cf. 11.2).

Sections 11.6 and 11.7 define stream transmission over files and over strings. The expansion of data specifications (11.6.2) is connected with section 11.4.3 which describes the initiation of get and put statements. Sections 11.6.3 and 11.6.4 define the output and input transmission of single data fields. Stream input may require a syntactical scanning of the data field. Since this scanning is related with character string to arithmetic type conversion (cf. 9.5.1.3), all related definitions have been separated out into section 11.6.1.
11.1 DATA SET MAPPING

In this section the structure of inner data sets is defined, the complete set of file attributes and the mapping parameter are described, the main functions for the data set mapping, i.e. "decipher" and "cipher", are specified together with some other implementation defined functions, and the notion of "data set sharing" is discussed. However, some properties of the implementation defined functions and predicates are given by means of axioms.

Metavariables

is-ds(ds) a data set
is-ids(ids) an inner data set
is-el(el) an element of the intrinsic data of an inner data set
is-p(p) a file parameter
is-mp(mp) a mapping parameter
is-id(id) a title
is-intg-val(i) a key
is-intg-val(j) & j > 0
is-char-val-list(key) a complete or incomplete set of file attributes

11.1.1 STRUCTURE OF INNER DATA SETS

Given a data set (out of the external storage) and a mapping parameter (from a file which is linked with the data set), one may examine this pair whether the data set is decipherable with regard to the mapping parameter. If it is, then the data set could be deciphered into an inner data set according to the mapping parameter. Roughly speaking, the deciphering is a one-to-one mapping out of the class of data sets into the class of inner data sets.

In general, a data set is linked with more than one file at the same time ("data set sharing"). Hence, in general, a data set holds more information than one of its sharing files could require for any I/O-transmission (e.g. the positions and possibly the buffers of the other sharing partners, the number and kind of the sharing partners, descriptive information about data and about the "I/O-device", hidden data which do not "exist" for the file under consideration, but may exist for another sharing file etc.).

This superfluous information is held in an own component of the inner data set in an implementation defined representation. The reason for the presence of this information in the inner data set is the required one-to-one mapping between inner and outer data sets.
Note: The structure of inner data sets makes only those entities explicit which could be used (inspected or altered) via the file under consideration. These entities are: The labels, the intrinsic data, and two parts of the "state" of the I/O-device, namely the position and the mp-number. The position is END if the filemark had been passed. The mp-number offers the actual number of sharing files with the same mapping parameter. (On opening a file the corresponding mp-number is increased by 1, on closing it is decreased by 1. If the mp-number is zero, then no file with the mapping parameter under consideration is open, but at least one could be opened).

Note: On deleting a record by means of a delete statement, the record is replaced by the elementary object \texttt{DELETED}.
Note: This predicate is implementation defined.

\[(8) \text{is-csa}(sa) = \]
\[\sa \in \{\text{sa-1} \cup \{a\} \cup \text{sa-2} \cup \text{sa-1} \in \{\text{CST}, \text{BST}, \text{REC}, \text{TR}, \text{KEY}\} \} \land \]
\[a \in \{\text{INP}, \text{OUT}, \text{DEL} \} \land \text{sa-2} \in \{\{], [\text{PET}], [\text{EXC}]\} \land \]
\[(\text{UPD} \in \text{sa} \land \text{EXC} \in \text{sa} \land \text{REC} \in \text{sa}) \land (\text{PET} \in \text{sa} \land \text{CST} \in \text{sa}) \lor \]
\[\sa \in \{\text{REC} \cup \{a-1, a-2, a-3\} \cup \text{sa-1} \cup a-1 \in \{\text{SEQ}, \text{TRA}\} \land a-2 \in \{\text{INP}, \text{OUT}, \text{UPD}\} \land \]
\[a-3 \in \{\text{BUF}, \text{UNB}\} \land \text{sa-1} \in \{\{], [\text{BAC}], [\text{KEY}], [\text{BAC}, \text{KEY}]\} \} \land \]
\[(\text{BAC} \in \text{sa} \lor \text{INP} \in \text{sa} \lor \text{SEQ} \in \text{sa}) \lor (\text{TRA} \in \text{sa} \lor \text{UPD} \in \text{sa})] \]

Note: Each of the 15 file qualification attributes which satisfies the above predicate is one of the 33 complete sets of attributes.

These complete sets of attributes allow the separation of data transmission by a file into the following six classes:

a) Stream transmission:
   
   CST with INP or OUT or OUT,PET.
   
   BST with INP or OUT.

b) Record sequential buffered transmission:
   
   REC, SEQ, BUF with INP or INP,KEY or INP,BAC or INP,KEY,BAC or
   
   OUT or OUT,KEY or
   
   UPD or UPD,KEY.

c) Record sequential unbuffered transmission:
   
   REC, SEQ, UNB with (see b)).

d) Record transient buffered transmission:
   
   REC, TRA, BUF with INP or INP,KEY or
   
   OUT or OUT,KEY.

e) Record transient unbuffered transmission:
   
   REC, TRA, UNB with (see d)).

f) Record direct transmission:
   
   REC, TRA, KEY with INP or OUT or UPD or UPD,EXC.

\[(9) \text{is-mp}(obj) = \]
\[(3p)(p \in p(\text{mp}(p) = \text{obj})] \]

Note: This predicate describes the class of all mapping parameters. A mapping parameter contains the entire information necessary for data set mapping. It consists of three components: The title, the evaluated environment attribute, and a set of file attributes.

\[(10) p \in p(p) = \]
\[
m(p; \{s-csa:s-csa(p) - \{\text{BUF}, \text{UNB}, \text{EXC}\}\})] \]

Note: This function transforms the file parameter p into the corresponding mapping parameter by decreasing the complete set of file attributes by those attributes which do not influence the data set mapping.
(11) \( \text{is-prop-ids}(mp, ids) = \)
\[
(\text{is-intg-val}\cdot s\cdot \text{pos}(ids) \geq 0 \leq s\cdot \text{pos}(ids) \leq \text{length}\cdot s\cdot \text{data}(ids)) \land \\

(s\cdot mp\cdot no(ids) \geq 0 \land \\
\forall i \left( 1 \leq i \leq \text{length}\cdot s\cdot \text{data}(ids) \Rightarrow \\
(\text{is-prop-el}(s\cdot \text{csa}(mp), el_{i0}) \lor \text{REC} \in s\cdot \text{csa}(mp) \land \text{is-DELETED}(el_{i0}) \land \\
(\text{KEY} \in s\cdot \text{csa}(mp) = \text{is-prop-key}(mp, ids, s\cdot \text{key}(el_{i0}))))) \\
\]

where:
\( el_{i0} = \text{elem}(i, s\cdot \text{data}(ids)) \)

Ref.: \( \text{is-intg-val} \ 9-3(5) \)

Note: This predicate restricts the notion of inner data sets to the form in which it is further used.

(12) \( \text{is-prop-el}(sa, el) = \)
\[
\text{PRT} \in sa \rightarrow (\text{is-str-el} \land \neg \text{is-bit-val})(el) \\
\text{CST} \in sa \rightarrow (\text{is-char-val} \lor \text{is-LDEL})(el) \\
\text{BST} \in sa \rightarrow (\text{is-bit-val} \lor \text{is-LDEL})(el) \\
\text{KEY} \in sa \rightarrow \text{is-krec}(el) \\
\text{REC} \in sa \rightarrow \text{is-rec}(el) \\
\]

for: PRT \( \in sa \lor \text{CST} \in sa \lor \text{BST} \in sa \lor \text{KEY} \in sa \lor \text{REC} \in sa \)

Ref.: \( \text{is-bit-val} \ 9-4(11) \)
\( \text{is-char-val} \ 9-3(6) \)

Note: \( \neg \text{is-prop-el}(sa, \text{DELETED}) \) holds.

(13) \( \text{is-prop-key}(mp, ids, key) = \)

Note: This predicate is implementation defined. It is true if a key is proper with regard to the mapping parameter and (possibly) to the descriptive information within the data set, false otherwise.

11.1.2 FUNDAMENTALS OF MAPPING

(14) \( \text{is-decipherable}(mp, ds) = \)

Note: This predicate is implementation defined. It describes the domain of the function decipher.

(15) \( \text{decipher}(mp, ds) = \)

Note: This function is implementation defined. Its range is described by:
ABSTRACT SYNTAX AND INTERPRETATION OF PL/I  
30 April 1969

(16) \textit{is-decipherable}(\textit{mp}, \textit{ds}) \Rightarrow \textit{is-ids} \circ \textit{decipher}(\textit{mp}, \textit{ds}) \& \textit{is-props-ids}(\textit{mp}, \textit{decipher}(\textit{mp}, \textit{ds}))

(17) \textit{cipher}(\textit{mp}, \textit{ids}) = \textit{ds} \\

\textbf{Note:} This function is implementation defined. It maps certain proper inner data sets into data sets. Its domain and range is implicitly given by:

(19) \textit{is-decipherable}(\textit{mp}, \textit{ds}) \Rightarrow \textit{cipher}(\textit{mp}, \textit{decipher}(\textit{mp}, \textit{ds})) = \textit{ds} \\

\textbf{Note:} This axiom states that the functions \textit{decipher} and \textit{cipher} are inverse with respect to their second argument.

(19) \textit{is-deciphered}(\textit{mp}, \textit{ids}) = \\

(\exists \textit{ds}) \{ \textit{is-decipherable}(\textit{mp}, \textit{ds}) \& \textit{ids} = \textit{decipher}(\textit{mp}, \textit{ds}) \} \\

\textbf{Note:} This predicate describes the domain of the function \textit{cipher} (which corresponds with the range of the function \textit{decipher}). With (18) it is evident that if \textit{is-deciphered}(\textit{mp}, \textit{ids}) holds, there exists only one \textit{ds}, exactly that one which is yielded by \textit{cipher}(\textit{mp}, \textit{ids}).

11.1.3 DATA SET ACTIVITY

Given a data set \textit{ds} and a mapping parameter \textit{mp}, the following case distinctions can be made:

1) \textit{is-decipherable}(\textit{mp}, \textit{ds}) holds, indicates that either the data set is improper (e.g. the result of a grave transmission error) or the data set is proper but is incompatible with the mapping parameter.

2) \textit{ds} is decipherable with regard to \textit{mp} and the \textit{mp-no} component of the corresponding inner data set is zero, indicates that no file with the mapping parameter \textit{mp} is linked with \textit{ds}, but that at least one such file could be linked with \textit{ds}. On opening this file, the \textit{mp-no} component is set to one.

3) \textit{ds} is decipherable with regard to \textit{mp} and the \textit{mp-no} component of the corresponding inner data set is the integer value \( n > 0 \), indicates that \( n \) files with \textit{mp} are sharing \textit{ds}, independently of whether \textit{ds} is also shared by other files or not.

A data set is said to be active if there exist at least one mapping parameter for which the corresponding \textit{mp-no} is greater than zero, inactive otherwise. A data set is said to be active with respect to a certain \textit{mp} if the corresponding \textit{mp-no} is greater than zero, inactive with respect to \textit{mp} otherwise. Data set sharing occurs on a certain \textit{ds} if there exist at least two files such that \textit{ds} is active with respect to the mapping parameters of these files.

In the following some properties concerning data set activity are defined.

(20) \textit{is-decipherable-o}(\textit{mp}, \textit{ds}) = \\

\textit{is-decipherable}(\textit{mp}, \textit{ds}) \& \textit{is-deciphered}(\textit{mp}, \textit{decipher}(\textit{mp}, \textit{ids}),<\textit{mp-no}>,\textit{mp-no}(\textit{ids}) + 1>) \\

where: \textit{ids} = \textit{decipher}(\textit{mp}, \textit{ds})
Note: During interpretation this predicate is used only on creating. It examines the data set ds whether ds is able to communicate with a file with mapping parameter mp. (If it is so, then the mp-no of the corresponding inner data set is increased by 1 and this altered inner data set is mapped back into an outer data set, say ds', by the function cipher. ds and ds' are different data sets with different properties.)

(21) \(\text{is-deciphered}(mp, ids) \land s\mp\text{-no}(ids) = 0 = \text{is-deciphered}(mp, mp(ids;s\mp\text{-no});1))\)

Note: This axiom is equivalent to

\[\text{is-decipherable}(mp, ds) \land s\text{-mp-no}\text{-decipher}(mp, ds) = 0 = \text{is-decipherable-0}(mp, ds)\]

(22) \(\text{is-decipherable}(mp, ds) \land s\text{-mp-no}(ids_0) > 0 =
\begin{align*}
(\&ds-1) & (\text{is-decipherable}(mp, ds-1) \land \text{decipher}(mp, ds-1) = ids_1 \land \\
(\&mp-1) & (mp-1 \neq mp \land \text{is-decipherable}(mp-1, ds) \land \\
& \delta(\text{decipher}(mp-1, ds-1); s\text{-garbage}) = \delta(\text{decipher}(mp-1, ds); s\text{-garbage}) \land \\
(\&mp-1) & (\text{is-decipherable}(mp-1, ds) \land \text{is-decipherable}(mp-1, ds-1) = \\
& \text{s-mp-no-decipher}(mp-1, ds-1) = 0)\)
\end{align*}\)

where:

\[ids_0 = \text{decipher}(mp, ds)\]
\[ids_1 = \mu(ids_0; s\text{-mp-no}; s\text{-mp-no}(ids_0) - 1)\]

Note: On closing a file with the mapping parameter mp the corresponding mp-no component is decreased by 1. This action must not influence the properties between the remaining sharing files and the data set. The new data set (i.e. after closing) could be able to share files which the old data set (i.e. before closing) could not share.

(23) \(\text{is-decipherable}(mp, ds) \land \text{is-decipherable}(mp, ds-1) \land \\
\text{s-mp-no}(ids_0) = \text{s-mp-no}(ids_1) \land \text{s\text{-garbage}(ids_0)} = \text{s\text{-garbage}(ids_1)} =
\begin{align*}
(\&mp-1) & (\text{is-decipherable}(mp-1, ds) \land \\
& \text{is-decipherable}(mp-1, ds-1) \land \\
& \text{s-mp-no-decipher}(mp-1, ds-1) = \text{s-mp-no-decipher}(mp-1, ds))\)
\end{align*}\)

where:

\[ids_0 = \text{decipher}(mp, ds)\]
\[ids_1 = \text{decipher}(mp, ds-1)\]

Note: By changing the labels and/or the intrinsic data and/or the position via a certain file (ds, ids_0 = ids_1, ds-1), the mp-no components are not influenced. The corresponding components of sharing files with different mapping parameters may be altered as a consequence, but only in a way in which these files themselves would have been able to do it, i.e. the dataset remains decipherable for these files.
11.1.4 CORRESPONDENCE BETWEEN BACKWARDS AND FORWARDS ACCESS

(24) \[ (\forall mp) \text{is-decipherable}(mp, ds) = s-mp-no(\text{id}_0) = 0 \]
\[ (\forall mp) (\text{EAC} \in s-csa(mp) \& \text{is-decipherable}(mp, ds) \&
\text{s-trailer} (\text{id}_0) = s-trailer(\text{id}_1) \&
\text{s-data} (\text{id}_0) = \text{invert}\cdot\text{s-data} (\text{id}_1) \&
\text{s-pos} (\text{id}_0) = s-pos (\text{id}_1)) \]

where:
\[ \text{id}_0 = \text{decipher}(mp, ds) \]
\[ \text{id}_1 = \text{decipher}(mp_1, ds) \]
\[ mp_1 = \mu(\text{mp}; <s-csa:s-csa}(mp) - \{\text{EAC}\}) \]

Note: This axiom, together with (22), describes the relation between a "backwards file" and the corresponding "forwards file". Although this axiom constitutes a statement only about inactive data sets, together with (22) all interesting cases are covered. Roughly speaking, each data set which is active with respect to the mapping parameter of a certain backwards file (\(EAC \in s-csa(mp)\)) could be altered (if necessary) only by closing sharing files in such a way, that the new data set would be able to communicate with the corresponding forwards file. Furthermore, header and trailer labels are interchanged, the data list is inverted, and the position remains the same.

(25) \[ \text{invert}(l) = \]
\[ \text{LIST} \quad \text{elem}(\text{length}(l) - k + 1, l) \]
\[ \text{for} : \text{is-list}(l) \]

11.1.5 MAPPING PROPERTIES RELATED WITH THE FILE ATTRIBUTES INPUT, OUTPUT, AND UPDATE

This section states that input access is always a specialization of update access, and output access is a specialization of update access in the case of DIRECT. Finally, a more general property is given which correlates two sharing partners if their mapping parameters differ only in the file attributes.

(26) \[ \text{UPD} \in s-csa(mp) \& \text{INP} \in s-csa(mp-1) \&
\mu(\text{mp-1}; <s-csa:s-csa}(mp-1) - \{\text{INP}\}) = \mu(\text{mp};<s-csa:s-csa}(mp) - \{\text{UPD}\}) =
\text{(is-decipherable-}\text{O}(mp, ds) \Rightarrow \text{is-decipherable-}\text{O}(mp-1, ds)) \&
\text{(is-decipherable}(mp, ds) \& \text{is-decipherable}(mp-1, ds) =
\delta (\text{decipher}(mp, ds); s-pos, s-mp-no, s-garbage) =
\delta (\text{decipher}(mp-1, ds); s-pos, s-mp-no, s-garbage)) \]

Note: A data set which is suitable for a certain update file is also suitable for an input file whose mapping parameters differ only in the relevant file attribute. The labels and the data are the same.

(27) \[ \text{DIR, UPD} \in s-csa(mp) \& \text{OUT} \in s-csa(mp-1) \&
\mu(\text{mp-1}; <s-csa:s-csa}(mp-1) - \{\text{OUT}\}) = \mu(\text{mp};<s-csa:s-csa}(mp) - \{\text{UPD}\}) =
\text{(is-decipherable-}\text{O}(mp, ds) \Rightarrow \text{is-decipherable-}\text{O}(mp-1, ds)) \&
\text{(is-decipherable}(mp, ds) \& \text{is-decipherable}(mp-1, ds) =
\delta (\text{decipher}(mp, ds); s-pos, s-mp-no, s-garbage) =
\delta (\text{decipher}(mp-1, ds); s-pos, s-mp-no, s-garbage)) \]
Note: A data set which is suitable for a direct update file is also suitable for a direct output file whose mapping parameters differ only in the relevant file attribute. The labels and the data are the same too.

\[ \delta(mp;s-csa) = \delta(mp-1;s-csa) \land (\text{KEY} \in s-csa(mp) \equiv \text{KEY} \in s-csa(mp-1)) \land \]
\[ \text{BAC} \land s-csa(mp) \land \text{BAC} \land s-csa(mp-1) \land \text{CST} \in s-csa(mp) \equiv \text{CST} \in s-csa(mp-1) \land \]
\[ \text{FST} \land s-csa(mp) \land \text{FST} \land s-csa(mp-1) \land \text{BST} \in s-csa(mp) \equiv \text{BST} \in s-csa(mp-1) \land \]
\[ \text{is-decipherable}(mp,ds) \land \text{is-decipherable}(mp-1,ds) \Rightarrow \]
\[ \delta(\text{decipher}(mp,ds);s-\text{pos},s-\text{mp-no},s-\text{garbage}) = \]
\[ \delta(\text{decipher}(mp-1,ds);s-\text{pos},s-\text{mp-no},s-\text{garbage}) \]

11.1.6 Properties necessary for data transmission

This section describes those properties of the functions decipher and cipher, which are necessary for the various types of I/O-transmission, as specified by the basic accessing functions in 11.2. In addition, some implementation defined functions and properties of them are given, concerning the position after an unsuccessful search for a key, the size violation of records, and the indication whether a data set being written is physically full.

11.1.6.1 Positioning, reading, and deleting

(29) position(mp,ids,key) =

Note: This function is implementation defined. A read, delete, or rewrite operation for which the specified key is not found, or a write operation for which the key already exists causes a raising of the KEY-condition and a change of the position of the inner data set under consideration. By this function the new position is yielded. The domain and range of this function is described by:

\[ \text{KEY} \in s-csa(mp) \land \text{is-deciphered}(mp,ids) \land \text{is-END}(s-\text{pos}(ids)) \land \]
\[ \text{OUT} \land s-csa(mp) \land \text{is-prop-key}(mp,ids,key) \land \]
\[ \neg (\exists j)(s-key\text{\_elem}(j, s-data(ids)) = key) \lor \]
\[ \text{OUT} \land s-csa(mp) \land \text{DIR,UPD} \land s-csa(mp) \land \]
\[ \neg (\exists j)(s-key\text{\_elem}(j, s-data(ids)) = key)) \Rightarrow \text{is-intg-val\_position}(mp,ids,key) \land \]
\[ 0 \leq \text{position}(mp,ids,key) \leq \text{length}\_s-data(ids) \]

Ref.: is-intg-val 9-3(5)

(31) is-deciphered(mp,ids) \land 0 \leq i \leq \text{length}\_s-data(ids) \Rightarrow
\[ \text{is-deciphered}(mp,\mu(ids;<s-pos:id>)) \]

Note: By this axiom the proper updating of the position is secured. A proper read operation, for instance, causes only this kind of data set transition.

(32) is-deciphered(mp,ids) = is-deciphered(mp,\mu(ids;<s-pos:END>))

Note: On output the transition \text{INTG-VAL} - \text{END} will occur when the data set being written is physically full (cf. (39)), on input when there is no further data on the data set.

11. INPUT AND OUTPUT 9
11.1.6.2 Rewriting

(34) \textit{is-size-violation}(\textit{mp}, \textit{ds}, \textit{el}) =

\textbf{Note:} This predicate is implementation defined. It is used on write and rewrite operations of records. It tests whether the record has a proper size with regard to the mapping parameter and the data set.

(35) \textit{size-el}(\textit{mp}, \textit{ds}, \textit{el}) =

\textbf{Note:} This function is implementation defined. It maps a record which is not proper in its size onto a record with a proper size.

(36) \text{ (DIR,UPD) = s-csa(mp) \& is-decipherable(mp,ds) \& is-prop-el(s-csa(mp),el) \& is-unique-key(s-data(ids),s-key(el)) =

(is-size-violation(mp,ds,el) \& is-prop-el(s-csa(mp),el_1) \& s-key(el_1) = s-key(el) \& is-size-violation(mp,ds,el_1)) \&

(is-size-violation(mp,ds,el) \&

(\exists j) \{(s-key\cdot elem(j, s-data(ids)) = s-key(el) \& is-deciphered(mp,\mu(ids;\langle\text{elem}(j)\rangle\cdot s-data; el))\} \&

(\exists j) \{(elem(j, s-data(ids)) = el) \Rightarrow \text{-is-size-violation}(mp,ds,el)\})

\textbf{where:}

ids = decipher(mp,ds)

el_1 = size-el(mp,ds,el)

\textbf{Note:} This axiom describes on the one hand properties of the functions cipher and decipher concerning the direct rewrite operation, on the other hand it describes together with (38, 40) domain and range of (34) and (35).

(37) \textit{is-unique-key}(\textit{data}, \textit{key}) =

(\exists j)

\{(s-key\cdot elem(j, data) = key \& (j = j-1 \lor s-key\cdot elem(j-1, data) \neq key)\}

\textbf{for:is-el-list}(\textit{data})
(38) \(\text{SEQ, UPD} \in s-csa(mp) \land \text{is-decipherable}(mp, ds) \land \text{is-prop-el}(s-csa(mp), el) \land \text{is-TMP-r-pos}(id_{ds}) \land \text{r-pos}(id_{ds}) \neq 0 \land \text{is-DELTID}(el_{0}) \land s-key(el_{0}) = s-key(el) \Rightarrow \\
\text{is-size-violation}(mp, ds, el) \Rightarrow \text{is-prop-el}(s-csa(mp), el_{1}) \land s-key(el_{1}) = s-key(el) \land \text{is-size-violation}(mp, ds, el_{1}) \land \\
\text{is-deciphered}(mp, \{\text{id}_{ds}, \text{<elem}(s-pos(id_{ds})), \text{<data:el>}\}) \land \text{is-size-violation}(mp, ds, el_{0}) \land \\
\text{is-end}(mp, ds, el) = \\
\text{Note: This axiom describes properties of the function cipher and decipher concerning the sequential rewrite operation.}

11.1.6.3 Writing

(39) \text{is-end}(mp, ds, el) = \\
\text{Note: This predicate is implementation defined. It indicates on a write operation that the data set being written is physically so full that a transmission of the actual information el is impossible.}

(40) \text{REC} \in s-csa(mp) \land \text{is-decipherable}(mp, ds) \land \text{is-prop-el}(s-csa(mp), el) \land \\
\text{OUT} \in s-csa(mp) \land \text{KEY} \notin s-csa(mp) \land \\
\{(\text{OUT}, \text{KEY}) \in s-csa(mp) \lor \text{[DIR, URI]} \in s-csa(mp) \land \\
\text{is-prop-key}(mp, id_{ds}, s-key(el)) \land \text{is-size-violation}(mp, ds, el) \Rightarrow \\
\text{is-prop-el}(s-csa(mp), el_{1}) \land s-key(el_{1}) = s-key(el) \land \text{is-size-violation}(mp, ds, el_{1}) \land \\
\text{is-deciphered}(mp, \{\text{id}_{ds}, \text{<data:el>}\}) \land \text{is-end}(mp, ds, el_{0}) \land \text{is-size-violation}(mp, ds, el) \Rightarrow \\
\text{is-deciphered}(mp, \{\text{id}_{ds}, \text{<data:el>}\}) \land \\
\text{Note: This axiom describes on the one hand properties of the functions cipher and decipher concerning the record write operation, on the other hand it describes together with (38) domain and range of (34) and (35), and together with (44) the domain of (39).}

(41) \text{OUT} \in s-csa(mp) \land \text{REC} \in s-csa(mp) \land \text{is-decipherable}(mp, ds) \land \\
\text{is-prop-el}(s-csa(mp), el) \land \text{is-end}(mp, ds, el) \Rightarrow \\
\text{is-deciphered}(mp, \{\text{id}_{ds}, \text{<data:el>}\}) \land \\
\text{Note: This axiom describes properties of the functions cipher and decipher concerning the stream write operation.}
11.2 BASIC ACCESS TO DATA

This section defines the five basic accessing functions:

- read,
- ignore,
- delete,
- rewrite,
- write

which are used for the various types of I/O-transmission in the following sections. The theoretical basis for these functions is given in section 11.1.

Common to all of these functions is, that they map a data set together with some additional information onto a data set and some information about the success of this data set transition.

The additional information consists of the mapping parameter in any case, an optional key in the case of keyed read- and keyed delete-access, a piece of data to be written or rewritten, or an integer in the case of an ignore operation.

The functions yield as value an object with the structure

\[(s-ds:is-ds),
\langle s-inf:is-KEY v is-REC v is-END v is-0 v is-el\rangle\]

where the ds-component is the result of the data set transition, and the inf-component either indicates which type of I/O-condition is to be raised, or that the volume has to be switched, or offers a piece of data if reading was successful.

All functions become undefined if

i) the data set is not decipherable with regard to the mapping parameter,

ii) the data set is inactive with respect to the mapping parameter,

iii) the position of the corresponding inner data set is the elementary object END.

Emetavariabes:

- is-mp(mp) a mapping parameter
- is-ds(ds) a data set
- (is-char-val-list v is-d) (ad) an optional key
- is-ingt-val(i) & i > 0

Abbreviations:

- ids_0 = decipher(mp, ds)
- el_i_0 = elem(i, s-data(ids_0))
(42) \[ \text{read}(mp,ds,ad) = \]
\[ \text{is-decipherable-1}(mp,ds) \rightarrow \text{is-EN}(s-pos(id_{0}) \rightarrow \text{error} \]
\[ \text{is-char-val-list}(ad) \& \text{-is-prop-key}(mp,ids_{0},ad) \rightarrow \mu_{0}\langle s-ds:ds\rangle,\langle s-inf:KEY\rangle) \]
\[ \text{is-char-val-list}(ad) \& (\forall i)(s-key(id_{0} \neq ad) \rightarrow \mu_{0}\langle s-ds:ds\rangle,\langle s-inf:KEY\rangle) \]
\[ i_{0} > \text{lengths-data}(id_{0}) \rightarrow \mu_{0}\langle s-ds:ds\rangle,\langle s-inf:END\rangle) \]
\[ T \rightarrow \mu_{0}\langle s-ds:ds\rangle,\langle s-inf:elem(i_{0},s-data(id_{0}))\rangle) \]

where:
\[ i_{0} = (\text{is-char-val-list}(ad) \rightarrow (\forall i)(s-key(id_{0} \neq ad) \rightarrow (\forall i)(s-pos(id_{0}) > i_{0}) \& \text{-is-DELETED}(id_{0}) \& (\forall i)(s-pos(id_{0}) < i_{0} \rightarrow \text{-is-DELETED}(s-data(id_{0})))) \)

for:OUT \& s-csa(mp) \& (DIR \& s-csa(mp) \& is-char-val-list(ad)) \& (\text{KEY} \& s-csa(mp) \& is-\text{V}(ad))

Ref.: is-char-val 9-3(6)
is-prop-key 11-5(13)
cipher 11-6(17)
position 11-9(29)

Note: On sequential (or possibly transient) access (is-\text{V}(ad)) the next record is to be taken which is not DELETED.

(43) \[ \text{is-decipherable-1}(mp,ds) = \]
\[ \text{is-decipherable}(mp,ds) \& \text{m-mp-wo}(id_{0}) > 0 \]

Ref.: is-decipherable 11-5(14)

Note: This predicate examines the data set whether it is active with respect to the mapping parameter mp.
(44) ignore(mp,ds,n) =
    is-decipherable-1(mp,ds) & is-END•s-pos(ids_0) \rightarrow error
    i_0 > length•s-data(ids_0) \rightarrow
    \mu_0(<s-ds:cipher(mp,\mu(ids_0;<s-pos:END>)),<s-inf:END>)
    T \rightarrow \mu_0(<s-ds:cipher(mp,\mu(ids_0;<s-pos:i_0>))))

where:
    i_0 = \{li (i > s-pos(ids_0) \& \& \text{is-DELETED}(el-i_0) \&
    n = 
    \text{card}(|{i-1 \leq i \leq n \&
    \neg \text{is-DELETED}\text{elem}(i-1,s-data(ids_0)))|)

for:OUT \& s-csa(mp) \& (SEQ \& s-csa(mp) \& TRA \& s-csa(mp)) \& is-intg-val(m) \& n > 0

Ref.: cipher 11-6(17)
      is-intg-val 9-3 (5)

Note: The new position points to the predecessor of the (n+1)th not DELETED
      record, relative to the old position.

(45) delete(mp,ds,ad) =
    is-decipherable-1(mp,ds) \& is-END•s-pos(ids_0) \& \text{in}(ad) \& s-pos(ids_0) = 0 \rightarrow
    is-char-val-list(ad) \& \text{is-prop-key}(mp,ids_0,ad) \rightarrow \mu_0(<s-ds:ds>,<s-inf:KEY>)
    is-char-val-list(ad) \& (\forall i)(s-key(el-i_0) \neq ad) \rightarrow
    \nu_0(<s-ds:cipher(mp,\mu(ids_0;<s-pos:position(mp,ids_0,ad)>))),<s-inf:KEY>)
    T \rightarrow \nu_0(<s-ds:cipher(mp,\mu(ids_0;<(el-i_0)*s-data:DELETE>)),<s-pos:i_0>)))

where:
    i_0 = (is-\text{in}(ad) \rightarrow s-pos(ids_0)),
    T \rightarrow (\forall i)(s-key(el-i_0) = ad))

for:UPD \& s-csa(mp) \& \text{(DIR \& s-csa(mp) \& is-char-val-list(ad)) \&
    \text{(KEY \& s-csa(mp) \& is-\text{in}(ad))}

Ref.: is-char-val 9-3(6)
      is-prop-key 11-5(13)
      cipher 11-6(17)
      position 11-9(29)

Note: A delete operation without a key is also erroneous, if the corresponding
      inner data set has the initial position 0.
(46) \( \text{rewrite}(mp, ds, el) = \)

\[
\begin{align*}
\text{is-decipherable-1}(mp, ds) \land \text{is-ENS-pos}(ids_0) \lor \text{is-Qs-key}(el) & \land \ (i_0 = 0 \lor \text{is-DELETED}\text{-elem}(i_0, s\text{-data}(ids_0))) \rightarrow \\
\text{error} \\
\text{is-char-val-list}\text{-s-key}(el) \land \neg \text{is-prop-key}(mp, ids_0, s\text{-key}(el)) & \rightarrow \\
\mu_0(<s\text{-ds:ds}, <s\text{-inf:KEY}>) \\
\text{is-char-val-list}\text{-s-key}(el) \land (\forall i (s\text{-key}(el-i_0) \neq s\text{-key}(el))) & \rightarrow \\
\nu_0(<s\text{-ds:cipher}(mp, p(ids_0; <s\text{-pos:position}(mp, ids_0, s\text{-key}(el))>))>, <s\text{-inf:REC}>) \\
\text{is-char-val-list}\text{-s-key}(el) \land \neg \text{is-unique-key}(s\text{-data}(ids_0), s\text{-key}(el)) & \rightarrow \text{error} \\
\text{is-size-violation}(mp, ds, el_4) & \rightarrow \\
\nu_0(<s\text{-ds:cipher}(mp, p(ids_0; <(elem(i_0))\text{-s-data:el_4}(mp, ds, el_4)>, <s\text{-pos:i_0}>)>), <s\text{-inf:REC}>) \\
T & \rightarrow \nu_0(<s\text{-ds:cipher}(mp, p(ids_0; <(elem(i_0))\text{-s-data:el_4}, <s\text{-pos:i_0}>)>)) \\
\end{align*}
\]

where:
\[
\begin{align*}
i_0 & = (\text{is-Qs-key}(el) \rightarrow \text{s-pos}(ids_0)), \\
T & \rightarrow (\forall i (s\text{-key}(el-i_0) = s\text{-key}(el))) \\
e_1 & = (\text{is-Qs-key}(el) \rightarrow \\
\nu_0(<s\text{-key:s-key}\text{-elem}(s\text{-pos}(ids_0), s\text{-data}(ids_0)>, <s\text{-wr:el}>)>, \\
T \rightarrow e_1) \\
\end{align*}
\]

for: UPD & is-csa(mp) & (is-Qs-key(el) = SEQ & is-csa(mp)) & (is-rec & is-rec)(el)

Ref.: is-char-val 9-3(6) 
is-prop-key 11-5(13) 
cipher 11-6(17) 
position 11-9(29) 
is-unique-key 11-10(37) 
is-size-violation 11-10(34) 
size-el 11-10(35) 
is-rec 11-3(5) 
is-rec 11-3(6)

Note: A sequential rewrite operation, i.e., is-Qs-key(el), is also erroneous, if the corresponding inner data set has the initial position 0, or if the record to be replaced is DELETED.
(47) \text{write}(mp, ds, el) =
\begin{align*}
\text{is-decipherable-1}(mp, ds) & \Rightarrow \text{is-END} \& \text{s-pos}(ids_0) \rightarrow \text{error} \\
\text{is-char-val-list-s-key}(el) & \& \text{-is-prop-key}(mp, ids_0, s-key(el)) \Rightarrow \\
\mu_0(<s-ds:ds>, <s-inf:KEY>) \\
\text{is-char-val-list-s-key}(el) & \& (\exists i) \text{ s-key}(el-i_0) = s-key(el) \Rightarrow \\
\mu_0(<s-ds:cipher(mp, ids_0; <s-pos:position(mp, ids_0, s-key(el))>), <s-inf:KEY>) \\
\text{is-end}(mp, ds, el) & \Rightarrow \mu_0(<s-ds:cipher(mp, ids_0; <s-pos:END>)>, <s-inf:END>) \\
\text{REC} \& \text{s-csa}(mp) & \& \text{is-size-violation}(mp, ds, el) \Rightarrow \\
\mu_0(<s-ds:cipher(mp, ids_0; <s-data:s-data(ids_0) <size-el(mp, ds, el)>), <s-pos:pos_1>), <s-inf:REC>) \\
\top & \Rightarrow \mu_0(<s-ds:cipher(mp, ids_0; <s-data:s-data(ids_0) <el>), <s-pos:pos_2>)))
\end{align*}

where:
\begin{align*}
pos_1 & = 1 + \text{length}s-data(ids_0)
\end{align*}

for:(\text{OUT} \& \text{s-csa}(mp) \& \text{DIR} \& \text{s-csa}(mp) \& \text{UPD} \& \text{s-csa}(mp)) \& \text{is-prop-el}(s-csa(mp), el)

\text{Ref.}: \text{is-char-val} 9-3(6) \\
\text{is-prop-key} 11-5(13) \\
\text{cipher} 11-6(17) \\
\text{position} 11-9(29) \\
\text{is-end} 11-11(39) \\
\text{is-size-violation} 11-10(34) \\
\text{size-el} 11-10(35) \\
\text{is-prop-el} 11-5(12)
11.3 INITIATION AND TERMINATION OF A FILE

A file is initiated by opening. Depending on how opening is achieved one may distinguish between explicit opening (by an open statement) and implicit opening (by an I/O statement other than open or close naming a file or by other statements if they refer to a standard system print file). Depending on what is achieved by opening one may discern between proper opening and improper opening. Proper opening makes a new entry in the file union directory FU and enters a link in the file directory FD. The entry in FU contains a link to a data set in ES, and the entry in FD is accessible to a file value. Improper opening is the opening of an already existing file or an erroneous case of opening which may be terminated by an undefined file condition call. Proper closing deletes the entries made at opening from FU and FD.

Since the evaluation of statement options proceeds similarly in an open and close statement, the description of both statements has been taken together (section 11.3.1). Implicit opening, the deduction of file attributes, and the checking of the type of the file against the statement causing implicit opening is described in 11.3.2. The next sections define the file initiation and termination actions which are common to explicit and implicit opening and closing.

Data set label processing and the related area of multiple volume handling and data set switching is covered mostly in section 11.3.5.

**Metavariables**

- \( n \): is-opt-\( n \)  
  - File value, to be applied to FU
- \( f \): is-s-st-prt \( v \) is-n  
  - File name, to be applied to FD
- \( u \): is-n  
  - File union name, to be applied to FU
- \( csa \): is-file-attr-set  
  - Complete or incomplete (including contradictory) set of file attributes
- \( ea \): is-ea  
  - Evaluated environment attribute

**Abbreviations**

- \( fu-el_0 = (is-fu-el<fu>) \rightarrow u(FU) \), \( T \rightarrow \text{error} \)  
  - This is the entry in FU belonging to a file characterized by the file union name \( u \). Application of \( u \) to \( FU \) may be erroneous if \( u \) characterizes a file which does not more exist.
- \( P_0 = s-f(fu-el_0) \)  
  - This is the mapping parameter of the file.
- \( chs_0 = s-csa(P_0) \)  
  - The complete set of attributes, the evaluated environment attribute, and the data set title of the file.
- \( f_0 = s-f(fu-el_0) \)  
  - The file name of the file. It is a copy of the file name from the FD entry of the file. I/O
on-condition raising is based upon the file name.

ds-sel_0 = (ds-sel(title_ea, ea_0, s-ds-sh(ES))) • s-ds-dir

ds_0 = s-ds•ds-sel_0(ES)

tmt_0 = s-tmt•ds-sel_0(ES)
The data set accessed by the file.

dvolno_0 = decipher(mp_0, ds_0)
The inner data set, i.e., the data set accessed by the file as seen by the file.

dvolno_0 = s-volno(fu-1_el_0)
The current volume number which is incremented when the data set is switched.

dst_0 = s-st(fu-1_el_0)
The status of the file.

11.3.1 OPEN AND CLOSE STATEMENT

The operations (i.e., opening including UNDF on-condition raising, closing, data set label processing including BOV, BOV, and TMT on-condition raising) specified by the options group of a statement are performed left to right. The evaluation of the options occurs in an implementation defined order. Option evaluation is never paralleled with evaluation of other options or performance of operations.

Metavariabes

i, j is-intg-val
pt is-optional-open-close-list v is-0

selector to an option or options group in a pt

a is-file-attr

file attribute

sa is-file-attr-set

set of file attributes

sca sa is-csa-set

set of complete sets of file attributes
(48) \texttt{int-open-close-st}(t) = \\
\texttt{length}(t) \\
\texttt{Et} \ (\texttt{is-s*e-volume} \texttt{elem}(i,t) \Rightarrow \\
\texttt{is-2}(\texttt{elem}(i,t); \texttt{s-file}, \texttt{s-open-attr}, \texttt{s-volume}) \& \\
\{\texttt{is-2} \lor \texttt{is-0}\} (\texttt{s-open-attr} \texttt{elem}(i,t)) \\ \\
\texttt{int-open-close-1}(t) \\
T \rightarrow \texttt{error} \\
where: \\
\texttt{t} = \texttt{s-list}(t) \\
for: (\texttt{is-open-st} \lor \texttt{is-close-st})(t) \\

Note: The volume option can only occur together with the file option.

(49) \texttt{int-open-close-1}(pt) = \\
(3o) (\texttt{is-free-sel-1}(pt, o)) \rightarrow \\
\texttt{int-open-close-2}(pt, \texttt{arb}\{o \mid \texttt{is-free-sel-1}(pt, o)\}, pt) \\
T \rightarrow \texttt{null} \\

(50) \texttt{is-free-sel-1}(pt, o) = \\
(\texttt{is-expr} \lor \texttt{is-env-attr}) \{o(pt) \& \\
(3i, s-1) (s-1 \in [\texttt{s-file, s-tile}, \texttt{s-isz}, \texttt{s-psz}, \texttt{s-env-attr}] \& o = o-1(\texttt{elem}(i)) \lor \\
\texttt{is-e-open-close}^{\ast}(pt) \& (\forall j) (1 \leq j \leq i_1 \Rightarrow (\texttt{is-2} \lor \texttt{is-f-cond}) (\texttt{elem}(j, pt)) \lor \\
\texttt{is-f-cond}^{\ast}(pt) \& (\forall j) (1 \leq j \leq i_1 \Rightarrow \texttt{is-0} \texttt{elem}(j, pt))) \\
where: \\
i_1 = (4i) (o = \texttt{elem}(i)) - 1 \\

Ref.: is-f-cond 3-21(78) \\

Note: The predicate is satisfied for selectors which yield components of the 
partially evaluated text pt which may be interpreted in the next step. 
First of all, the options (of type \texttt{is-expr} or \texttt{is-env-attr}) are candidates 
for evaluation. Whenever all options in an options group are evaluated, 
the evaluated options group satisfies the predicate \texttt{is-e-open-close}. Such 
an evaluated options group may become candidate for interpretation if it is 
the leftmost evaluated group which has not yet been interpreted. As a 
result of the interpretation of an options group, the options group is 
deleted from pt or an indication for \texttt{TMT} or \texttt{UNDF on-} 
condition raising is returned. Any on-condition indication will become candidate for 
interpretation after all operations have been performed, and if the 
particular indication is the leftmost in pt.

(51) \texttt{arb(set, type)} = \\

Note: This is an implementation defined ordering function which yields on 
arbitrary member from the argument set. The selection depends on the 
second argument type which is some object. The function is characterized 
by the following axiom.
11. INPUT AND OUTPUT

20
(56) \[ \text{int-open-close}(t,a) = \]
\[ \begin{align*}
  \& = \text{a-file} \rightarrow \text{eval-ref}(t) \\
  \& = \text{a-title} \rightarrow \\
  & \text{pass-op-val}(\text{op} \rightarrow 1, \text{convert-1}(\text{aggr-scalar}(\text{CHAR-EDA}), \text{op}); \text{eval-expr}(t, 0)) \\
  \& = \text{a-lsz} \& \text{a-ps} \rightarrow \text{eval-intg-expr}(t) \\
  \& = \text{a-env-attr} \rightarrow \text{eval-env-attr}(t) \\
  \& = \text{is-f-cond}(t) \rightarrow \text{call-io-cond-1}(t) \\
  \& = \text{is-e-open-close}(t) \& \text{is-ges-open-attr}(t) \& \text{is-file-s-da-s-evae-s-file}(t) \rightarrow \\
  & \text{op-open-1}(n_1, \text{assoc-ea}(\text{is-open-attr}(t) \& \text{s-attr}(f_1(\text{FD})), \text{title}_1, \text{assoc-ea}(\text{s-env-attr}(t), \text{s-va}\text{f}(f_1(\text{FD})), \text{s-volume}(t)) \\
  \& = \text{is-e-open-close}(t) \& \text{is-FILL-s-da-eve-s-file}(t) \rightarrow \\
  & \text{op-close-1}(t, \text{assoc-ea}(\text{s-env-attr}(t), \text{s-va}\text{f}(f_1(\text{FD})), \text{s-volume}(t)) \\
  \& = \text{error} \\
\end{align*} \]

where:
\[ n_1 = \text{op-val-s-file}(t) \]
\[ f_1 = f_1(\text{ FD}) \]
\[ \text{title}_1 = \text{is-ges-title}(t) \rightarrow \text{mk-id-s-title}(t) \]

for:
\[ (\text{is-expr} \& \text{is-env-attr} \& \text{is-e-open-close} \& \text{is-f-cond})(t), a \rightarrow \text{ref-s file, title, s-lsz, s-ps, s-env-attr, I} \]

Ref.:
\[ \text{eval-ref 8-20(54)} \]
\[ \text{convert-1 9-29(119)} \]
\[ \text{aggr-scalar 8-23(64)} \]
\[ \text{eval-expr 8-10(24)} \]
\[ \text{eval-intg-expr 8-22(60)} \]
\[ \text{is-f-cond 3-21(78)} \]
\[ \text{call-io-cond-1 10-24(72)} \]
\[ \text{op-val 9-9(33)} \]

Note: The component of the text which is characterized by the selector \( a \) is interpreted, and the result of the interpretation is returned. Statement options are evaluated and, if necessary, converted. Information on on-condition raising in connection with previous data set label processing will lead to an on-condition call. Opening or closing is performed.

(57) \[ \text{eval-env-attr(env-attr)} = \]

Note: This implementation defined instruction evaluates the environment attribute env-attr and returns the evaluated attribute ea which satisfies the predicate is-ea(ea). There is no agreement on a particular treatment of special environment attributes.
ABSTRACT SYNTAX AND INTERPRETATION OF FI/T

30 April 1969

(58) \text{is-ea}(ea) =

Note: Every ea which satisfies this implementation defined predicate is an
evaluated environment attribute. Evaluated environment attributes are
returned by the instruction eval-env-attr, or by the function assoc-ea, or
are the constant objects DEF-EA-1 or DEF-EA-2.

(59) \text{assoc-ea}(ea-1,ea-2) =

Note: This implementation defined function maps the environment attribute of an
open or close statement and the evaluated declared environment attribute
into a single ea.

(60) \text{is-ea} \land \text{assoc-ea}(ea-1,ea-2) \land \text{is-ea}(DEF-EA-1) \land \text{is-ea}(DEF-EA-2)

Note: DEF-EA-1 is used as an indication for closing at task termination.
DEF-EA-2 is the default environment attribute of a standard system print
file.

(61) \text{assoc-csa}(sa) =

\{(\text{csa-1}) (sa-1 \land 
\text{csa-set-2}(\text{csa-set-2}(\text{csa-set-2}(\text{csa-set-1}(sa),\text{INP}),\text{CST}),\text{SEQ}),\text{BUF}) \lor
sa-1 = sa)\}

Note: The function yields either the complete set of attributes associated with
sa after applying defaults, or sa itself in the case when sa is
inconsistent. The function is never undefined over its domain.

(62) \text{csa-set-1}(sa) =

\{\text{csa} | \text{is-csa}(\text{csa}) \land sa \subseteq \text{csa} \land (\text{INR} \in \text{csa} \land \text{EXC} \in sa = \text{EXC} \in \text{csa},
T \rightarrow sa \land \text{add-a}_{1} = \text{csa} \land \text{add-a}_{1})\}

where:
\text{add-a}_{1} = \{\text{PRT, BAC, KEY}\}

Ref.: is-csa 11-4(8)

Note: For any set of attributes the set of complete sets of attributes is derived
without using defaults for alternative attributes.

(63) \text{csa-set-2}(\text{scsa},a) =

\{(\text{csa}) (a \in \text{csa} \land \text{csa} \subseteq \text{scsa}) \rightarrow \{\text{csa} | a \in \text{csa} \land \text{csa} \subseteq \text{scsa}\}
T \rightarrow \text{scsa}

11.3.2 IMPLICIT OPENING

This section defines opening as it occurs before the proper interpretation of
I/O statements, and the opening of standard system print files.
11.3.2.1 Opening with a file option

The interpretation of the options of I/O statements (cf. chapter 11.4.1) comprises handling of the file option by the instruction \texttt{int-impl-open}(t). There are essentially three steps: Evaluation, testing and insertion of the current file name (or the identifier, respectively), opening, and testing of the compatibility between statement text \texttt{t} and the file.

\textbf{Metavariabes}

\begin{itemize}
  \item \texttt{t} \quad \text{is-rec-io-st} \quad \text{is-file-get-st} \quad \text{is-file-put-st} \quad \text{is-f} \quad \text{is-rec-get} \quad \text{is-rec-put}
  \item \text{cond} \quad \text{is-f-cond} \quad \text{is-n}
  \item \text{fd} \quad \text{is-fd}
\end{itemize}

\textit{the text of an I/O statement causing implicit opening. In case of standard system print file \texttt{G} is used.}

\textit{indication for I/O on-condition raising or no raising}

\textit{file directory}

\begin{verbatim}
(64) \texttt{int-impl-open(t) =}
    \texttt{int-impl-open-1(t,n);}
    \texttt{n:int-file-option(op);}
    \texttt{op:eval-ref(s-file(t))}
\end{verbatim}

Ref.: eval-ref 8-20 (54)

\begin{verbatim}
(65) \texttt{is-rec-io-st =}
    \texttt{is-delete-st} \quad \texttt{is-locate-st} \quad \texttt{is-read-st} \quad \texttt{is-rewrite-st} \quad \texttt{is-unlock-st} \quad \texttt{is-write-st}
\end{verbatim}

\begin{verbatim}
(66) \texttt{int-file-option(op) =}
    \texttt{is-FILE*s-da*v-eva(op) ->}
    \texttt{PASS:n} \quad \texttt{s-cs}\mu\texttt{(CS;is-onfile-def*s-chif*s-id*(n_f (EN)) (FD) >)}
    \texttt{t -> error}
\end{verbatim}

where: \texttt{n_f = cp-val(op)}

Ref.: \texttt{cp-val 9-9(36)}

\textbf{Note:} The component \texttt{s-onfile-def} provides the onfile condition builtin function with a value if a conversion condition call occurs in the statement causing the implicit opening. The inheritance rules for \texttt{CS} guarantee a dynamic inheritance (cf. chapter 10.3, \texttt{mk-conv-chif}). At the end of any I/O statement the \texttt{s-onfile-def} component will be reset (by execution of the instruction \texttt{revert-onfile}) to the value that the component had before the
interpretation of int-file-option (cf. chapter 6.4, is-co-actions, in case of abnormal end).

(67) \textbf{int-impl-open-1}(t, n) =

\textbf{open-3}(t, f_1, \textbf{cond});
\textbf{cond} = \textbf{open-1}(n, \text{assoc-csa}(\text{ded-attr}(t, \text{UPD} \in \text{attr}_1 \lor \text{REC} \in \text{attr}_1) \land \text{attr}_1), \text{is-def}(f_1), \text{is-enf}(f_1), \text{IMEL}, 0)

where:
\begin{align*}
 f_1 &= n(f_n) \\
 \text{attr}_1 &= s-\text{attr}_1(f_1)
\end{align*}

for: is-n(n)

Note: is-n(f_1)

(68) \textbf{open-3}(t, f, \textbf{cond}) =

is-0(\textbf{cond}) \land is-open-file(f, f_1) \rightarrow \text{PASS}: \text{assoc-u}(f, f_1)

is-TMT-s-\textbf{cond}(\textbf{cond}) \rightarrow

\textbf{error}:
\textbf{call-is-cond-1}(\textbf{cond})

is-UPD-s-\textbf{cond}(\textbf{cond}) \rightarrow

\textbf{verify-impl-cpen}(t, f);
\textbf{call-is-cond-1}(\textbf{cond})

Ref.: \text{call-is-cond-1} 10-24(72)

(69) \textbf{verify-impl-cpen}(t, f) =

\neg is-open-file(f, f_1) \rightarrow

\textbf{error}:
\textbf{call-cond}(\text{ERROR})

\text{REC} \in \text{csa}_1 \land is-rec-file(t, \text{csa}_1) \lor \text{REC} \notin \text{csa}_1 \land is-str-file(t, \text{csa}_1) \rightarrow

\text{PASS}: \text{assoc-u}(f, f_1)

T \rightarrow \text{error}

where:
\text{csa}_1 = s-\text{csa-s-pe}(\text{assoc-u}(f, f_1))

Ref.: \text{call-cond} 10-16(54)

(7) \text{assoc-u}(f, \text{fd}) =

is-ST-FT-s-fd-stf(fd) \rightarrow s-fus-st-prt(fd)

T \rightarrow s-fus-s-fd-stf(fd)
(71) \text{is-open-file}(t, fd) =
    \text{is-n-assoc-u}(t, fd)

(72) \text{is-str-file}(t, csa) =
    \text{is-}(\text{t} \rightarrow \text{DRT} \& \text{csa})
    \text{is-file-put-st}(t) \rightarrow \text{OUT} \& \text{csa}
    \text{is-file-get-st}(t) \rightarrow \text{INP} \& \text{csa}

Note: The first alternative applies to a standard system print file.

(73) \text{is-rec-file}(t, csa) =
    (\text{KEY} \& \text{csa} \rightarrow \text{is-g-em-into}(t) \& \text{is-g-em-ident}(t)) \&
    (\text{BUF} \& \text{cسا} \rightarrow \text{TSA} \& \text{csa} \rightarrow \text{is-g-em-event}(t)) \&
    \text{is-rec-file-1}(t, csa)

(74) \text{is-rec-file-1}(t, csa) =
    \text{is-delete-st}(t) \rightarrow \text{UPD} \& \text{csa} \& (\text{DIR} \& \text{csa} \rightarrow \text{is-g-em-ident}(t))
    \text{is-locate-st}(t) \rightarrow \text{OUT} \& \text{csa} \& \text{BUF} \& \text{csa} \& (\text{KEY} \& \text{csa} \rightarrow \text{is-g-em-ident}(t))
    \text{is-into-read-st}(t) \rightarrow
        \text{OUT} \& \text{csa} \& (\text{DIR} \& \text{csa} \rightarrow \text{is-g-em-ident}(t)) \& (\text{EXC} \& \text{csa} \rightarrow \text{is-g-em-elock}(t))
    \text{is-set-read-st}(t) \rightarrow \text{OUT} \& \text{csa} \& \text{BUF} \& \text{csa}
    \text{is-ignore-read-st}(t) \rightarrow \text{OUT} \& \text{csa} \& \text{DIR} \& \text{csa}
    \text{is-rewrite-st}(t) \rightarrow
        \text{UPD} \& \text{csa} \& (\text{DIR} \& \text{csa} \rightarrow \text{is-g-em-ident}(t)) \& (\text{BUF} \& \text{csa} \rightarrow \text{is-g-em-from}(t))
    \text{is-unlock-st}(t) \rightarrow \text{EXC} \& \text{csa}
    \text{is-write-st}(t) \rightarrow
        \text{INP} \& \text{csa} \& (\text{SEQ} \& \text{csa} \rightarrow \text{UPD} \& \text{csa}) \& (\text{KEY} \& \text{csa} \rightarrow \text{is-g-em-ident}(t))

(75) \text{ded-attr}(t, alt) =
    (\text{is-read-st}(t) \lor \text{is-write-st}(t)) \& alt \rightarrow \{}
    \text{is-file-get-st}(t) \rightarrow \{\text{INP}\}
    \text{is-file-put-st}(t) \rightarrow \{\text{OUT}\}
    \text{is-read-st}(t) \rightarrow \{\text{REC, INP}\}
    \text{is-rewrite-st}(t) \lor \text{is-delete-st}(t) \rightarrow \{\text{UPD}\}
    \text{is-unlock-st}(t) \rightarrow \{\text{EXC}\}
    \text{is-locate-st}(t) \rightarrow \{\text{OUT, BUF}\}
    \text{is-write-st}(t) \rightarrow \{\text{REC, OUT}\}

for: (is-\text{T} \lor is-\text{T})(alt)
11.3.2 Opening of a standard system print file

(76) \texttt{int-stprt-open =}

\texttt{open-3(0, s-stprt, cond);}
\texttt{cond: open-1(0, [CS,OUT,FET], st-id(SYSPRINT), DET-IA-2, IMPL, 0)}

\textbf{Note:} There may be one standard system print file per task. Opening with the system file name \texttt{s-stprt} occurs in connection with the interpretation of a copy option (cf. chapter 11.7.1) or a check standard system action (cf. chapter 11.7.3).

11.3.3 OPENING

Opening proceeds in two steps: The first tests whether a new file is to be opened, and if so makes all entries in the state components; the second step processes data set labels (if applicable) and returns indications for \texttt{MT} or \texttt{UNDP} on-condition raising. The environment attribute may cause additional extra-lingual actions which are unspecified.

\textbf{Metavariables}

- \texttt{title} \hspace{1cm} \texttt{is-id} \hspace{1cm} title of the file to be opened
- \texttt{vol} \hspace{1cm} \texttt{is-IMPL \& is-opt} \hspace{1cm} indication for implicit opening or for opening by an open statement with (*) or without (U) a volume option
- \texttt{cu-ent} \hspace{1cm} \texttt{<s-lsz:is-intg-val \& is-O>, <s-blz:is-opt> <s-err:is-intg-val \& is-O>} \hspace{1cm} evaluated linesize (bitstream or stream) and pagesize

(77) \texttt{open-1(n, csa, title, ea, vol, op-opt) =}

\texttt{is-\&(vol) \& KEY \& csa \rightarrow error}
\texttt{T \rightarrow}
\texttt{open-2(u-1, s, vol)};
\texttt{u-1: open(u, s, st-prt, csa, title, ea, vol, op-opt)};
\texttt{u: \&u-name}

\textbf{where:}
\texttt{l_1 = (is-\&(n) \rightarrow s-stprt,}
\texttt{T \rightarrow s(PN))}
\texttt{st-prt_1 = is-n(n) \& is-INT\&-scope\&-n(\texttt{AT}) \& is-id\&-(n\&(\texttt{PN}))(\texttt{PN}) = mk-id(SYSPRINT) \& [CS,OUT] \& csa}
\texttt{csa_1 = (st-prt_1 \rightarrow [CS,OUT,FET],}
\texttt{T \rightarrow csa)}

cont'd
Ref.: \textit{un-name} 3-10(20)

(78) \texttt{open}(u,f,vol) =
\begin{align*}
&u \neq \text{assoc-}u(f,\text{FD}) \rightarrow \\
&\text{PASS}:\nu_0(<s-f:f>,<s-cond:UND<e>,<s-cncode:chif:code(UND<e>,t)<e>.)> \\
&\text{is-}\Omega(vol) \vee \text{is-INDF(st)}_0 \rightarrow \text{null} \\
&\text{is-SW-BOV}(st)_0 \rightarrow ds-lab(u,BOV) \\
&\text{is-IMPL}(vol) \rightarrow \text{null} \\
&\text{is-}\omega(vol) \rightarrow ds-lab(u,FOV-BOV)
\end{align*}

Ref.: \textit{code} 10-27(80)

Note: Data set label processing by instruction \textit{ds-lab} may return 0 or an indication for TMT on-condition raising.

(79) \texttt{open}(u,f,\texttt{st-prt},\texttt{csa},\texttt{title},ea,vol,\texttt{op-opt}) =
\begin{align*}
&\text{is-open-file}(f_{,}ID) \rightarrow \text{PASS}:\text{assoc-}u(f,\text{ID}) \\
&(\texttt{st-prt} = \text{is-open-file}(s-st-prt,ID)) \& \text{is-csa}(csa) \& \text{is-ds}(ds_1) \& \text{is-decipherable}=0(MF_4,ds_4) \rightarrow \\
&\text{PASS}:u \\
&s-f_1:u(f_1;<s-fd-st*f_1:u_0(<s-fu:u,<s-own:>)>) \\
&s-f_2:u(<u_1(\text{open-fu-sel}(csa,ea,op-opt);<s-p:prt>,<s-f:f><f>)) \\
&s-f_3:u(f_3;<s-ds*ds-sel:cipher(MF_4,\mu(\text{ids}_1;<s-mp-no:s-mp-no(ids_1) + 1>)), > \\
&T \rightarrow \text{PASS}:u
\end{align*}

where:
\begin{align*}
&f_1 = (\texttt{st-prt} \rightarrow s-st-prt, \\
&T \rightarrow f) \\
&f_0 = (\texttt{st-prt} \rightarrow s-st-prt, \\
&T \rightarrow \text{ID}) \\
&\text{ds-sel}_4 = (\text{ds-sel}(\text{title},ea,s-ds-sb(ES))) \& \text{ds-dir} \\
&F_1 = \mu_0(<s-csa:csa>,<s-ea:ea>,<s-title:title>) \\
&ds_1 = s-ds*ds-sel_4(ES) \\
&\text{ids}_4 = \text{decipher}(MF_4,ds_4) \\
&MF_4 = F*MP(f_1)
\end{align*}

Ref.: \textit{is-csa} 11-4(8) \\
\textit{is-ds} 3-27(163) \\
\textit{is-decipherable}=0 11-6(20) \\
\textit{cipher} 11-6(17) \\
\textit{ds-sel} 3-27(105) \\
\textit{decipher} 11-5(15) \\
\textit{p-mp} 11-4(10)

Note: The first alternative applies to opening of an existing file. The second alternative is proper opening. It is taken if the file to be created is not to be linked with an existing standard system print file, and if \texttt{csa} is a complete set of attributes, and if the data set \texttt{ds}_1 is not only effectively available but also available for this particular opening. The third alternative is the kind of improper opening which will yield an \texttt{UND}<e> on-condition call. In the second alternative, the change of the accessed
data set reflects the increasing of the number of files having the same mapping parameter np1 being associated with the data set.

(80) \texttt{open-fu-el(csa,ea,op-opt) =}

\begin{align*}
\text{BST & csa } \land \text{ CST & csa } & \rightarrow \\
\mu_0(<s-col:1>,<s-lsz:lsz_1>,<s-line:line_1>,<s-psz:psz_1>,<s-status:SW-BOV>) \\
\text{REC & csa } \land \text{ BUT & csa } \land \text{ TRA & csa } & \rightarrow \mu_0(<s-ic-ev:{1}>) \\
T & \rightarrow \emptyset
\end{align*}

where:

\begin{align*}
\text{lsz}_1 &= \text{OUT & csa } \rightarrow \text{fu-lsz(csa,s-lsz(cp-opt),s-blz(cp-opt)),} \\
T & \rightarrow \emptyset \\
\text{line}_1 &= \text{FRT & csa } \rightarrow \text{bin-g-op(1)}, \\
T & \rightarrow \emptyset \\
\text{psz}_1 &= \text{FRT & csa } \rightarrow \text{fu-psz(csa,s-psz(op-opt))}, \\
T & \rightarrow \emptyset
\end{align*}

Ref.: bin-g-op 9-10(40)

Note: The fu-status (component s-status) is initialized to SW-BOV. Reading of the data set header label changes the fu-status to BOV (cf. IO-ds-lab), and writing of the data set label changes the status from BOV to 0 (cf. WR-ds-lab). The fu-status must be empty and remains empty for normal I/O data transmission. If the end of the data set volume is reached, reading of the data set trailer label and subsequent writing changes the status from 0 to EOV and to SW. This is the status where data set switching might occur which causes an irreversible status transition to EMD or a transition to the initial status SW-BOV (cf. ds-switch).

(81) \texttt{fu-lsz(csa,i,opt) =}

\begin{align*}
\text{is-intg-val(i)} & \land i < 1 & \land \text{CST & csa } \land \text{is-* (opt)} & \rightarrow \text{error} \\
\text{is-intg-val(i)} & \rightarrow i \\
\text{BST & csa } & \rightarrow \text{DEF-LSZ-1} \\
\text{FRT & csa } & \rightarrow \text{DEF-LSZ-2} \\
T & \rightarrow \text{DEF-LSZ-3}
\end{align*}

Ref.: \text{is-intg-val 9-3(5)}

Note: A bitstream linesize option requires a bitstream file.

(82) \texttt{fu-psz(csa,i) =}

\begin{align*}
\text{is-intg-val(i)} & \land i < 1 & \rightarrow \text{error} \\
\text{is-intg-val(i)} & \rightarrow i \\
T & \rightarrow \text{DEF-PSZ}
\end{align*}

Ref.: \text{is-intg-val 9-3(5)}
11.3.4 CLOSING

Closing of a file occurs through a close statement or at the termination of a task having opened the file. Closing with a volume option performs data set trailer label processing and data set switching but does not delete the file. Proper closing proceeds in three steps: The first makes terminating actions on buffered files or on files having non-completed I/O-events; this step is followed by trailer label processing; in the third step the ID entry and the status component of the ID entry are deleted and any TMT on-conditicis indication originating from data set label processing is returned.

Metavariable

\[ v_{cl} \text{ is-IMPL } v \]
\[ v \text{ is-opt} \]

indication for closing at task termination or for closing by a close statement having a volume option (?) or no volume option (O).

Abbreviation

\[ f_{1} = (\text{is-ST-PRTE-sd-stof}(\text{PT}) \rightarrow s-st-prt, T \rightarrow f) \]
(84) close-1(f, ea, vol) =
   - is-open-file(f, FD) → null
   is-€(vol) & is-*es-owns-fd-st=f1(FD) →
      pass(cond);
      close(u1, f);
      cond: ds-lab-3(u1);
      close-2(u1, f)
   is-* (vol) & KEY 4 s-csa=s-p•u1(FD) → de-lab(u1, EOF)
   T → error

where:
   u1 = assoc-u(f, FD)

Note: This instruction may cause extra-lingual actions which may depend on its arguments, in particular on ea.

(85) close-2(u, f) =
   BUF = csa0 & OUT = csa0 → int-io-st-2(<>, µ0(<s-st:CLOSE-LOCATE>,<s-file:u>), E)
   BUF = csa0 →
      s-g:free-buffer-vr(s-area-p•s-buf•u(FU), s-cos-buf•u(FU), s, PA)
      s-fu:0(FU; s-buf•u)
   is-n-set•s-io-ev(fu-el0) →
      null;
      [delete-task-event(en, AEBNL) | cu = s-io-ev(fu-el0)]
   T → null

Ref.: int-io-st-2 11-41(112)
      free-buffer-vr 11-56(147)
      delete-task-event 5-8(28)

Note: The first alternative performs output transmission of a buffer much like the buffer transmission step of a locate statement. The second alternative frees a buffer which has been allocated by a set or into read statement on a buffered file. The third alternative deletes I/O-events.

(86) de-lab-3(u) =
   is-Ω(st0) & -is-Ω(EA) → de-lab(u, EOF)
   T → null

Note: The trailer label is processed only if it has not been processed previously and if the instruction is not executed in a task epilogue. A task epilogue does not allow on-condition calls.
(87) \[ \text{close}(u, t) = \]
\[ \text{is-decipherable-1}(\text{mp}_0, \text{ds}_0) \rightarrow \]
\[ \text{s-fd}(\text{fd}_1; \text{s-fd-st}_f) \]
\[ \text{s-fy}(\text{fp}_u) \]
\[ \text{s-fp}(\text{fp}_s; \text{e-fp} \ast \text{ds} \ast \text{sel}_r \ast \text{cipher}(\text{mp}_0; \mu(\text{ds}_0; <\text{e-fp} \ast \text{mp} \ast \text{mp}(\text{mp}_0) - 1>))) \]

where:
\[ \text{fd}_1 = (\text{is-ST-PET}\ast \text{s-fd-st}_f(\text{II}) \rightarrow 6(\text{fp}_s; \text{s-fd-st}_f), \]
\[ T \rightarrow \text{fp} \]

Ref.: is-decipherable-1 11-13(43)
cipher 11-6(17)

Note: The diminishing of the number of mapping parameters identical with \( \text{mp}_0 \) is
the inverse process made by the instruction \text{open}.

11.3.5 DATA SET LABEL PROCESSING

Every data set is considered to be a multiple volume data set having a header
and a trailer label. A begin of volume on-condition call processes a header
label, an end of volume call processes a trailer label. This assumption is due to
the modelling of backwards and forward access to a data set (cf. chapter 11.1.4).
Depending on the file attributes the label is read, written or read and written.
Depending on the "condition type" ct label handling is followed by volume
switching (if ct is \( \text{BOV} \)) or followed by volume switching and processing of the
header label of the switched volume (if ct is \( \text{EOV-BOV} \)).

Netvariable

ct \quad \text{is-BOV} \lor \text{is-EOV} \lor \text{condition type}
\text{is-BOV-BOV}

(88) \[ \text{ds-lab}(u, ct) = \]
\[ \text{call-ds-lab-cond}(u, t, \text{lab}) ; \]
\[ \text{hup-name}, \]
\[ \text{lab} : \text{id-ds-lab}(u, ct) \]

Ref.: \text{hup-name} 3-10(20)
(89)  \[ \text{lab-ds} = (\text{is-BOV}(\text{ct}) \land \text{INP} \in \text{CSA}_0 \land \text{DIR} \in \text{CSA}_0 \land \text{TRA} \in \text{CSA}_0 \rightarrow \text{cipher}(\text{MP}_0, \mu(\text{IDS}_0;<\text{s-POS};0>)), \]

\[ T \rightarrow \text{DS}_0) \]

Ref.: is-decipherable-1 11-13(43)

code 10-27(80)
cipher 11-6(17)

Note: Reading of the label is followed by data set positioning.

(90)  \[ \text{cond-ct}(\text{ct}) = \]

\[ (\text{is-BOV}(\text{ct}) \rightarrow \text{BOV}, \]

\[ T \rightarrow \text{EOV}) \]

(91)  \[ \text{lab-sel}(\text{ct}) = \]

\[ (\text{is-BOV}(\text{ct}) \rightarrow \text{s-header}, \]

\[ T \rightarrow \text{s-trailer}) \]
(92) \texttt{call-ds-lab-cond}(u,b,lab) =

\texttt{is-f-cond}(lab) \rightarrow \texttt{PASS:lab}

\texttt{is-char-val-list}(lab) \rightarrow

\texttt{ds-lab-1}(u,ct,tat);
\texttt{tat:=y-ds-lab}(u,ct,op);
\texttt{free}(b);
\texttt{or:gen-op}(gen);
\texttt{call-io-cond-1}(cond);
\texttt{cond:pass-env-bcv-cond}(u,gen,ct);
\texttt{assign}(gen,val-op(ONID-EDA\_1,lab));
\texttt{gen:allocate}(b,aggr-scalar(ONID-EDA\_1),DUMMY)

where:

\texttt{ONID-EDA\_1} = \texttt{char-eda-var}(ONID-LENGTH)

for:\texttt{is-n}(b), (\texttt{is-char-val-list} \land \texttt{is-TMT*is-cond})(lab)

Ref.:
\texttt{is-f-cond} 3-21(78)
\texttt{is-char-val} 9-3(6)
\texttt{free} 7-18(58)
\texttt{gen-cr} 8-22(59)
\texttt{call-io-cond-1} 10-24(72)
\texttt{assign} 8-9(23)
\texttt{val-cr} 9-9(36)
\texttt{allocate} 7-11(25)
\texttt{aggr-scalar} 8-23(64)
\texttt{char-eda-var} 9-12(60)

Note: The begin or end of volume on-condition call is preceded by the creation of a dummy for the data set label. The allocated storage is freed upon normal return. Writing of the data set label, data set switching (if necessary) followed by a possible TMT on-condition call occurs also upon abnormal return from the on-unit (cf. 6.4.).

(93) \texttt{eov-bcv-cond}(u,gen,ct) =

\mu_0(<s-if_{\texttt{e}},<s-cond:cond-ct(\texttt{ct})>,<s-chif:\mu_0(<s-onident:\mu_0(<s-gen:gen>,<s-input:INP * c="a">),<s-ahm-ret:\mu_0(<s-u:u,<s-cond:cond-ct(\texttt{ct})>))))>

for:\texttt{is-gen}(gen)

Ref.: \texttt{is-gen} 3-14(30)

Note: The component \texttt{g-onident} is used for the interpretation of the onident condition builtin function. Assignment to the pseudo variable is illegal in the case of input.
(94) \[ \text{wx-ds-lab}(u, ct, cp) = \]
\[ \text{is-BOV}(ct) \& \text{is-EOV}(st_0) \lor \text{is-BOV}(ct) \& \text{is-EOV}(st_0) \rightarrow \text{error} \]

\[ \text{INF} \in \text{csa}_0 \rightarrow \text{s-fu:}(\text{FU};<\text{s-st}*\text{st}_1>) \]

\[ \text{is-decipherable-1}(mp_0, ds_0) \rightarrow \text{error} \]

\[ \text{is-TMT}(tmt_0) \rightarrow \]

\[ \text{PASS}:\mu_0(\text{s-fi}_{p_0},<\text{s-cond}:\text{TMT},<\text{s-oncode}=\text{s-cbif}:\text{code}(\text{TMT},\xi)>)) \]
\[ \text{s-fu:}(\text{FU};<\text{s-st}*\text{st}_1>) \]
\[ \text{s-fu:}(\text{FU};<\text{s-st}*\text{st}_1>) \]

where:

\[ \text{lab-ds}_2 = (\text{is-BOV}(ct) \& \text{UPD} \in \text{csa}_0 \& \text{STQ} \in \text{csa}_0 \rightarrow \]
\[ \text{cipher}(mp_0, \mu(\text{ids}_0;\text{lab-sel}(ct);\text{op-val}(op)),<\text{s-pos}:0>), T \rightarrow \text{cipher}(mp_0, \mu(\text{ids}_0;\text{lab-sel}(ct);\text{op-val}(op))) \]

\[ \text{st}_1 = (\text{is-BOV}(ct) \rightarrow \varnothing, T \rightarrow 5W) \]

for:is-cp(cp)

Ref.: is-decipherable-1 11-13(83)
code 10-27(80)
cipher 11-6(17)
cp-val 9-9(36)
is-cp 9-9(34)

Note: Writing of the label causes data set positioning in the cases when positioning has not been done by rd-ds-lab.

(95) \[ \text{ds-lab-1}(u, ct, tmt) = \]
\[ \text{is-f-cond}(tmt) \rightarrow \text{PASS}:tmt \]
\[ \text{is-EOV}(ct) \rightarrow \text{null} \]
\[ \text{is-EOV}(ct) \rightarrow \text{ds-switch}(u) \]
\[ \text{is-EOV-EOV}(ct) \rightarrow \]
\[ \text{ds-lab-2}(u); \]
\[ \text{ds-switch}(u) \]

for: (is-\varnothing \lor is-TMT\text{*s-cond})(tmt)

Ref.: is-f-cond 3-21(78)
[53x721]ABSTRACT SYNTAX AND INTERPRETATION OF PL/I

(96) \[ \text{ds-switch}(u) = \]
\[ \text{-is-SW}(st_0) \rightarrow \text{error} \]
\[ \text{KEY} \in \text{CSA}_0 \land \text{OUT} \notin \text{CSA}_0 \land \text{is-last-vol}(\#_p, ds_0, volno_0) \rightarrow \]
\[ \text{s-fu}: \mu(\#; <s-stu:ENDF>) \]
\[ T \rightarrow \text{s-fu}:\mu(\#; <s-stu:SW-BOV>, <s-volnc-u:(is-0(\text{volno}_0) \rightarrow 2, T \rightarrow \text{volno} + 1)>) \]

Note: This is the only instance where a data set is queried whether it is the last volume or not. For a keyed file every volume is considered to be the last, for a non-keyed output file no volume is taken as the last. In fact the file status (fu-status) is switched but no change occurs in the data set. Of course, environmental influence may replace the data set.

(97) \[ \text{ds-lab-2}(u) = \]
\[ \text{is-INDF}(st_0) \rightarrow \text{null} \]
\[ T \rightarrow \text{ds-lab}(u, \text{BOV}) \]

(98) \[ \text{is-last-vol}(\#_p, ds, volno) = \]
\[ \text{for:is-0(volno) \land is-intg-val(volno) \land volno \geq 2, is-\#p(\#_p)} \]

Ref.: \[ \text{is-intg-val 9-3(S)}, \text{is-\#p 11-4(9)} \]

Note: This is an implementation defined predicate which discriminates between the last or an intermediate volume of a data set.
11.4 INITIATION OF I/O STATEMENTS

The initiating actions which occur for all I/O statements (including get and put statements with string option) are defined in this section. Statement options are evaluated and converted if necessary in an implementation defined order, one after the other (cf. 11.4.1). In particular, interpretation of a file option causes implicit opening. Any tests of purely syntactical nature are made before evaluation of the specific option. The evaluated options are inserted into the statement text in place of the unevaluated options ("evaluated text", et).

Section 11.4.2 defines the subsequent actions for record I/O and display statements. Data transmission is initiated by the instruction ic-transmission which refers to section 11.5 (Record Transmission) and section 11.8 (Display Transmission). Before data transmission takes place, the end of file status is interrogated and an I/O event is activated (if the event option is specified in et). After data transmission is completed, I/O on-conditions are raised and the case distinctions concerning the returning or waiting for input (in case of tele-processing data sets) or data set switching (in case if end of volume has been reached) are made.

The definition of the unlock statement is given separately in section 11.4.2.1 since this statement does not cause any data transmission. The stream or bitstream statements get and put are initiated in section 11.4.3.

Metavariables

| t | is-display-st ∧ is-file-get-st ∧ is-file-put-st ∧ is-rec-io-st | text of an I/O statement |
| et | is-e-display-st ∧ is-e-file-get-st ∧ is-e-file-put-st ∧ is-e-rec-io-st ∧ is-close-locate | evaluated text of an I/O statement or an artificial locate statement |
| g | is-gen ∨ is-ps-gen | generation or scalar pseudo generation corresponding to a statement option |

11.4.1 INTERPRETATION OF STATEMENT OPTIONS

Metavariable

| pt | the partially evaluated text of an I/O statement, i.e., an intermediary in the transition from t to et |
ABSTRACT SYNTAX AND INTERPRETATION OF PL/I

(99) \( \text{is-e-[pred]}(et) = \)

\[
(\forall t) (\text{is-[pred]}(t) \land et = \mu(t; \left\{ \begin{array}{l}
\text{is-free-sel-2}(t,a) \land \\
(a \in \{s\text{-into},s\text{-from},s\text{-event}\} \Rightarrow \text{is-gen}(x)) \land \\
(s = s\text{-idto} \Rightarrow \text{is-gen}(x)) \lor \\
(is\text{-ps-gen}(x)) \land \\
(s = s\text{-ident} \Rightarrow \text{is-char-val-list}(x)) \land \\
(s \in \{s\text{-ignore},s\text{-skip},s\text{-line}\} \Rightarrow \text{is-intg-val}(x)) \land \\
(s = s\text{-file}) \Rightarrow \text{is-n}(x)) \right\})
\]

Ref.: is-gen 3-14(39)
      is-ps-gen 12-67(169)
      is-char-val 9-3(6)
      is-intg-val 9-3(5)

Note: This definition schema is used to define the following predicates by proper substitution of all occurrences of "[pred]" by a predicate name the prefix "is-" of which has been deleted:

- is-e-display-st, is-e-file-get-st,
- is-e-file-put-st, is-e-rec-io-st;
- is-e-delete-st, is-e-locate-st, is-e-rewrite-st,
- is-e-unlock-st, is-e-write-st, is-e-read-st;
- is-e-into-read-st, is-e-set-read-st,
- is-e-ignore-read-st.

(100) \( \text{is-free-sel-2}(pt,a) = \)

\[
\text{is-expr}(pt) \land \\
(a \in \{s\text{-into},s\text{-from},s\text{-event},s\text{-idto},s\text{-ident},s\text{-ignore},s\text{-skip},s\text{-line},s\text{-file}\})
\]

(101) \( \text{int-options-1}(t,pt) = \)

\[
(\exists a)(\text{is-free-sel-2}(pt,a) \rightarrow \\
\text{int-options-2}(t,pt,\text{arb}(\{a \mid \text{is-free-sel-2}(pt,a)\}),pt))
\]

\( T \rightarrow \text{PASS}\{pt\} \)

Ref.: \( \text{arb} \ 11-19(51) \)

Note: The argument \( t \) is used only for implicit opening (see \text{int-options-3}) where the unevaluated options are needed. The selection of an arbitrary option is similar with \text{int-open-close-1}.

(102) \( \text{int-options-2}(t,pt,a) = \)

\[
\text{int-options-1}(t,pt-1); \\
\sigma(pt-1):\text{int-options-3}(t,\sigma(pt),a); \\
pt-1: \text{PASS}(pt-1)
\]
\( \text{int-options-3}(t, t-1, s) = \)

\begin{align*}
& s \in \{\text{m-into, m-from}\} \rightarrow \text{eval-ref-gen}(t-1) \\
& s = \text{idto} \& \text{is-CHAF-ref-base}(t-1, at) \vee s = \text{s-event} \& \\
& \text{is-EVENTS} \rightarrow \text{eval-ref}(t-2) \\
& \text{t-2: pre-eval}(t-1) \\
& s = \text{s-ident} \rightarrow \\
& \text{pass-op-eval}(op-1) \\
& \text{cf-1: convert-1}(\text{aggr-scalar}(\text{CHAR-IDB}), op); \\
& \text{op: eval-expr}(t-1, 0) \\
& s \in \{\text{s-ignore}, \text{s-skip}, \text{s-line}\} \rightarrow \text{eval-intg-expr}(t-1) \\
& s = \text{s-file} \rightarrow \text{int-iml-open}(t) \\
& T \rightarrow \text{error} \\
\end{align*}

\text{for: is-exrr}(t-1)

Ref.: \text{eval-ref-gen 8-28(62)} \\
\text{aggr-ref 8-25(71)} \\
\text{eval-le 8-9(20)} \\
\text{pre-eval 8-11(26)} \\
\text{op-val 9-9(36)} \\
\text{convert-1 9-29(119)} \\
\text{aggr-scalar 9-23(60)} \\
\text{eval-expr 8-10(20)} \\
\text{eval-intg-expr 8-22(60)} \\
\text{int-iml-open 11-23(64)}

\text{Note: The evaluation of set options (s-opt) or the evaluation of their defaults is postponed until data transmission (cf. 11.5).}

\( \text{ref-base}(ref, at) = \)

\begin{align*}
& \text{is-var-ref}(ref, at) \rightarrow \text{mk-base-1}(s-\text{da}\text{*aggr-ref}(ref, at)) \\
& \text{is-ref}(ref) \vee \text{is-BUILTIN}(attr_0) \rightarrow \text{nil} \\
& \text{id}_1 \in \{\text{mk-id(SUBSTR)}, \text{mk-id(STRING)}\} \& \text{is-var-ref}(arg_1, at) \rightarrow \\
& \text{mk-base-1}(s-\text{da}\text{*aggr-ref}(arg_1, at)) \\
& \text{id}_1 = \text{mk-id(UNSPEC)} \& \text{is-var-ref}(arg_1, at) \rightarrow \text{BIT} \\
& \text{id}_1 \in \{\text{mk-id(O NSOURCE), mk-id(CONCEDE), mk-id(CNIDENT)}\} \rightarrow \text{CHAR} \\
& T \rightarrow \text{\text{nil}} \\
\end{align*}

\text{where:}

\begin{align*}
& attr_0 = \text{s-n}(ref)(at) \\
& \text{id}_1 = \text{head*id-list}(ref) \\
& \text{arg}_1 = \text{elem}(1, \text{elem}(1, s-ap(ref))) \\
\end{align*}

\text{for: is-exrr(ref), is-at(at) cont'd}
Ref.:  
- is-var-ref 8-9(21)  
- aggr-ref 8-25(71)  
- is-at 3-11(21)  

Note: The function value is CHAR (or BIT) iff ref is a reference which refers to a scalar value representation of character (or bit) class. Assignment of character (or bit) operands to such a reference effectively occurs without conversion.

\[(105)\]

\[
\text{mk-base-1}(s) = \\
\begin{align*}
\text{is-string}(s) &\rightarrow s-\text{base}(s) \\
\text{is-char-p}(s) &\rightarrow \text{CHAR} \\
T &\rightarrow 0
\end{align*}
\]

for: \((\text{is-s} \lor \text{is-0})(s)\)

11.4.2 DISPLAY AND RECORD HANDLING STATEMENTS

This section defines all peripheral actions occurring with display and record I/O statements except options evaluation including implicit opening (cf. 11.4.1) and proper data transmission.

**Metavariables**

- ref-list \(\text{is-ref-list}\)  
  list of references for which check on-condition is to be raised

- cond-list \(\text{is-f-end-cond-list}\)  
  a list of condition arguments for which I/O on-conditions are to be raised

- \(n\) \(\text{is-crt-n}\)  
  a name for the message displayed by a display statement

**Abbreviations**

- \(\text{fu-el}_0 = (\text{is-fu-file} \ast (s-\text{file}(et))) (FU) \rightarrow (s-\text{file}(et)) (FU), T \rightarrow \text{error}\)  

\[F_0 = s-f(fu-el_0)\]

\[\text{csa}_0 = s-csa(t_e)\]

\[ds_0 = s-ds \ast (s-\text{sel}(s-title(F_0), s-\text{ea}(F_0), s-ds-sh(E_S))) \ast s-ds-dir(E_S)\]

\[ep_0 = 1-\text{mp}(F_0)\]

The abbreviations are explained at the beginning of section 11.3.
(106) \[ \text{int-rec-io-st}(t) = \]
\[
\begin{align*}
& \text{is-expres-ident}(t) = \text{is-0\textbullet s-into}(t) & \text{&} & \text{is-0\textbullet s-ident}(t) = \text{is-0\textbullet s-molock}(t) & \text{&} \\
& \text{is-var-ref}(t\text{-into}(t),\text{AT}) = \text{is-connected}(s\text{-into}(t),\text{AT}) & \text{&} \\
& \text{is-var-ref}(t\text{-from}(t),\text{AT}) = \text{is-connected}(s\text{-from}(t),\text{AT}) \rightarrow \\
& \text{revert-onfile}; \\
& \text{int-io-st-1}(t,0) \\
& T \rightarrow \text{error}
\end{align*}
\]

Ref.: \quad \text{is-var-ref 8-9(21)} \\
\quad \text{is-connected 8-34(109)}

Note: Syntactical tests are made before options evaluation is started.

(107) \[ \text{revert-onfile} = \]
\[
\begin{align*}
& \text{s-side}(\text{CS};<\text{s-onfile-def\textbullet s-cbifs\textbullet s-onfile-def}\textbullet s-cbifs\textbullet s-cs(D)>) \\
& \text{Note: This is the resetting of the s-onfile-def component to the value that the} \\
& \text{component had before implicit opening has occurred (cf. int-file-options).} \\
& \text{This instruction is executed at normal or abnormal termination of an I/O} \\
& \text{statement by a goto statement.}
\end{align*}
\]

(108) \[ \text{int-display-st}(t) = \]
\[
\begin{align*}
& \text{is-ref\textbullet s-event}(t) = \text{is-ref\textbullet s-ident}(t) \rightarrow \\
& \text{int-io-st-1}(t,n); \\
& \text{t:in-name} \\
& T \rightarrow \text{error}
\end{align*}
\]

Ref.: \quad \text{in-name 3-19(20)}

(109) \[ \text{int-io-st-1}(t,n) = \]
\[
\text{int-io-st-2}(\text{check-list}(t),et,n); \\
\text{et:in-opts}(1,t)
\]

(110) \[ \text{check-list}(t) = \]
\[
\text{order-set}([t-1] | (\forall a)(t-1 = a(t) \text{ & is-ref\textbullet s}(t) \text{ &} \\
\quad (a \in \{s\text{-into},s\text{-ident}\} \text{ & } s = s\text{-ptr} \text{ & is-READ\textbullet s-st}(t)))},t)
\]

\text{Note: Event options are not checked. Checking of set options on locate} \\
\text{statements occurs like checking by allocate statements (cf. 11.5.1).}
(111) \[ \text{order-set(set, type)} = \]
\[ \text{is-()}(\text{set}) \rightarrow <> \]
\[ \neg\text{is-()} \text{arb(set, type)} \rightarrow \text{arb(set, type)} \}
\[ \text{order-set(set - arb(set, type), type)} \]
\[ \text{for:is-set(set) & (}x\text{) \{x \in \text{set} \rightarrow \neg\text{is-}0(x)\}} \]

Ref.: arb 11-19(51)

Note: This is an implementation defined ordering function which yields a list containing exactly the members from the argument set.

(112) \[ \text{int-ic-st-2(ref-list, et, n)} = \]
\[ \text{is-ns-file(et) \& is-END*s-st(fu-10) \rightarrow} \]
\[ \text{call-ic-cond-1(}x\text{) \{<s-f:s-f(fu-10)>,<s-cond:INDT>\}} \]
\[ \text{is-gen*s-event(et)} \rightarrow \]
\[ \text{attach-ic-event(ref-list, et, n, en); en:un-name} \]
\[ \text{is-e-unlock-} \neg(\text{et)} \rightarrow \text{int-unlock-1(et)} \]
\[ \text{T} \rightarrow \]
\[ \text{call-check-cond(ref-list);} \]
\[ \text{unlock(et);} \]
\[ \text{IC-COND-return(cond-list, et);} \]
\[ \text{IC-COND-iterate-2(cond-list);} \]
\[ \text{cond-list:IC-transmission(et, n)} \]

Ref.: call-ic-cond-1 10-20(72)
\[ \text{is-gen 3-14(30)} \]
\[ \text{un-name 3-1c(20)} \]
\[ \text{call-check-cond 10-21(64)} \]

Note: For a file in end of file status the endfile on-condition will be called before any I/O event is attached. This is not restricted to input files.
(113) \texttt{attach-ic-event}(\texttt{ref-list,et,n,en}) =

\texttt{is-active-event}(s-event(et),PA) \rightarrow \texttt{call-cond}(\texttt{ERROR})

\texttt{is-active}(s-event(et),PA) \rightarrow \texttt{error}

T \rightarrow

\texttt{e-s-el-ass}(ev-\texttt{ev}_0,s-\texttt{pr-e-event}(et),s)

\texttt{e-tc}:\mu(\texttt{en}:\mu_0\langle s-te:u_0\langle s-tn:TN\rangle, s-ev:s-event(et), s-check:ref-list\rangle,\langle s-c:event-transmission(et,n)\rangle) >)

\texttt{e-fa}:\mu(\texttt{en}:\mu_0\langle s-ic-ev:s-ic-ev(fu-el_0) u \{en\} >)

where:

\texttt{ev-ev}_0 = \texttt{rep}(s-eva\ast s-event(et),u_0\langle s-compl:0-BIT\rangle, s-status:0\rangle)

for:is-n(en)

Ref.: \texttt{is-active-event} 5-5(11)

\texttt{call-cond} 10-18(54)

\texttt{is-active} 5-5(12)

\texttt{el-ass} 3-16(48)

\texttt{rep} 9-6(19)

Note: The event name is entered in the \texttt{FU} entry in order to be available for closing (cf. \texttt{close-2}).

(114) \texttt{event-transmission}(et,n) =

\texttt{terminate-transmission}(cond-list,et);

\texttt{cond-list:ic-transmission}(et,n)

(115) \texttt{terminate-transmission}(cond-list,et) =

\texttt{e-pa:activate-tasks}(PA)

\texttt{e-te}:\mu(\texttt{T_1}; s-ev:bow;el_1, s-cond:cond-list_1)>

where:

\texttt{et}_1 = (is-\texttt{U}(i_1) \rightarrow \texttt{U},

T \rightarrow \texttt{ot})

\texttt{i}_1 = erd-index(cond-list)

\texttt{cond-list}_1 = (is-\texttt{U}(i_1) \rightarrow \texttt{cond-list},

T \rightarrow \texttt{remove}(\texttt{cond-list},i_1))

Ref.: \texttt{activate-tasks} 5-12(42)

\texttt{remove} 7-3(5)

Note: The proper data transmission is the same in the \texttt{I/O} event or non-event case. However, special actions are required in order to delay any \texttt{I/O} on-condition calls until a wait for this event will occur. If such a wait occurs, the arguments for \texttt{on-condition} calls are taken from the \texttt{s-cond} component of the event specification (cf. \texttt{ic-cond-iterate}). If the end of the volume has been reached as a result of the data transmission, the data transmission is retried after the data set volume has been switched (cf. \texttt{test-evr-beg}). Notice that the component \texttt{s-check} of the event specification is set by \texttt{attach-ic-event}, and that the components \texttt{s-wait} and \texttt{s-unlock} are set by \texttt{set-wait-state-5} and \texttt{unlock} (cf. section 11.4.2.1).
(116) \texttt{is-transmission}(et, add) =
\begin{align*}
\text{is-e-display-st}(et) \rightarrow \\
\text{pass}(\langle \rangle) ; \\
\text{wait-for-reply}(\texttt{e-into}(et), add) ; \\
\text{display}(\texttt{e-ident}(et), add)
\end{align*}
\begin{align*}
\text{is-e-rec-io-st}(et) \rightarrow \\
\text{pass-order-set}(\texttt{cond-set}, et) ; \\
\texttt{cond-set}: \texttt{rec-io-transmission}(et, add)
\end{align*}
\begin{align*}
\text{is-close-locate}(et) \rightarrow \\
\text{pass-order-set}(\texttt{cond-set}, et) ; \\
\texttt{cond-set}: \texttt{pass-s-cond-set}(\texttt{cond-obj}) ; \\
\texttt{cond-obj}: \texttt{buffer-transmission}(et)
\end{align*}

Ref.:
\begin{align*}
\text{wait-for-reply} & \text{ 11-124(334)} \\
\text{display} & \text{ 11-124(333)} \\
\text{rec-io-transmission} & \text{ 11-54(141)} \\
\text{buffer-transmission} & \text{ 11-55(142)}
\end{align*}

Note: The first alternative applies to the display statement (cf. section 11.2), the second alternative is for all record I/O statements (cf. 11.5) except unlock. The third alternative is the execution of an artificial locate statement which performs buffer freeing and output transmission of the buffer (cf. close-2).

The argument add is a unique name in the case of a display statement; for read statements on transient files, add may be empty or NO-FIND. The argument add is empty in all other cases.

(117) \texttt{is-close-locate} =
\begin{align*}
(\langle \texttt{st:is-CLOSE-LOCATE} \rangle, \\
\langle \texttt{file:ic-n} \rangle)
\end{align*}

(118) \texttt{is-ccond-iterate-2}(cond-list) =
\begin{align*}
\text{is-\langle\rangle}(\texttt{cond-list}_1) \rightarrow \text{null} \\
T \rightarrow \\
\text{is-ccond-iterate-2}(\text{tail}(\texttt{cond-list}_1)) ; \\
\text{call-ic-ccond-1}(\text{head}(\texttt{cond-list}_1))
\end{align*}

where:
\begin{align*}
\texttt{cond-list}_1 & = \langle \texttt{i=0}(i_1) \rightarrow \texttt{cond-list}, \\
T & \rightarrow \text{remove}(\texttt{cond-list}, i_1) \\
i_1 & = \text{end-index}(\texttt{cond-list})
\end{align*}

Ref.:
\begin{align*}
\text{call-ic-ccond-1} & \text{ 10-24(72)} \\
\text{remove} & \text{ 7-3(6)}
\end{align*}
(119) \[ \text{is-f-end-cond} = \]
\[ \text{is-f-cond} \land (\langle s-f:is-n \rangle, \langle s-cond:is-\text{END} \rangle, \langle s-chif:is-chif \rangle) \]

Ref.: \text{is-f-cond} 3-21(78)
\text{is-chif} 3-22(85)

Note: The artificial on-condition END is used as a temporary indication for an end of data set/volume situation, or as an indication of no input by a tele-processing data set.

(120) \[ \text{io-cond-return} (\text{cond-list}, \text{et}) = \]
\[ \text{is-\langle\rangle (cond-list)} \rightarrow \text{null} \]
\[ \text{io-cond-return} (\text{cond-list-1}, \text{et}) = \]
\[ \text{io-cond-iterate-2} (\text{cond-list-1}); \]
\[ \text{cond-list-1:wait-for-input} (\text{et}); \]
\[ \text{call-io-cond-1} (\text{elem} (\text{end-index} (\text{cond-list}), \text{cond-list}); \langle s-cond:PEND \rangle) \]
\[ \text{is-\langle\rangle (cond-list)} \land \text{is-intg-val-end-index} (\text{cond-list}) \rightarrow \]
\[ \text{io-cond-return} (\text{cond-list-1}, \text{et}) = \]
\[ \text{io-cond-iterate-2} (\text{cond-list-1}); \]
\[ \text{cond-list-1:retry-ic-transmission} (\text{cond}, \text{et}); \]
\[ \text{cond:ic-lab} (\text{f-file} (\text{et}), \text{I/O-BOV}) \]
\[ \text{T} \rightarrow \text{null} \]
for: \((\text{is-\langle\rangle} \lor \text{is-f-end-cond-list}) (\text{cond-list})\)

Ref.: \text{is-intg-val} 9-3(5)
\text{call-ic-cond-1} 10-24(72)
\text{ic-lab} 11-37(88)

Note: The second alternative makes the pending on-condition call in the case of no input by a tele-processing data set. The wait following normal return from the on-condition call retries the reading. The third alternative is trailer label processing and data set switching. If switching cannot occur, the file will reach the end of file status (cf. \text{ds-switch}), otherwise the header label of the next volume will be processed and the data transmission is retried. This alternative is also taken if \text{ic-cond-return} is executed in connection with a wait on an I/O event (cf. \text{ic-lab-EOV}).

(127) \[ \text{wait-for-input} (\text{et}) = \]
\[ \text{is transient-input} (\text{et}, \text{IR}, \text{DS}) \rightarrow \text{ic-transmission} (\text{et}, \text{NO-PEND}) \]
\[ \text{T} \rightarrow \]
\[ \text{wait-for-input} (\text{et}); \]
\[ \text{et:wait-state-5} (\text{et}) \]

11. INPUT AND OUTPUT
Note: Input normally occurs through an environmental influence (cf. section 3.7.2, is-ds-dir-step).

(122) \text{ret-wait-state-5}(et) =
\text{is-transient-input}(et,mp,ds) \rightarrow \text{ic-transmission}(et,\text{NO-PEND})

T \rightarrow \text{s-te: $>$ ($>$; $<$-wait:$>$)}

Note: If the first alternative is taken, subsequent reading must be possible. This is guaranteed by the argument \text{NO-PEND}.

(123) \text{is-transient-input}(et,mp,ds) =
\begin{align*}
\text{-is-decipherable-1}(mp,ds) & \rightarrow T \\
\text{is-e-ignore-st}(et) & \rightarrow \text{-is-IND*:in*ignore}(mp,ds,ign) \\
\text{is-e*:ident}(et) & \rightarrow \text{-is-PEND*:in*read}(mp,ds,0) \\
\text{is-prop-key}(mr,ids,\text{s-ident}(et)) & \rightarrow \text{-is-KEY*:in*read}(mp,ds,\text{s-ident}(et)) \\
\text{-is-prop-key}(mp,ids,\text{s-ident}(et)) & \rightarrow T
\end{align*}

where:
\begin{align*}
\text{ids} & = \text{decipher}(mp,ds) \\
\text{ign} & = \left(\text{-is:*e*:ignore}(et) \rightarrow 1, \\
& \quad T \rightarrow \text{s-ignore}(et)\right)
\end{align*}

\text{for:is-mp}(mp),\text{is-ds}(ds)

Ref.: is-decipherable-1 11-13(43)
ignore 11-13(44)
read 11-12(42)
is-prop-key 11-5(13)
decipher 11-5(15)
is-mp 11-8(4)
is-ds 3-27(163)

(124) \text{retry-ic-transmission}(\text{cond},et) =
\begin{align*}
\text{is-f-cond}(\text{cond}) & \rightarrow \text{PASS}:<\text{cond}> \\
\text{is-IND*:st}(fu-el) & \rightarrow \text{PASS}:<\text{mk-cond}(\text{INDF},s-f(fu-el)),\text{s-ident}(et))>
\end{align*}

\begin{align*}
T & \rightarrow \text{ic-transmission}(et,0)
\end{align*}

Ref.: is-f-cond 3-21(78)
mk-cond 11-56(146)

Note: Data transmission is retried only if no cm-conditions (TMT or ENDF) have to be called as a result of label processing or data set volume switching.
(125) \text{end-index}(\text{cond-list}) =

\text{(if)} (\text{is-END}\ast\ast\text{cond-element}(i,\text{cond-list})) \rightarrow (\text{if}) (i \text{-END}\ast\ast\text{cond-element}(i,\text{cond-list}))

\text{T} \rightarrow \text{G}

\text{for:is-intg-val(i)}

\text{Ref.: is-intg-val 9-3(5)}

11.4.2.1 Locking and unlocking by exclusive files

An entry in \(I\) being associated with an exclusive file may have components selected by task names. These components are sets of keys (record identifications). Any record identified by a key which is a member of such a set is said to be locked by that task, the name of which selects the set.

The keys are entered only by the instruction \text{wait-for-unlock} which is executed before data transmission may occur. The deletion of a locked key is done by the instruction \text{iunlock-1}.

Since it provides a notational convenience for the usage of the instructions \text{unlock} and \text{wait-for-unlock}, the instructions are defined also for non-exclusive files. In this case their effect is \text{null}.

\textbf{Metavariables}

\begin{align*}
\text{et} & \quad \text{is-e-unlock-st} \quad \text{evaluated text of an} \\
\text{et} & \quad \text{is-e-read-st} \quad \text{I/O statement} \\
\text{et} & \quad \text{is-e-delete-st} \\
\text{et} & \quad \text{is-e-rewrite-st} \\
\text{et} & \quad \text{is-e-write-st} \\
\text{tn} & \quad \text{is-tn} \quad \text{task name} \\
\text{ident} & \quad \text{is-char-val-list} \quad \text{identifying key of a record} \\
\text{fu-el} & \quad \text{is-fu-file} \quad \text{entry of file union directory}
\end{align*}

\textbf{Abbreviation}

\begin{align*}
\text{tn}_1 = (\text{is-task-te}(T) \rightarrow \text{Tn}, \text{is-io-te}(T) \rightarrow \text{s-tn}(T))
\end{align*}

This is the name of the task for which locked keys are considered as owned.
(126) \[
\text{int-unlock-1(et)} = \text{is-locked-own(fu-el_0, tn_1, s-ident(et))} \rightarrow \text{s-fu-m}\{\text{fn_1; s-file(et)}; \text{tn}_1 (\text{fu-el_0}) - \{s-ident(et)}\}\}
\]

\[
\text{s-fu: activate-tasks(FA)}
\]

\[T \rightarrow \text{null}\]

Ref.: activate-tasks 5-12(42)

Note: \text{is-e-into-read-st(et), tn_1 = TN.}

(127) \text{is-locked-own(fu-el, tn, ident) =}

\[
\text{EXC} \equiv \text{s-csa}\cdot s-p(fu-el) \& \text{is-\~O*tn(fu-el) \& ident \& tn(fu-el)}
\]

(128) \text{is-locked-foreign(fu-el, tn, ident) =}

\[
\text{EXC} \equiv \text{s-csa}\cdot s-p(fu-el) \& (3tn-1)(tn-1 \& \text{tn} \& \text{ident} \& \text{tn-1(fu-el)})
\]

(129) \text{wait-for-unlock(et) =}

\[
\text{is-locked-foreign(fu-el_0, tn_1, s-ident(et))} \rightarrow \text{wait-for-unlock(et); set-wall-state-6(et)}
\]

\[
\text{s-fu-m}(\text{fn_1; s-file(et)}; \text{ident-set_1} \& \{s-ident(et)}\}
\]

\[T \rightarrow \text{null}\]

Where:

\[
\text{ident-set_1} = (\text{is-\~O*(tr_1*s-file(et)}) (10) \rightarrow \{}),
\]

\[T \rightarrow \text{(tn_1*s-file(et)) (12)}\]

Note: This instruction is executed immediately before data transmission takes place by an into read no lock, into read, delete, rewrite, and write statement (cf. chapter 11.5, sec-ic-transmission). A wait actually takes place for an exclusive file, and if the particular key (ident) is locked by a foreign task. Locking actually takes place for an exclusive file, and if the particular key is neither locked in a foreign task nor has the statement no no lock option.

(130) \text{set-wait-state-6(et) =}

\[
\text{is-locked-foreign(fu-el_0, tn_1, s-ident(et))} \rightarrow \text{\_e-tg-m(\text{tn}_1; \text{<e-wait:}>)}
\]

\[T \rightarrow \text{null}\]

Note: This instruction may also set an event specification into the wait state.
(131) unlock(et) =

- is-e-rec-io-st v
  (EXC in CSA) -> is-e-write-st & is-e-rewrite-st & is-e-delete-st) ->
  null

is-s-event(et) -> int-unlock-1(et)
T -> st-te:(T;<e-unlock;et>)

for: (is-e-rec-io-st v is-e-display-st v is-close-locate) (ev)

Note: Unlocking by delete, rewrite, and write statements with event options will be postponed until execution of a wait statement for the event (cf. task-unlock) or until the end of the locking task (cf. unlock-taskkeep).

11.4.3 GET AND PUT STATEMENTS

This section describes the peripheral initiating and terminating actions for the stream I/O statements. The evaluation of any file, line, or skip options is made in section 11.4.1. The evaluation of a string option results in a string type scalar generation or pseudo-generation which is a sub-generation corresponding to a string type scalar reference or a generation corresponding to an allocated dummy. The resulting generation and an index which is initially zero are entered as an entry in the file union directory under a new file union name.

The interpretation of page, line, and skip options refers to the definition of the corresponding control formats. Expansion of the data specification is defined in section 11.6.2.

Metavariables

<table>
<thead>
<tr>
<th>var</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>t</td>
<td>is-get-st v is-put-st v is-e-file-get-st v is-e-file-put-st</td>
</tr>
<tr>
<td></td>
<td>unevaled or evaluated text of a get or put statement</td>
</tr>
<tr>
<td>et</td>
<td>is-e-file-get-st v is-e-file-put-st</td>
</tr>
<tr>
<td></td>
<td>evaluated text of a get or put statement</td>
</tr>
<tr>
<td>spec</td>
<td>is-data-spec v is-u</td>
</tr>
<tr>
<td></td>
<td>data specification consisting of data lists and of optional format lists</td>
</tr>
<tr>
<td>tr</td>
<td>is-tr-par</td>
</tr>
<tr>
<td></td>
<td>transmission parameter</td>
</tr>
<tr>
<td>w</td>
<td>is-intg-val v is-ont</td>
</tr>
<tr>
<td></td>
<td>integer for skip or line option, indication for page or no page option</td>
</tr>
<tr>
<td>u</td>
<td>is-n</td>
</tr>
<tr>
<td></td>
<td>file union name which may refer to a file or string entry in FD</td>
</tr>
</tbody>
</table>
\[(132) \text{int-get-put-st}(t) = \]
\[\text{is-get-spec}(t) \land \text{is-Get-page}(t) \land \text{is-Get-line}(t) \land \text{is-Get-skip}(t) \lor \]
\[\text{is-expr-s-skip}(t) \land (\text{is-Get-page}(t) \lor \text{is-extr-s-line}(t)) \rightarrow \]
\[\text{error} \]
\[\text{is-file-get-st}(t) \land \text{is-file-put-st}(t) \rightarrow \]
\[\text{revert-on-file}: \]
\[\text{int-get-put-file}(:t), \]
\[\text{et: int-options}(:t, t) \]
\[\text{s-base}(t) = \text{ref-base}(\text{s-string}(t), \text{AT}) \rightarrow \]
\[\text{check}(\text{is-PUT}\ast\text{s-st}(t), \text{s-string}(t)); \]
\[\text{expand-spec}(\text{tr}, \text{s-spec}(t)); \]
\[\text{init-string}(\text{tr}, \text{q}); \]
\[\text{s-string}(\text{tr}): \text{un-page}; \]
\[\text{g: eval-lr}(\text{ref}); \]
\[\text{ref: pre-eval}(\text{s-string}(t)); \]
\[\text{tr: pass-tr-cnt}(t) \]
\[\text{is-string-get-st}(t) \rightarrow \]
\[\text{free}(\text{b}); \]
\[\text{expand-spec}(\text{tr}, \text{s-spec}(t)); \]
\[\text{init-string}(\text{tr}, \text{gen}); \]
\[\text{s-string}(\text{tr}): \text{un-name}; \]
\[\text{assign}(\text{gen}, \text{op-1}); \]
\[\text{gen: alloc}(\text{b, eva-1, dummy}); \]
\[\text{b: un-name}, \]
\[\text{eva-1: range-eva}(\text{op-1}); \]
\[\text{oc-1: convert-l}(\text{aggr1}, \text{op}); \]
\[\text{or: eval-expr}(\text{s-string}(t), \text{g}); \]
\[\text{tr: eval-tr-cnt}(t) \]
\[\text{is-string-put-st}(t) \rightarrow \text{error} \]

where:
\[\text{aggr1} = \text{aggr-scaler}(\mu(\text{STB}\ast\text{LEN}; \langle \text{s-base}; \text{s-base}(t) \rangle)) \]

Ref.:
\begin{itemize}
  \item expand-spec 11-74 (206)
  \item un-name 8-10 (20)
  \item eval-lr 8-9 (20)
  \item EES-eval 8-11 (26)
  \item free 7-18 (58)
  \item assign 8-9 (23)
  \item allocate 7-11 (25)
  \item convert-l 9-29 (119)
  \item eval-expr 8-16 (24)
  \item aggr-scaler 8-23 (64)
\end{itemize}

Note: syntactical tests are made before option evaluation is started.
(133) \text{check}(\text{alt},\text{expr}) = \\
\quad \text{alt} \rightarrow \text{call-check-cond}(\text{expr}) \\
\quad T \rightarrow \text{null} \\
\text{for:} (\text{is}-T \land \text{is}-T)(\text{alt}), \text{is-expr-1}(\text{expr}) \\
\text{Ref.:} \quad \begin{align*}
\text{call-check-cond} & \quad 10-21(64) \\
\text{is-expr-1} & \quad 8-16(40)
\end{align*}

(134) \text{int-get-put-file}(\text{et}) = \\
\text{is-INDRES-set}(\text{s-file}(\text{et}))(\text{FV}) \rightarrow \\
\quad \text{call-io-cond-1}(\text{is-fis-t}(\text{s-file}(\text{et}))(\text{FV}), \text{<e-cond:INDP>}) \\
\text{is-PUT-SET}(\text{st}(\text{et}) \rightarrow \\
\quad \text{expand-spec}(\text{tr-par}(\text{et}), \text{s-spec}(\text{et})); \\
\quad \text{init-control}(\text{tr-par}(\text{et}), \text{type}_{1}, \text{w}_{1}); \\
\quad \text{init-control}(\text{tr-par}(\text{et}), \text{PAGE}, \text{PAGE}(\text{et})); \\
\quad \text{err-fu-elem}(\text{s-file}(\text{et}), \text{s-count}, \text{bintg-op}(0)) \\
\text{is-GET-SET}(\text{st}(\text{et}) \rightarrow \\
\quad \text{expand-spec}(\text{tr-car}(\text{et}), \text{spec}(\text{et})); \\
\quad \text{init-control}(\text{tr-par}(\text{et}), \text{SKIP}, \text{SKIP}(\text{et})); \\
\quad \text{err-fu-elem}(\text{s-file}(\text{et}), \text{s-count}, \text{bintg-op}(0))

\text{where:} \\
\text{type}_{1} = (\text{is-intg-val-SET}(\text{skip}(\text{et}) \rightarrow \text{SKIP}, \\
\quad T \rightarrow \text{LINE}) \\
\text{w}_{1} = (\text{is-intg-val-SET}(\text{skip}(\text{et}) \rightarrow \text{<s-skip(\text{et})>}, \\
\quad T \rightarrow \text{s-line}(\text{et}))

\text{Ref.:} \quad \begin{align*}
\text{call-io-cond-1} & \quad 10-24(72) \\
\text{expand-spec} & \quad 11-74(239) \\
\text{bintg-op} & \quad 9-10(40) \\
\text{is-intg-val} & \quad 9-3(5)
\end{align*}

\text{Note:} \quad \text{For any file in end of file status the endfile on-condition will be called.} \\
\text{Skip and line options are mutually exclusive. A skip option is assumed to} \\
\text{specify an integer value in any case.}

(135) \text{err-fu-elem}(u,s,e) = \\
\quad \text{s-fu-elem}(u,(\text{IF};u;\mu(\text{fu-e10};<s;e>)) \\
\text{where:} \\
\text{fu-e10} = (\text{is-fu-e10}(\text{IF}) \rightarrow u(\text{IF}), \\
\quad T \rightarrow \text{error})

\text{Ref.:} \quad \begin{align*}
\text{is-fu-e10} & \quad 3-25(62)
\end{align*}

\text{Note:} \quad \text{The instruction is used in a way that the resulting element of the file} \\
\text{union directory will be correct if it was correct before updating occurred.}
IBM LAB VIENNA

30 April 1969

ABSTRACT SYNTAX AND INTERPRETATION OF PL/I

(136) \( tr-par(t) = \mu(\delta(t; s-base, s-spec, s-page, s-line, s-skip); \)
\( \langle s-type: (is-edit-dir-list \cdot s-spec(t) \rightarrow EDIT, \)
\( T \rightarrow s-type \cdot s-spec(t) \rangle) \)

Note: This function maps the text of a get or put statement into the
"transmission parameter". The transmission parameter \( tr \) plays a similar
role in stream I/O as the evaluated text \( et \) in record I/O. However, in
record I/O normally all options can be evaluated before the proper
interpretation of the statement is started. In stream I/O, the data
specification is subject to subsequent interpretation, and is not part of
\( tr \).

(137) \( is-tr-par = \)
\( is-data-dir \) \( \cdot is-all-data \cdot is-check-data \)

Note: The \( is-data-dir \) is used in those cases where list-, edit-, and data-directed
\( type \) would make no sense (cf. section 11.7.1, \( copy \), and section 11.7.4,
\( syst-end-page-exec \)).

(138) \( is-data-dir = \)
\( is-data \cdot is-all-data \cdot is-check-data \)

Note: The type \( ALL-DATA \) refers to a data list which should contain all
unsubscripted, fully qualified references to proper variables which are not
parameters, and are known in the block where the statement is executed.
\( ALL-DATA \) indicates that the order in the data list is irrelevant (cf.
section 11.6.2.1, order-test-data-list), and that variables which cannot be
converted to character string are to be skipped (cf. section 11.6.3.1,
\( copy-char-string \)).

The type \( DATA \) refers to a data list which is the translation of a data
list present in the concrete text.

The type \( CHECK-DATA \) is used by \( syst-stand-exec \) (cf. section 11.7.3).

(139) \( init-string(tr, g) = \)
\( s-fy: \mu(\delta; \langle s-string(tr) ; \mu(\langle s-g; g \rangle, \langle s-hi; 0 \rangle \rangle) \)
for: \( (is-gen \cdot is-ps-gen) \{ g \} \)

Ref.: \( is-gen \ 3-14(3^c) \)
\( is-ps-gen \ 12-67(165) \)

11. INPUT AND OUTPUT 51
(10?) $init-control(tr, \text{type}, w) =$

- $\text{is-format-match}(\text{fu-el}_1, \text{type}) \rightarrow \text{error}$

- $(\text{is-Skip} \vee \text{is-LINE})(\text{type}) \rightarrow control-format(tr, \mu_0(<\text{e-type}:\text{type}>, <\text{s-w}:w>))$

- $\text{is-PAGE}(\text{type}) \& \text{is-t}(w) \rightarrow \text{put-page}(tr)$

$T \rightarrow \text{null}$

where:

- $\text{fu-el}_1 = (\text{is-n}\ast\text{s-file}(tr) \& \text{is-fu-file}(\text{s-file}(tr)) \rightarrow \text{s-file}(tr) \uparrow_{\text{v}},$
  
- $\text{is-n}\ast\text{s-string}(tr) \rightarrow \text{s-string}(tr) \uparrow_{\text{v}},$

- $T \rightarrow \text{error}$

Ref.: $\text{is-format-match}$ 11-84(232)

$control-format$ 11-82(226)

$\text{put-page}$ 11-98(262)

$\text{is-fu-file}$ 3-25(03)

Note: If the statement option is compatible with the file or string, the option is interpreted like a control format.
11.5 RECORD TRANSMISSION

In this section the record transmission for the write, locate, rewrite, read, and delete statements are defined. The initiating actions for these statements, i.e., syntax tests, evaluation of most of the statement options, explicit opening, and attaching of I/O events, is already described in the foregoing section. The instruction \textit{ic-transmission} of 11.4 constitutes the connection for this section. Its arguments, the evaluated text and one bit of information for transient files, is the basis for the record transmission.

In general, a record transmission consists of the following actions:

- A data set transition causing a modification of the external storage $ES$.
- Freeing and/or allocation of a buffer and/or several assignments, causing a modification of the storage $S$.
- A slight modification of the relevant entry of the file union directory $FH$ on buffered transmissions,
- And the construction of a set of I/O condition indications, which is passed back to the instruction \textit{ic-transmission}.

The further interpretation of the record I/O statements, as e.g. raising of the I/O conditions and data set switching, is described in section 11.4 again.

Common to all types of record transmission is, that the interpretation becomes undefined if the file is in the status of volume switching, or begin and end of volume processing, or in the ultimate end status, i.e., if $-is-G(st_0)$ holds.

\textbf{Metavariables}

$$(is-WC-PEND \lor is-\mathbf{G}) \ (add)$$

is-size(z)

is-co(co)

is-co-set(co-set)

(is-char-val-list \lor is-\mathbf{G}) \ (key)

is-gen(gen)

is-n(f)

\textbf{Abbreviations}

$fu-el_0 = (-is-\mathbf{G} \cdot s-file(et)(F)) \rightarrow s-file(et)(F), \ T \rightarrow \text{error})$

ds-sel_0 = (ds-sel(s-title(p_0), s-ea(p_0), s-ds-sh(ES)) \cdot s-ds-air$

ds_0 = s-ds \cdot ds-sel_0(ES)$

tnt_0 = s-tnt \cdot ds-sel_0(ES)$

d_0 = s-p(fu-el_0)$
\[ \text{cse}_0 = s \text{-cse}(f_0) \]
\[ \text{wr}_0 = r \text{-et}(f_0) \]
\[ \text{buf}_0 = s \text{-buf}(\text{fu-e}_{1_0}) \]
\[ \text{ap}_0 = s \text{-area-r(buf}_0) \]
\[ c_0 = s \text{-o(buf}_0) \]
\[ \text{key}_0 = s \text{-key(buf}_0) \]
\[ \text{st}_0 = s \text{-st(fu-e}_{1_0}) \]
\[ f_0 = s \text{-f(fu-e}_{1_0}) \]

(131) \[ \text{recursive-transmission(et,add)} = \]
\[ \text{is-e-write-st(et)} \to \]
\[ \text{write(condset,condset-1)}; \]
\[ \text{condset}\text{-pass-s-cond-set(cond-obj)}; \]
\[ \text{condset-1}\text{-test-and-write(et,skip)}; \]
\[ \text{skip}\text{-pass-s-skip(cond-obj)}; \]
\[ \text{cond-obj}\text{-buffer-transmission(et)}; \]
\[ \text{wait-for-unlock(et)} \]
\[ \text{is-e-allocate-st(et)} \to \]
\[ \text{pass-s-cond-set(cond-obj)}; \]
\[ \text{test-and-allocate(et,skip)}; \]
\[ \text{skip}\text{-pass-s-skip(cond-obj)}; \]
\[ \text{cond-obj}\text{-buffer-transmission(et)} \]
\[ \text{is-e-rewrite-st(et)} \to \]
\[ \text{rewrite-transmission(et)}; \]
\[ \text{wait-for-unlock(et)} \]
\[ \text{is-e-inter-read-st(et)} \to \]
\[ \text{int-set-transmission(et,add,0)}; \]
\[ \text{wait-for-unlock(et)} \]
\[ \text{is-e-set-read-st(et)} \to \text{is-e-set-read-st(et,add)} \]
\[ \text{is-e-ignore-read-st(et)} \to \text{ignore-transmission(et,add)} \]
\[ \text{is-e-delete-st(et)} \to \]
\[ \text{delete-transmission(et)}; \]
\[ \text{wait-for-unlock(et)} \]

Ref.: \[ \text{wait-for-unlock 11-47(125)} \]
\[ \text{is-e-[pred] 11-36(69)} \]

Note: On a buffered file, a write statement requires two record transmissions. First, the buffer is transmitted and freed by the instruction \text{buffer-transmission}, and second, the proper write transmission is done by \text{test-and-write}. Thence one has to consider, that if a proper performance of the buffer transmission is not possible, the write transmission must not

54 11. INPUT AND OUTPUT
be carried out. Otherwise, the both sets of condition indications are
united before they are passed.

The instruction wait-for-unlock is of no effect, except for exclusive
files (cf. 11.4.2.1).

Similar to the write statement is the interpretation of the locate
statement, in that the allocation of the buffer depends on the success of
the previous buffer transmission.

11.5.1 WRITE AND LOCATE

11.5.1.1 Buffer transmission

This section defines the transmission of buffers into the external storage, the
freeing of buffers in main storage or in areas, and the construction of the set of
condition indications required for all types of record transmissions.

(142) buffer-transmission(et) =

\[
\text{is}=\emptyset (\text{buf}_0) \rightarrow \text{PASS} : \mu_0 (\langle s\text{-cond-set:}\rangle )
\]

\[
\text{is}=\emptyset (\text{st}_0) \rightarrow \text{EXIT}
\]

\[
(s\text{-KEY } \lor s\text{-END}) (s\text{-inf}(\text{write}_1)) \rightarrow
\]

\[
\text{PASS} : \mu_0 (\langle s\text{-cond-set:cond-set}(\langle s\text{-inf}(\text{write}_1), \text{tmt}_0, \text{f}_0, \text{key}_0 \rangle, \langle s\text{-skip:SKIP} \rangle )
\]

\[
\text{COND}: \mu (\langle \text{ds\text{-sel}_0 : \mu_0 (\langle s\text{-ds:ds}(\text{write}_1) \rangle ) \rangle )
\]

\[
T \rightarrow
\]

\[
\text{PASS} : \mu_0 (\langle s\text{-cond-set:cond-set}(\langle \text{opt}\text{-rec}_1, \text{tmt}_0, \text{f}_0, \text{key}_0 \rangle )
\]

\[
\text{COND}: \mu (\langle \text{ds\text{-sel}_0 : \mu_0 (\langle s\text{-ds:ds}(\text{write}_1) \rangle ) \rangle )
\]

\[
\text{FREE}: \mu (\langle \text{buf}\text{-free-file}(\mu_0, \text{opt}_0, \text{ET}) \rangle )
\]

\[
\text{COND}: \delta (\langle \text{FU}: \text{buf}\text{-free-file}(\mu_0, \text{opt}_0, \text{ET}) \rangle )
\]

where:

\[
\text{write}_1 = \text{write}(\mu_0, \text{ds}_0, \text{rec}_0)
\]

\[
\text{rec}_0 = \mu_0 (\langle s\text{-key:ke}_0 \rangle , \langle s\text{-vr:ptr}_0 (\text{ET}) \rangle )
\]

\[
\text{ptr}_0 = (\text{is}=\emptyset (\text{ap}_0) \rightarrow \text{ap}_0)
\]

\[
T \rightarrow \text{ap}_0 \text{-ap}_0
\]

\[
\text{opt}\text{-rec}_1 = (\text{is}\text{-e}\text{-inf}(\text{write}_1) \rightarrow \text{rec}\text{-code}(\text{size-1}(\text{ptr}_0), \text{size}\text{-vr}\text{-size-el}(\mu_0, \text{ds}_0, \text{rec}_0)), T \rightarrow \emptyset)
\]

for: (is-e-write-st \lor is-e-locate-st \lor is-close-locate)(et)

Ref.:

\[
\text{write} \ 11-15(47)
\]

\[
\text{size-1} \ 3-15(38)
\]

\[
\text{size} \ 3-15(33)
\]

\[
\text{size-el} \ 11-10(35)
\]

\[
is\text{-e}\text{-[pred]} \ 11-36(99)
\]

\[
is\text{-close-locate} \ 11-43(117)
\]

Note: If the transmission of the buffer was impossible because of the KEY- or END
indication of the basic accessing function "write" (cf. 11.2), an object is
passed which indicates by its s-skip component that the next action, i.e.,
allocation of a buffer or a write transmission, should be skipped.

(143) cond-set(count-set, f, key) =
Note: This function constructs a set of condition indications. The mapping is completely defined by the following axiom.

\[(144) \text{is-cc} = \text{is-key} \lor \text{is-INT} \lor \text{is-TMT} \lor \text{is-intg-val} \lor \text{is-} \nu\]

Ref. : \text{is-intg-val} 9-3(5)

\[(145) \text{is-set(set}_1) \land \text{card(set}_1) = \text{card(co-set} - \{\nu\}) \land (\forall cc \in \text{co-set} = \text{mk-cond(co-set,cc,f,key)} \lor <\text{-oncode} \lor \text{-chif:co}) \lor \text{set}_1) \land (((\text{is-key} \lor \text{is-INT} \lor \text{is-TMT})(cc) \Rightarrow \text{mk-cond(cc,f,key)} \lor \text{set}_1))\]

where:
\[\text{set}_1 = \text{cond-set(co-set,f,key)}\]

Ref. : \text{is-intg-val} 9-3(5)

\[(146) \text{mk-cond}(\text{io-cond},f,key) = \mu_e(\langle e-f:e, e-\text{cond:io-cond}, e-\text{onkey:e-chif:chif:key} >)\]

Ref. : (is-key \lor is-INT \lor is-TMT \lor is-REC \lor is-TMT)(io-cond)

\[(147) \text{free-buffer-vr}(t,c,vr,pa) = \text{is-} \nu(c) \rightarrow vr\]
\[\text{is-} = \nu (p) \land \neg(\exists \text{gen} \neg(\text{is-indep(s-rn(gen),c) \land is-active(gen,pa)}) \land \text{el-free}(t,c,vr))\]
\[\text{is-prr-val}(p) \& o \in \text{alloc-state}(vr) \land \neg(\exists \text{gen} \neg(\text{is-indep(s-prr(gen),c)} \land \text{is-active(gen,pa)})) \land \text{el-ass}(\text{el-free}(t,c,vr),o,vr)\]
\[T \rightarrow \text{error}\]

Ref. : is-indep 3-16(43)
\[\text{is-active} 5-5(12)\]
\[\text{el-free} 3-17(62)\]
\[\text{is-prr-val} 3-15(36)\]
\[\text{alloc-state} 3-16(53)\]
\[\text{el-ass} 3-16(46)\]
\[\text{is-pa} 5-3(2)\]

Ref. : (is-prr-val \lor is-\nu)(p) \& (is-prr-val \lor is-\nu)(p) \& is-pa(pa)

Note: This function frees storage, specified either by the pointer to an area \(p\) and an offset \(o\) or by a pointer \(o\) if \(is-\nu(p)\). The function value is the size of the piece of storage to be freed correlates with the storage associated with an active task or event variable.
### 11.5.1.2 Write Transmission

**Definition:**

1. **is-ard-write**
   
   \[
   \text{is-ard-write}(et, \text{skip}) = \]
   
   \[
   \begin{align*}
   &\text{is-SKIF}(\text{skip}) \rightarrow \text{PASS:}\phi \text{ } \\
   &T \rightarrow \text{write-transmission}(et) \text{ } \\
   &\text{for:is-e-write-st}(et) \land (\text{is-SKIF} \lor \text{is-0})(\text{skip}) \text{ } \\
   \end{align*}
   \]

2. **write-transmission**

   \[
   \text{write-transmission}(et) = \]
   
   \[
   -\text{is-0}(st_0) \rightarrow \text{error} \text{ } \\
   T \rightarrow \text{PASS:cond-set}((\text{cond}_1, \text{tmt}_0), s_0, s\text{-ident}(et)) \text{ } \\
   s_0 = s_\text{SS}(\text{SS}\text{-main}_0; s_\text{SS}(\text{SS}\text{-wing}(\text{write}_1))) \text{ } \\
   \]

   **where:**

   - \text{write}_1 = \text{write}(s_\text{BP}_0, s_\text{DS}_0, s_\text{rec}_0)
   - \text{rec}_0 = s_\text{SS}(s\text{-key}:s\text{-ident}(et), s\text{-VR}:s\text{-DS-from}(et))(s)
   - \text{cond}_1 = (\text{is-SIF}\text{-inf}(\text{write}_1) \land \text{rec-code}(\text{size}\text{-L}\text{S}\text{-DS-from}(et), \text{size}\text{-VR}\text{-size}\text{-el}(ar_0, ds_0, rec_0)),
   
   \[
   T \rightarrow s\text{-inf}(\text{write}_1) \text{ } \\
   \]

   **for:** is-e-write-st(et)

   **Ref.:**

   - 11-15(47)
   - size-1 3-15(36)
   - size 3-15(33)
   - size-el 11-10(35)
   - is-e-[pred 11-36(99)]

---

**Note:** This function is implementation defined. Its value is an integer value indicating which of the both sizes is greater than the other, to be used for the CMODE built-in function. The function is partially described by the following axiom.

\[
\text{card}([\text{rec-code}(x_1, z_2) \mid x_1 \neq z_2]) = 2 \land \\
(x_1, z_2) (x_1 \neq z_2 \rightarrow \text{is-intg-val}\text{-rec-code}(x_1, z_2) \land \\
\text{rec-code}(x_1, z_2) \neq \text{rec-code}(z_2, x_1))
\]

**Ref.:** is-intg-val 9-3(5)
11.5.1.3 Buffer allocation

In this section the allocation and initialization of the based variable specified in the locate statement is described. The allocation in main storage as well as the allocation in an area is closely related to the corresponding sections of chapter 7. Furthermore, the new pointer value is assigned to the explicit or implicit pointer variable, and the corresponding element of \( T \) is updated by the relevant information about the new buffer.

\[
\begin{align*}
\text{locate}(\text{set-1}, \text{set-2}) &= \text{PASS: set-1 \& set-2} \\
\text{for: is-set(set-1) \& is-set(set-2)}
\end{align*}
\]

(152) \[
\begin{align*}
\text{test-and-allocate}(\text{et, skip}) &= \\
\text{is-SKIP}(\text{skip}) &\rightarrow \text{null} \\
\text{T} &\rightarrow \text{allocate-buffer}(\text{et}) \\
\text{for: is-e-locate-st(}\text{et}) \& \{\text{is-SKIP} \land \text{is-}A\text{(skip)}\}
\end{align*}
\]

Ref.: \text{is-e-[tre]} \ 11-36(99)

(153) \[
\begin{align*}
\text{allocate-buffer}(\text{et}) &= \\
\text{is-based(}\text{attr1}) \land \text{is-var-ref(ptr1, AT)} \land \text{is-PTR\_s-aggr-ref(ptr1, AT)} &\rightarrow \\
\text{initialize}(\text{gen, aggr1}); \\
\text{set-and-refer}(\text{ptr1, gen, eva, aggr1}); \\
\text{gen:allocate-based-buffer(eva, s-file(}\text{et}), s-ident(}\text{et}); \\
\text{eva:eval-alloc-aggr(aggr1)} \\
\text{is-based(}\text{attr1}) \land \text{is-var-ref(ptr1, AT)} \land \text{is-var-ref(area1, AT)} &\rightarrow \\
\text{initialize}(\text{gen, aggr1}); \\
\text{set-and-refer}(\text{ptr1, gen, eva, aggr1}); \\
\text{gen:allocate-area-buffer(gena, eva, area1, s-file(}\text{et}), s-ident(}\text{et}); \\
\text{eva:eval-alloc-aggr(aggr1),} \\
\text{gen:1:eval-ref-gen(area1)} \\
\text{T} &\rightarrow \text{error}
\end{align*}
\]

where:
\[
\begin{align*}
n_1 &= s-p(\text{et}) \\
\text{attr1} &= n_1(AT) \\
\text{aggr1} &= s-aggr(\text{attr1}) \\
\text{ptr1} &= \{\text{is-S-P-PTR(}\text{et}) \rightarrow s-ptr(}\text{et)}, \\
\text{T} &\rightarrow s-ptr(\text{attr1}) \\
\text{area1} &= s-area\_s-aggr-ref(ptr1, AT)
\end{align*}
\]

for: \text{is-e-locate-st(}\text{et})

Ref.: \text{is-var-ref 8-9(21)} \\
\text{aggr-ref 8-25(71)} \\
\text{initialize 7-15(49)}

50 11. INPUT AND OUTPUT
ABSTRACT SYNTAX AND INTERPRETATION OF PL/I

allocate-based-buffer(eva, p, key) =

- is-correct-eva(eva) \land - is-s-buffer(eva) \rightarrow error

is-free-space(p, S, BUFFER) =

PASS: mk-gen(eva, p)
S := el-alloc(p, S, BUFFER)
S := p(eva, S, BUFFER)
where:

\[ p = \text{alloc-space(alloc-size(eva), S, BUFFER)} \]

for: is-eva(eva) \land is-n(n)

Ref.: is-correct-eva 7-5(9)
is-free-space 3-17(56)
mk-gen 7-5(10)
el-alloc 3-17(55)
sto-overflow 7-5(11)
alloc-space 7-11(28)
alloc-size 7-11(26)

allocate-area-buffer(gen, eva, area, n, key) =

- is-area-gen-da(gen) \land - is-correct-eva(eva) \land - is-s-buffer(eva) \rightarrow error

is-free-space(o, e, S, ARBA) =

PASS: ek-gen(eva, o, e, p)
S := el-alloc(e, S, eva, p, S, ARBA)
S := p(S, eva, p, S, ARBA)
T = allocate-area-buffer(gen-1, eva, area, n, key);
where:

\[ o = \text{alloc-space(alloc-size(eva), p, S, ARBA)} \]

for: is-gen(gen) \land is-ev-eva(eva) \land is-ref(area) \land is-n(n)

Ref.: gen-da 7-17(53)
is-correct-eva 7-5(8)
is-free-space 3-17(56)
mk-gen 7-5(10)
el-gen 3-17(54)
el-alloc 3-17(55)
eval-ref-gen 8-29(82)
11.5.2 REWRITE TRANSMISSION

(157) \( \text{rewrite-transmission}(\text{et}) = \)

\[ \neg \text{is-O}(\text{st}_0) \lor \text{is-O-s-from}(\text{et}) \land \text{is-O}(\text{buf}_0) \rightarrow \text{error} \]

\( T \rightarrow \)

\( \text{PASS} : \text{cond-set} \{(\text{cond}_1, \text{tmt}_0), f_0, \text{s-ident}(\text{et})\} \)

\[ \text{s}\text{-et} : \mu : (\exists s : \text{ds} \in \text{et} : \mu_0(\langle s : \text{ds} : \text{ds}(\text{rewrite}_1) \rangle)) \]

where:

\( \text{rewrite}_1 = \text{rewrite}(\text{mp}_0, \text{ds}_0, \text{rec}_0) \)

\( \text{rec}_0 = \mu_0(\langle \text{s-key} : \text{s-ident}(\text{et}) \rangle, \langle \text{s-vr} : \text{ptr}_0(\langle 5 \rangle) \rangle) \)

\( \text{ptr}_0 = (\neg \text{is-O-s-from}(\text{et}) \rightarrow \text{s-pas-s-from}(\text{et}), \neg \text{is-O}(\text{ap}_0) \rightarrow \text{o}_0 \text{sp}_0, T \rightarrow \text{o}_0) \)

\( \text{cord}_1 = (\text{is-REC}\text{s-inf}(\text{rewrite}_1) \rightarrow \text{rec-code}(\text{size}-1(\text{ptr}_0), \text{size}\text{-s-vr}(\text{rec}_1)), T \rightarrow \text{s-inf}(\text{rewrite}_1)) \)

\( \text{rec}_1 = \text{elem}(\text{s-pos}(\text{id}_1), \text{s-data}(\text{id}_1)) \)

\( \text{id}_1 = \text{decipher}(\text{mp}_0, \text{s-ds}(\text{rewrite}_1)) \)

for:is-e-rewrite-st(\text{et})

Ref.: rewrite 11-14(46)
size-1 3-15(36)
size 3-15(33)
decipher 11-5(15)
is-e-[tred] 11-36(69)

Note: If for a buffered file neither the from option nor a buffer exists, then the rewrite statement is in error.

11.5.3 INTO AND SET READ

This section describes the record transmission for the into-read and set-read statements. A case distinction whether a buffer has to be allocated in an area or not is done, rather than a distinction with view to the into and set option. The both main instructions of this section, into-set-transmission and set-transmission, according to the case distinction above, contain a lot of actions, each of which would normally be described in an own instruction. The reason for this fact is, that the entire transmission cannot be interrupted by the execution of another instruction.
(158) \texttt{inter-set-read-st(et,add) =}
\begin{align*}
& \text{is-var-ref(ptr,AT) \& is-ptr\-\texttt{da}\-agg-ref(ptr,AT) \rightarrow} \\
& \text{\texttt{inter-set-transmission(et,add,gen);}} \\
& \text{\texttt{gen=eval-ref-gen(ptr);}} \\
& \text{is-var-ref(ptr,AT) \& is-var-ref(area,AT) \&} \\
& \text{is-area\-da\-agg-ref(area,AT) \rightarrow} \\
& \text{area-test(et,add,gen-1,gen);} \\
& \text{\texttt{cond=eval-transmission(et,add,gen-1,gen-2);}} \\
& \text{\texttt{gen-1=eval-ref-gen(ptr);}} \\
& \text{\texttt{gen-2=eval-ref-gen(area);}} \\
\end{align*}
\[T \rightarrow \text{error}\]
\texttt{where:}
\begin{align*}
\text{ptr} &= \text{e-\texttt{ptr}(et)} \\
\text{area} &= \text{e-area\-da\-agg-ref(ptr,AT)}
\end{align*}
\texttt{for:is-e-set-read-st(et)}

\texttt{Ref.: is-var-ref 8-9(21)}
\begin{align*}
\text{aggr-ref} &= 8-25(71) \\
\text{eval-ref-gen} &= 8-28(82) \\
\text{is-e\-[pred]} &= 11-36(99)
\end{align*}

\textbf{11.5.3. Read without area allocation}

(159) \texttt{inter-set-transmission(et,add,gen) =}
\begin{align*}
\text{\texttt{-is-0(st,0) \& (is-KEY \& is-END) \& is-inf(read,0) \& is-NO-PEND(odd) \rightarrow error}} \\
\text{\texttt{(is-KEY \& is-END) \& is-inf(read,0) \& is-NO-PEND(odd) \rightarrow error}} \\
\text{\texttt{PASS:cond-set\{is-inf(read,0),tmt,0,\texttt{f,0,\texttt{s-ident(et)}}\}}} \\
\text{\texttt{s-eps:ps<ds-select,0<<(ds-ds:ds(read,0))>>}} \\
\text{\texttt{PUI \& csa_0 \& is-free-space(s,\texttt{vr-f,buffer} \rightarrow \texttt{sto-overflow}}} \\
\text{\texttt{-is-e\-into(et) \& is-active(s\-into(et),PA) \& -is-e-s-ptr(et) \&}} \\
\text{\texttt{is-active(gens,PA) \rightarrow error}} \\
\text{\texttt{T \rightarrow}} \\
\text{\texttt{PASS:cond-set\{opt-rec_0,opt-key_0,tmt,0\}f,0,key_0\}}} \\
\text{\texttt{s-eps:ps<ds-select,0<<(ds-ds:ds(read,0))>>}} \\
\text{\texttt{\texttt{s-fu:fu_0}}}}
\end{align*}
\texttt{for:is-e-inte-read-st(et) \& is-\texttt{0}(gen) \& is-e-set-read-st(et) \& is-ptr\-\texttt{da}\-agg-ref(s-ptr(et),AT) \& is-gen(gen)}

\texttt{Ref.: is-free-space 3-17(56)}
\begin{align*}
\text{sto-overflow} &= 7-5(56) \\
\text{is-active} &= 5-5(12)
\end{align*}
\[\text{cont'd}\]
Note: The various abbreviations of this instruction are listed in the following entry.

This instruction covers the into-read case and those set-read cases where the buffer is allocated in main storage.

A record on-condition indication is passed if the size of the record that was read does not coincide with the size of the into variable. Similar, if the key cannot be assigned to the idtc option, a key on-condition indication is passed.

(160) where-cf-into-set-transmission

where:
read \( \hat{=} \) read\( (\text{read}, s, \text{ident}) \)
vr-\( \hat{=} \) vr = vr - read\( (\text{vr}, s) \)
key \( \hat{=} \) key = key - read\( (\text{key}, s) \)
vr-f \( \hat{=} \) vr-f = free-buffer\( (\text{vr}, \text{buff}) \)
p \( \hat{=} \) p = alloc-space\( (\text{size}(\text{vr-r}), \text{vr-f}) \)
vr-b-ass \( \hat{=} \) vr-b-ass = alloc\( (\text{size}(\text{vr-r}), \text{vr-f}) \)
vr-key-ass \( \hat{=} \) vr-key-ass = assign\( (\text{key}, \text{key}) \)
vr-r\( \hat{=} \) vr-r = (\( \text{size}(\text{vr-r}) \) \( \neq \) size into\( (\text{et}) \))
vr-ass \( \hat{=} \) vr-ass = alloc\( (\text{size}(\text{vr-r}), \text{vert}, \text{buff}) \)
vr-ass \( \hat{=} \) vr-ass = alloc\( (\text{size}(\text{vr-r}), \text{vert}, \text{buff}) \)
vr-key-ass \( \hat{=} \) vr-key-ass = assign\( (\text{key}, \text{key}) \)
vr-b-ass \( \hat{=} \) vr-b-ass = alloc\( (\text{size}(\text{vr-r}), \text{vert}, \text{buff}) \)
vr-ass \( \hat{=} \) vr-ass = alloc\( (\text{size}(\text{vr-r}), \text{vert}, \text{buff}) \)
vr-key-ass \( \hat{=} \) vr-key-ass = assign\( (\text{key}, \text{key}) \)
vr-b-ass \( \hat{=} \) vr-b-ass = alloc\( (\text{size}(\text{vr-r}), \text{vert}, \text{buff}) \)
vr-ass \( \hat{=} \) vr-ass = alloc\( (\text{size}(\text{vr-r}), \text{vert}, \text{buff}) \)
vr-key-ass \( \hat{=} \) vr-key-ass = assign\( (\text{key}, \text{key}) \)
vr-b-ass \( \hat{=} \) vr-b-ass = alloc\( (\text{size}(\text{vr-r}), \text{vert}, \text{buff}) \)

Ref.: read 11-12(42)
alloc-space 7-11(28)
size 3-15(13)
el-ass 3-16(48)
el-allcc 3-17(55)
size-1 3-15(38)
rep 5-6(19)

Note: This entry lists the abbreviations of the instruction into-set-transmission above.
(161) assign-Idto(g,cl,chif,vr,pa) =
    is-Ω(v₀) → Ω
    is-active(g₀,pa) → error
    T ← el-ass(rep(s-eva(g₀),v₀),s-pp(g₀),vr)

where:
    v₀ =
        adjust-.substr-1(s-char-eva(g₀),val(s-eva(g₀),s-pp(g₀),vr)),
        elem(2,arg₀),
        elem(i,arg₀),cl)
    arg₀ = s-arg-list·substr-gen(g,1,g,chif)
    g₀ = elem(1,arg₀)

for: (is-gen · is-ps-gen) (g) & is-char-val-list(cl) & is-chif(chif) & is-ps(pa)

Ref.:  is-active 5-5(12)
       el-ass 3-16(48)
       rep 9-6(19)
       adjust-.substr-1 11-93(251)
       val 9-6(18)
       substr-gen 11-52(249)
       is-gen 3-14(31)
       is-ps-gen 12-67(165)
       is-char-val 9-2(65)
       is-chif 3-22(251)
       is-ra 3-3(2)

Note: This function performs the assignment of a key cl to a generation or
      pseudogeneration g with respect to the storage vr. If the assignment is
      possible, the function yields the so modified storage, else the function
      yields Ω unless it is undefined.

(162) size-vr(vr-1,vr-2) =

for:size(vr-1) ≠ size(vr-2)

Ref.:  size 3-15(33)

Note: This function is implementation defined. It is partially described by:

(163) size(vr-1) ≠ size(vr-2) = is-vr·size-vr(vr-1,vr-2) &
    size·size-vr(vr-1,vr-2) = size(vr-2)

Ref.:  size 3-15(33)
       is-vr 3-15(32)
11.5.3.2 **READ with area allocation**

*(164)* \[set-transmission(et, add, gen-1, gen-2) =
\]

\[-is-0(st_0) \vee (is-NIL \vee is-END) (s-inf(read_1)) & is-NO-FEND(add) \mapsto \text{ERROR}\]

\[(is-NIL \vee is-END) (s-inf(read_1)) \mapsto\]

PASS: cond-set \{(s-inf(read_1), txt_1), f_0, s-ident(et)\}
\[s-\theta: \mu(S; <ds-sel_0: \mu_0 (<ds-sis-ds(read_1)>))\]

\[-is-free-space(c_1, p_2 (vr-f_1), \text{AREA}) \mapsto \text{PASS: AREA} \]

\[\text{is-active(gen-1, PA) \mapsto \text{ERROR}}\]

\[T \mapsto\]

PASS: cond-set \{(ctl-key_1, txt_0), f_0, key_1\}
\[s-\theta: \mu(S; <ds-sel_0: \mu_0 (<ds-sis-ds(read_1)>))\]

\[s-\theta: \mu(S; \text{rep(agr-scalar}(0, s-\text{ctxt}))(c_1), s-key\text{(gen-1)}, vr-key-ass_4)\]

\[s-\theta: \mu(S; \text{is-sel-area-file(et); } \mu_0(<s-area-p:p_2>, <s-area-p:p_2>))\]

for \[-e-set-read-st(et) & is-area\text{*s-da\text{*agr-ref(area_1, \text{AT}})\]

*Ref.*: The various abbreviations of this instruction are listed in the following entry.

This instruction defines the transmission for set-read statements where the buffer is to be allocated in an area which is given implicitly by the offset variable in the ptr option.

*(165)* **where-of-set-transmission**

where:

*read_1* = read(agr_0, ds_0, s-ident(et))

*vr-r_1* = s-vr-s-inf(read_1)

*key_1* = s-key-s-inf(read_1)

*vr-f_1* = free-buffer-vr(ap_0, c_0, s, PA)

*p_2* = s-pr(gen-2)

*c_1* = alloc-space(size(vr-f_1), p_2 (vr-f_1), AREA)

*vr-b-ass_2* = el-ass(vr-f_1, c_1, p_2, el-ass(el-alloc(c_1, p_2 (vr-f_1), AREA), p_2, vr-f_1))

*vr-key-ass_3* = assign-idto(s-idto(et), key_1, s-chif(cs), vr-b-ass_2, PA)

*vr-key-ass_3* = (\[is-G(vr-key-ass_3) \mapsto \text{KEY}, \]

\[T \mapsto 0\]

*area_1* = s-area\text{*s-da\text{*agr-ref(s-trt(et), \text{AT}})

*Ref.*: read 11-12(42)

alloc-space 7-11(28)

size 3-15(33)

cont'd
Note: This entry lists the abbreviations of the instruction set-transmission above.

(166) \texttt{area-test}(et,add,gen,cond) =

\begin{align*}
\text{is-AREA}(\text{cond}) & \rightarrow \\
\text{area-test}(et,add,gen,\text{cond}-1); \\
\text{cond}-1 & \text{- set-transmission}(et,add,gen,gen-2); \\
\text{gen-2} & \text{- eval-ref-are}(\text{area}_{1}); \\
\text{call-are}(\text{AREA})
\end{align*}

T \rightarrow \text{PASS:cond}

where:

\begin{align*}
\text{area}_{1} & = s-\text{-area} \ast s-d\ast \text{aggr-ref}(s-ptr(et),A1)
\end{align*}

for: is-e-set-read-st(et) \& is-areas-da\ast aggr-ref(\text{area}_{1},A) \& \\
(is-i-end-cond-set \text{\& is-AREA})(\text{cond})

Ref.: eval-ref-are 8-26(62) \\
call-are 11-18(54) \\
aggr-ref 8-25(71) \\
is-e[pred] 11-36(59) \\
is-i-end-are 11-43(119)

11.5.4 IGNORE READ

(167) \texttt{ignore-transmission}(et,add) =

\begin{align*}
\text{ignore} & < 1 \rightarrow \text{PASS:} \emptyset \\
\text{is-s}(et) & \text{\& is-\text{-FIN}\ast \text{-inf}(ignore) \& is-NO-FIN}(add) \rightarrow \text{ERROR} \\
T & \rightarrow \\
\text{PASS:cond-set}(\text{is-inf}(ignore),tet,0,0) \\
\text{cond-are}(\text{\& s-inf}(et) \ast \text{ds-are}(ignore))
\end{align*}

where:

\begin{align*}
\text{ignore} & = (\text{is-s-\text{-ignore}}(et) \rightarrow 1, \\
T & \rightarrow \text{-s-\text{-ignore}}(et)) \\
\text{ignore} & = \text{ignore}(tet,ds_{0},\text{ignore})
\end{align*}

for: is-e-\text{-ignore-read-st}(et)

Ref.: ignore 11-13(44) \\
is-e[pred] 11-36(59)
(168) \textbf{delete-transmission}(et) =

\begin{align*}
&\neg\text{is-n}(st_0) \rightarrow \text{error} \\
&1 \rightarrow \\
&\text{PASS:cond-set}([s-\text{inf}(\text{delete}_1), \text{mt}t_0], f_0, s-\text{ident}(et)) \\
&\Rightarrow \mu(\text{et} \cdot \text{ds-}\text{select}_0:\mu_0(\langle s-\text{ds}:s-\text{ds}(\text{delete}_1)\rangle)) \\
\end{align*}

\textbf{where:}
\begin{align*}
\text{delete}_1 &= \text{delete}(\#f_0, \text{ds}_0, s-\text{ident}(et)) \\
\text{for:is-n-delete-st}(et)
\end{align*}

\textbf{Ref.:} delete 11-14(45) \\
18-\text{set}\{;\text{id}\} 11-36(75)
11.6 STREAM TRANSMISSION

This section defines further interpretation of get and put statements (cf. section 11.4.3 for the initiation actions). Section 11.6.2 describes the expansion of the data specification, i.e., data list and format list expansion. Stream output and input are defined in sections 11.6.3 and 11.6.4 respectively. The sub-sections on elementary output and input define proper data transmission to and from a string or a file.

The section dealing with syntactical correctness of data fields (11.6.1) has been separated out. It is related with (character) stream input and character string to arithmetic type conversion.

11.6.1 DATA TYPES

In the following a set of one-place predicates over objects is defined. A particular predicate will hold for a particular argument if this argument is a nested list constructed from empty lists or character values as terminals, and if the argument has a certain structuring. Section 11.6.1.1 defines the abstract syntax of constants, section 11.6.1.2 gives the abstract syntax of names used in data-directed input only.

An argument for which a predicate is true is said to be the parsing of the list of characters which results from concatenating all terminals of the argument with the left-to-right ordering preserved. This process of linearization is called unresting (cf. 11.6.1.3).

The conversion of arithmetic data fields to arithmetic type is defined in section 11.6.1.4.

11.6.1.1 Syntax of constants

(169) is-ic-data =
    is-ic-arithm-data \lor is-ic-char-string \lor is-ic-bit-string

(170) is-ic-arithm-data =
    is-ic-arithm-const \lor is-ic-cplx-expr

(171) is-ic-arithm-const =
    (<elem(1):is-<> \lor is-ic-sign>,
     <elem(2):is-ic-real-const>,
     <elem(3):is-<> \lor is-J-CHAR>)

(172) is-ic-cplx-expr =
    (<elem(1):is-<> \lor is-ic-sign>,
     <elem(2):is-ic-real-const>,
     <elem(3):is-ic-real-const>,
     <elem(4):is-ic-real-const>)

(173) is-ic-sign =
    is-POS \lor is-NEG

11. INPUT AND OUTPUT  67
(174) is-ic-char-string = (elem(1):is-APOST,  
(elem(2):is-ic-string-char-list),  
(elem(3):is-APOST)

(175) is-ic-bit-string =  
(elem(1):is-APOST, 
(elem(2):is-ic-bit-char-list), 
(elem(3):is-APOST) 

(176) is-ic-integer =  
(elem(1):is-<> v is-ic-integer), 
(elem(2):is-digit?)

Ref.: is-digit 5-4(5)

(177) is-ic-bit-char =  
is-C-CHAR v is-CHAR

(178) is-ic-real-const =  
is-ic-decimal-fixed v is-ic-decimal-float v is-io-binary-fixed v is-ic-binary-float

(179) is-ic-decimal-fixed =  
(elem(1):is-ic-integer), 
(elem(2):is-* v is-POINT) v  
(elem(1):is-* v is-ic-integer),  
(elem(2):is-POINT) v 
(elem(3):is-ic-integer)

(180) is-ic-decimal-float =  
(elem(1):is-ic-decimal-fixed),  
(elem(2):is-io-exponent?)

(181) is-ic-exponent =  
is-ic-float-exponent v is-editdir-exponent

(182) is-icf-exponent =  
(elem(1):is-C-CHAR,  
(elem(2):is-* v is-ic-sign),  
(elem(3):is-ic-integer))

(183) is-editdir-exponent =  
(elem(1):is-<>),  
(elem(2):is-ic-sign),  
(elem(3):is-ic-integer)}
Note: This type of exponent is legal with F-format only.

(184) is-ic-binary-fixed =
    (<elem(1):is-ic-binary-fixed-part>,
     <elem(2):is-ic-char>)

(185) is-ic-binary-fixed-part =
    (<elem(1):is-ic-binary-integer>,
     <elem(2):is-ic-char>)

(186) is-ic-binary-integer =
    (<elem(1):is-ic-binary-integer>,
     <elem(2):is-ic-bit-char>)

(187) is-ic-binary-float =
    (<elem(1):is-ic-binary-fixed-part>,
     <elem(2):is-ic-exponent>,
     <elem(3):is-ic-char>)

(188) is-ic-string-char(x) =
    (is-char-val(x) \& is-ic-string-apost(x)) & is-ic-apost(x)

Ref.: is-char-val = 3(6)

(189) is-ic-string-apost =
    (<elem(1):is-ic-char>,
     <elem(2):is-ic-char>)

11.5.1.2 Syntax_of_names

(190) is-ic-basic-ref =
    (<elem(1):is-ic-identifier>,
     <elem(2):is-ic-cual-list>,
     <elem(3):is-ic-sub-part \& is-ic-blank-list>)

(191) is-ic-identifier =
    (<elem(1):is-ic-letter>,
     <elem(2):is-ic-alpham-char-list>)

Ref.: is-letter = 3(6)
    is-alpham-char = 3(7)

11. INPUT AND OUTPUT 69
This section defines the notion of parsing a list of character values according to a predicate defining nested lists of character values and empty lists. A prerequisite for parsing is syntactical correctness.

A list of character values is called "left-correct" up to a certain position and according to a particular predicate if there exists at least one continuation which allows parsing.

Metavaribles

<table>
<thead>
<tr>
<th>cl</th>
<th>is-char-val-list</th>
<th>list of character values being subject to parsing</th>
</tr>
</thead>
<tbody>
<tr>
<td>ncl</td>
<td>is-ncl</td>
<td>nested list being the product of parsing</td>
</tr>
<tr>
<td>pred</td>
<td>pred ∈</td>
<td>predicate</td>
</tr>
<tr>
<td></td>
<td>{is-char-num-data, is-char-prop-radio, is-is-basic-ref, is-nl-data, is-e-data, is-f-data}</td>
<td>according to which parsing has to be made</td>
</tr>
<tr>
<td>i,j,k</td>
<td>is-intg-val</td>
<td>list indices</td>
</tr>
</tbody>
</table>
is-ncl =
  is-char-val ∨ is-ncl-list

Ref.: is-char-val 9-3(6)

raising(cl,pred) =
  (is-ncl)(pred(ncl) & cl = unnest-cl(ncl))

unnest-cl(ncl) =
  is-<>(ncl) ↔ <>
  is-char-val(ncl) ↔ <ncl>
  T ↔ unnest-cl(head(ncl))·unnest-cl·tail(ncl)

Ref.: is-char-val 9-3(6)

is-complete-correct(cl,pred) =
  (∃ ncl)(pred(ncl) & cl = unnest-cl(ncl))

is-left-correct(cl,pred) =
  (∃ cl-1)(is-complete-correct(cl·cl-1,pred))

wrong-pos(cl,pred) =
  (li)(i ≥ 1 & (i > length(cl) ∨ ¬is-left-correct(cl·i·1,pred)) ∧
  (Vj)(1 ≤ j < i ⇒ j ≤ length(cl) & is-left-correct(cl·j·1,pred)))

where:
  cl·i·1 = LIST
  k=1
  elem(k,cl)
  j
  cl·j·1 = LIST
  k=1
  elem(k,cl)

Note: The function yields the position of the first wrong character. In particular, if cl is incorrect only because it is too short, the position is length(cl) + 1

11.6.1.4 Arithmetic data fields

The arithmetic data field is checked for syntactical correctness and the numeric value is computed. Notice that the apparent data attributes of the data field are not needed.
(202) \texttt{char-num-conv}(v) =
\begin{align*}
\text{is-compl-correct}(v, \text{is-char-num-data}) \rightarrow \\
\text{PASS: num-val} \cdot \text{elem}(2, \text{parsing}(v, \text{is-char-num-data})) \\
T \rightarrow \\
\text{char-num-conv}(\text{corr-v}) ; \\
\text{corr-v} : = \text{call-conv-cond}(v, \text{wrong-pos}(v, \text{is-char-num-data}))
\end{align*}

Ref.: \texttt{call-conv-cond} 10-24(70)

(203) \text{is-char-num-data} =
\begin{align*}
\langle \text{elem}(1) : \text{is-IFFK-list} > , \\
\langle \text{elem}(2) : \text{is-io-prop-arithm-data} > , \\
\langle \text{elem}(3) : \text{is-IFFK-list} >
\end{align*}

(204) \text{is-io-prop-arithm-data}(dt) =
\begin{align*}
\text{is-io-arithm-data} \& \text{is-e-exponent}(dt)
\end{align*}

(205) \text{is-e-exponent}(dt) =
\begin{align*}
(\forall e) (\text{is-io-decimal-float}(dt) \lor \text{is-io-binary-float}(dt) = \\
\text{is-prop-exponent elem}(2, e(dt)))
\end{align*}

for: \text{is-io-data}(dt)

Note: This is to exclude the type of exponent which is legal with E-format only.

(206) \text{num-val}(dt) =
\begin{align*}
\text{is-io-arithm-const}(dt) \& \text{is-}< (dt_1) \rightarrow \text{real-val}(dt) \\
\text{is-io-arithm-const}(dt) \& \text{is-I-CHAR}(dt_2) \rightarrow \text{cplx}(0, \text{real-val}(dt_1)) \\
\text{is-io-cplx-err}(dt) \rightarrow \text{cplx}(\text{real-val}(dt_1), \text{real-val}(dt_2))
\end{align*}

where:
\begin{align*}
dt_1 &= p(dt; \text{elem}(3):<>>) \\
dt_2 &= \text{elem}(3, dt)
\end{align*}

for: \text{is-io-arithm-data}(dt)

(207) \text{real-val}(dt) =
\begin{align*}
\text{real-val-1}(dt, ^*)
\end{align*}

for: \text{is-io-arithm-const}(dt)
(208)  \[ \text{real-val-1}(dt, \text{base}) = \]
\[ \text{is-expr}(dt) \rightarrow 0 \]
\[ \text{is-digit}(dt) \rightarrow (\text{ui}) (dt = \text{single-numeric-char}(i)) \]
\[ \text{is-ic-arithm-const}(dt) \lor \text{is-ic-exponent}(dt) \rightarrow s_{t_1} \cdot \text{real-val-1}(pr_1, bs_1) \]
\[ \text{is-ic-integer}(dt) \lor \text{is-ic-binary-integer}(dt) \rightarrow \]
\[ \text{base} \cdot \text{real-val-1}(\text{elem}(1, dt), \text{base}) + \text{real-val-1}(\text{elem}(2, dt), \text{base}) \]
\[ \text{is-ic-decimal-fixed}(dt) \lor \text{is-ic-binary-fixed-part}(dt) \rightarrow \]
\[ \text{real-val-1}(\text{elem}(1, dt), \text{base}) + \text{real-val-1}(\text{elem}(3, dt), \text{base}) \cdot \text{base} \cdot \text{li}_1 \]
\[ \text{is-ic-decimal-float}(dt) \lor \text{is-ic-binary-float}(dt) \rightarrow \]
\[ \text{real-val-1}(\text{elem}(1, dt), \text{base}) \cdot \text{base} + \text{real-val-1}(\text{elem}(2, dt), \text{base}) \]

where:
\[ s_{t_1} = (\text{is-MINUS} \cdot \text{head}(dt) \lor \text{is-MINUS} \cdot \text{elem}(2, dt) \rightarrow -1, \]
\[ \text{T} \rightarrow 1) \]
\[ p_{t_1} = (\text{is-ic-exponent} \rightarrow \text{elem}(3, dt), \]
\[ \text{is-ic-binary-fixed-part} \cdot \text{elem}(2, dt) \rightarrow \text{elem}(1, \text{elem}(2, dt)), \]
\[ \text{T} \rightarrow \text{elem}(2, dt)) \]
\[ b_{s_1} = (\text{is-B-CHAR} \cdot \text{elem}(2, \text{elem}(2, dt)) \lor \text{is-B-CHAR} \cdot \text{elem}(3, \text{elem}(2, dt)) \rightarrow 2, \]
\[ \text{T} \rightarrow 10) \]
\[ \text{li}_1 = \text{length} \cdot \text{unnest-char} \cdot \text{elem}(3, dt) \]

for: is-\( \text{expr} \) (base) \lor base = 10 \lor base = 2

Ref.:
- is-digit 9-4(9)
- single-numeric-char 9-32(129)

11.6.2 DATA SPECIFICATIONS

Data specifications may consist of data lists and format lists. A data specification is optional in a get and put statement. The presence of a data specification implies the presence of a data list also in the data-directed case. The presence of a format list is restricted to edit-directed specification.

Data lists may specify iteration rather like do groups, format lists may specify iteration or substitution by means of repetition factors or remote format. The expansion of the data list is nearly independent of the format list; the expansion of the format list is governed by the data list.

Section 11.6.2.1 defines expansion of the data list including the few connections with format list expansion which itself is described in section 11.6.2.2.

Metavariable

tr  is-tr-par  transmission parameter
Abbreviation

fu-el = (is-n*+ file (tr) & is-fu-file (s-file (tr))) (FD) → (s-file (tr)) (FD),
     is-n*+ string (tr) → (s-string (tr)) (FD),
     T → error)

11.6.2.1 Data_list expansion

Dependent on the type of the transmission (s-type (tr)) and on the type of the
statement (s-st (tr)), the instruction expand-spec will result in a series of
instructions put-data, put-edit, put-list, get-data-lp, get-edit, and get-list
which handle single data fields.

This section is particularly related with section 6.5.2 (int-eor^tr-do) and
section 8.1.2 (geo).

Metavariables

<table>
<thead>
<tr>
<th>it1</th>
<th>is-item-list</th>
<th>data list</th>
</tr>
</thead>
<tbody>
<tr>
<td>expr</td>
<td>is-expr-1</td>
<td>pre-evaluated expression, member of the data list</td>
</tr>
<tr>
<td>aggr</td>
<td>is-aggr</td>
<td>aggregate corresponding to expr</td>
</tr>
</tbody>
</table>
(209) \texttt{expand-spec}(tr, spec) =

\hspace{1em} \text{is-}(\langle \rangle spec) \rightarrow \text{null}

\hspace{1em} \text{is-EDIT} = \text{e-type}(tr) \rightarrow

\hspace{2em} \text{expand-spec}(tr, \text{tail}(spec));
\hspace{2em} \text{expand-list}(tr, \text{s-data-list}(\text{head}(spec)));
\hspace{2em} \text{init-format-ci}(\text{s-format-list}(\text{head}(Spec)));

\hspace{1em} -(\text{is-CST} \land \text{is-CFAR}) \rightarrow \text{ERROR}

\hspace{1em} \text{is-LIST} = \text{e-type}(tr) \rightarrow \text{expand-list}(tr, \text{s-data-list}(spec))

\hspace{1em} \text{is-PUT} = \text{e-st}(tr) \rightarrow

\hspace{2em} \text{put-val-list}(tr, \langle \text{SEND} \rangle);
\hspace{2em} \text{expand-list}(tr, \text{order-test-data-list}(\text{s-data-list}(spec), \text{s-type}(spec), \text{AT}))

\hspace{1em} \text{is-GIT} = \text{e-st}(tr) \rightarrow

\hspace{2em} \text{get-data-list}(tr, \text{test-data-list}(\text{s-data-list}(spec), \text{AT}));
\hspace{2em} \text{copy}(tr, \text{f})

\hspace{1em} \text{for:is-data-spec(spec)}

\textbf{Ref.:}

\hspace{1em} \text{put-val-list} 11-91(247)
\hspace{1em} \text{get-data-list} 11-107(297)
\hspace{1em} \text{COPY} 17-127(327)

\textbf{Note:} The first alternative is the terminating step of iterated application of the second alternative. The test made in the third alternative guarantees that list- or data-directed transmission is on (character) stream files (CST) or (character) string files (CFAR) only.

(210) \texttt{init-format-ci}(fcl) =

\hspace{1em} \text{s-ci}(CI; \langle s-fol; pe(\langle s-init; fcl \rangle) \rangle)

\hspace{1em} \text{for:is-format-list-1(fcl)}

\textbf{Note:} The whole current format list is carried on a statement local level in order to be available if wrap-around occurs (cf. first alternative of \texttt{trail-format}).
11.11 ORDER-TEST-DATA-LIST

order-test-data-list(itl, type, at) =

| (211) | \[\text{is-DATA(type)} \land \exists_{i=1}^{\lg_{\mathfrak{A}}} \text{is-data-dir-ref\_elem}(i, \text{itl}) \rightarrow \text{itl} \]
| where: | \[\text{itl\_set}_{\mathfrak{A}} = \{ \text{item} \mid \text{item} = \text{elem}(i, \text{itl}) \land 1 \leq i \leq \lg_{\mathfrak{A}} \} \]
| Ref.: | order-set 11-40(111)
| Note: | The second alternative performs an implementation defined reordering of a data list built by the translator.

| (212) | \[\text{is-data-dir-ref}(\text{ref}, \text{at}) = \]
| where: | \[\text{attr}_{1} = \text{e\_n}(\text{ref})(\text{at}) \]
| Ref.: | is-at 3-11(21)

| (213) | \[\text{test-data-list}(\text{itl}, \text{at}) = \]
| where: | \[\lg_{\mathfrak{A}} = \text{length}(\text{itl}) \]
| Ref.: | base-elem 8-25(85)
| aggr-ref 8-25(71)
(214) \texttt{expand-iter-list}(tr,\texttt{itl}) = \\
\texttt{is-e}(\texttt{itl}) \rightarrow \texttt{null} \\
\texttt{is-contr-item} \rightarrow \texttt{head}(\texttt{itl}) \\
\texttt{expand-iter-list}(tr,\texttt{tail}(\texttt{itl})); \\
\texttt{int-contr-do}(\texttt{head}(\texttt{itl}),tr) \\
\texttt{is-get-st}(tr) \rightarrow \texttt{is-ref-head}(\texttt{itl}) \\
\texttt{expand-iter-list}(tr,\texttt{tail}(\texttt{itl})); \\
\texttt{check}(\texttt{is-get-st}(tr),\texttt{head}(\texttt{itl})); \\
\texttt{expand-exp}(tr,\texttt{expr}); \\
\texttt{expr:pre-eval}(\texttt{head}(\texttt{itl})) \\
T \rightarrow \texttt{error}

Ref.: \texttt{int-contr-do} 6-47(130) \\
\texttt{check} 11-49(133) \\
\texttt{pre-eval} 8-11(26)

Note: Execution of the instruction \texttt{int-contr-do(contr-item, tr)} usually leads to execution of \texttt{expand-iter-list(tr, s-do-list(contr-item))}.

(215) \texttt{expand-exp}(tr,\texttt{expr}) = \\
\texttt{is-FUTs-st}(tr) \& \texttt{is-scalar}(aggr_1) \rightarrow \\
\texttt{count-fut}(tr,\texttt{opt}); \\
\texttt{opt:put-data-field}(tr,\texttt{expr}) \\
\texttt{is-FUTs-st}(tr) \& \texttt{is-scalar}(aggr_1) \rightarrow \\
\texttt{count-convert-assign}(tr,\texttt{g,op,ref-base(\texttt{expr},AT)}); \\
\texttt{expr:put-data-field}(tr); \\
\texttt{g:eval-lp}(\texttt{expr}) \\
\texttt{is-array}(aggr_1) \rightarrow \\
\texttt{iterate-expansion}(tr,\texttt{expr,aggr-1,i}); \\
\texttt{i:pass-s-clb}(aggr-1); \\
\texttt{aggr-1:array-eva-exp}(\texttt{expr}) \\
\texttt{is-struct}(aggr_1) \rightarrow \texttt{iterate-expansion}(tr,\texttt{expr,aggr_1,1})

where: 
\texttt{aggr_1} = \texttt{aggr-expr(\texttt{expr},AT)}

Ref.: \texttt{ref-base} 11-38(160) \\
\texttt{eval-tr} 8-9(20) \\
\texttt{aggr-expr-exp} 8-26(76) \\
\texttt{aggr-expr} 8-25(62)
(216) \texttt{iterate-expansion}(tr, expr, agr, i) =
\begin{align*}
i > \texttt{ubd}(agr) & \rightarrow \texttt{null} \\
T & \rightarrow \\
\texttt{iterate-expansion}(tr, expr, agr, i + 1); \\
\texttt{expand-expr}(tr, expr - 1); \\
\texttt{expr-1:mod}(expr, agr, i)
\end{align*}
\texttt{fcr:is-intg-val}(i)

Ref.:  \texttt{ubd} 7-13(36) \\
\texttt{mod} 8-5(8) \\
\texttt{is-intg-val} 9-3(5)

(217) \texttt{count-mt}(tr, opt) = 
\begin{align*}
is-\texttt{is-string}(tr) \land \texttt{is-}\ast\texttt{(opt)} & \rightarrow \texttt{null} \\
is-\texttt{is-file}(tr) & \rightarrow \\
\texttt{call-ic-cond-1}(\texttt{cond}); \\
\texttt{cond: test-transit}(tr); \\
\texttt{upd-fu-elem}(\texttt{s-file}(tr), s-count, \texttt{bintg-op}); \\
\texttt{bintg-op: add-one}(s-count(\texttt{fu-01}));
\end{align*}
\texttt{fcr:is-cpt}(opt)

Ref.:  \texttt{call-ic-cond} 10-2\texttt{a}(72) \\
\texttt{upd-fu-elem} 11-50(135) \\
\texttt{bintg-op} 9-10(40)

\textbf{Note:} If a data format item or data-directed item is to be skipped, \texttt{gct-data-field} will return \ast as the value of opt.

(218) \texttt{add-cpt}(cp) = 
\begin{align*}
\texttt{convert-1}(s\texttt{-eval}(cp), \texttt{op-1}); \\
\texttt{op-1:eval-infix-expr}(op, \texttt{bintg-op}(1), \texttt{ADD})
\end{align*}
\texttt{fcr:is-bintg-cp}(cp)

Ref.:  \texttt{convert-1} 9-2\texttt{a}(119) \\
\texttt{eval-infix-expr} 5-13(64) \\
\texttt{bintg-cp} 9-10(40) \\
\texttt{is-bintg-cp} 9-10(42)

\textbf{Note:} Increment of count or line-number builtin function value.
(219) \[\text{test-transmit}(tr) = \]
\[\qquad \text{is-*}(\text{tmt-sel}(s-f(tu-el_{1}), FF))(FD) \rightarrow \]
\[\qquad \text{PASS}:(<s-f:s-f(tu-el_{1}>, <s-cond:TMT>) \]
\[\qquad s-fd: FF; \text{tmt-sel}(s-f(tu-el_{1}), FF)) \]
\[\quad T \rightarrow \text{null} \]

Note: The transmission error flag has been set by the instruction \text{stream-transmission} (cf. section 11.6.3.3.1).

(220) \[\text{tmt-sel}(\ell, fd) = \]
\[\qquad \text{is-ST-\text{tmt}*s-\ell*st*f(fd)} \rightarrow s-st-es-\ell*st*-s-st-prt \]
\[\quad T \rightarrow s-st-es-\ell*st*ef \]

for: \( \text{is-s-st-prt} \land \text{is-a}(\mathcal{I}), \text{is-fd(fd)} \)

Ref.: \text{is-s-st-prt} 3-24(90)
\text{is-fd} 3-24(86)

(221) \[\text{put-data-field}(tr, expr) = \]
\[\qquad (\text{is-label-const} \lor \text{is-entry-const})(\text{attr}_{1}) \rightarrow \text{put-data}(tr, s-id-list(expr),<>,9) \]
\[\text{is-data-dir*es-type}(tr) \rightarrow \]
\[\qquad \text{put-data}(tr, s-id-list(expr),isl,op); \]
\[\qquad \text{isl:pass-eval-sl}(\text{aggri}, rl); \]
\[\qquad \text{op:gen-op}(\text{gen}); \]
\[\qquad \text{gen:eval-sub-gen}(\text{expr}, rl); \]
\[\qquad \text{rl:mk-rl}(s-sl(\text{expr}), egl,s-eva(\text{gen}_{1})) \]
\[\text{is-EDIT*e-type}(tr) \rightarrow \]
\[\qquad \text{put-edit}(tr, op, edfo); \]
\[\qquad \text{edfc:iterate-control-format}(tr, 0); \]
\[\text{op:eval-expr-2}(expr) \]
\[\text{is-LIST*e-type}(tr) \rightarrow \]
\[\qquad \text{put-list}(tr, op); \]
\[\text{op:eval-expr-2}(expr) \]

where:
\[\text{attr}_{1} = s-n(\text{expr})(AT) \]
\[\text{egl}_{1} = \text{eval-cl}(\text{tail}s-id-list(\text{expr}), s-agg(\text{attr}_{1})) \]
\[\text{gen}_{1} = \text{head}(s-n(\text{expr})(FF)(AG)) \]

Ref.: \text{put-data} 11-88(240)
\text{is-data-dir} 11-51(138)
\text{gen-op} 8-22(59)
\text{eval-sub-gen} 8-29(87)
\text{mk-rl} 8-30(61)
\text{put-edit} 11-98(263)
\text{eval-expr-2} 8-19(50)
\text{put-list} 11-87(237)
\text{eval-cl} 8-30(94)
Note: The first alternative is taken only if is-CHECK-DATA•s-type(tr) holds.

(222) eval-s1(aggr,rl) =
  is-array(aggr) ← head(rl)>eval-s1(s-elem(aggr),tail(rl))
  is-struct(aggr) ← eval-s1(s-aggr-elem(head(rl),aggr),tail(rl))
  is-scalar(aggr) ← <>

for:is-intg-val-list(rl)

Ref.: is-intg-val 9-3(5)

(223) count-convert-assign(tr,q,op,base) =
  is-FMT•s-type(tr) & is-<>(cp) & is-<(base) ←
  count-convert-assign(tr,q,eval-op(p0(<p-base:base>,<s-length:0>),<>),base)
  is-* (op) v is-<>(op) ← null
  is-ses-string(tr) ← convert-assign(g,op,0,0)
  is-ses-file(tr) ←
  call-ic-cond-1(cond);
  cond: test-true(tr);
  convert-assign(g,op,0,0);
  ses-elem(s-file(tr),s-count,hint-oq);
  hint-oq: add-one(s-count(fu-el1))

for:is-* v is-<> v is-op (cp), (is-gen v ir-ps-gen) (q), (is-f v is-CHAR v
  is-FMT) (base)

Ref.: val-op 9-9(38)
  convert-assign 8-8(16)
  call-ic-cond-1 10-24(72)
  upd-elem 11-50(135)
  hint-oq 9-10(40)
  is-op 9-9(34)
  is-gen 3-14(30)
  ir-ps-gen 12-67(169)

Note: If a data format item is to be skipped, get-data-field will return * or <>
  as the value of op; in the case of list-directed input, skipping is
  indicated by *.
\textbf{Abstract Syntax and Interpretation of FL/I}

\begin{equation}
\text{get-data-field}(tr) = \\
\quad \text{is-EDIT}s\text{-type}(tr) \rightarrow \\
\quad \text{get-edit}(tr, edfc); \\
\quad \text{edfc}\text{-iterate-control-format}(tr, 0)
\end{equation}

\begin{equation}
\text{is-LIST}s\text{-type}(tr) \rightarrow \\
\quad \text{get-list}(tr); \\
\quad \text{copy}(tr, 0)
\end{equation}

\textbf{Metavariables}

\begin{itemize}
\item \texttt{edfc} \quad \text{evaluated format iter or } 0
\item \texttt{edfc} \quad \text{evaluated data format iter}
\item \texttt{fcl} \quad \text{is-format-1-list} \quad \text{(unevaluated) format list which may contain remote format extent flags}
\end{itemize}

\textbf{Abbreviations}

\begin{itemize}
\item \texttt{f\_t} = s\text{-type (edfc)}
\item \texttt{w\_1} = s\text{-w (edfc)}
\item \texttt{w\_2} = (\text{is-}0(w\_1) \lor w\_1 < 1 \rightarrow 1, 7 \rightarrow w\_1)
\end{itemize}

\textbf{11. Input and Output} 81
(225) \texttt{iterate-control-format}(tr, efo) =
\begin{verbatim}
is-0(efo) ->
iterate-control-format(tr, efo-1); efo-1:eval-format(tr)
\end{verbatim}
\begin{verbatim}
e-type(efo) \in \{III, ILT, CPLX, CHAR, FIT, PIC, BEIT, EPIC\} ->
pass(efo):
\end{verbatim}
\begin{verbatim}
copy(tr, x)
\end{verbatim}
T ->
iterate-control-format(tr, x);
control-format(tr, efo)
\end{verbatim}
\begin{verbatim}
Ref.: copy 11-121(327)
\end{verbatim}

	extbf{Note}: Control format items are evaluated and executed as long as no data format item is encountered. The evaluated data format item is returned.

(226) \texttt{control-format}(tr, efo) =
\begin{verbatim}
is-SKIP(ft1) \& is-intg-val(w1) \& w1 < 1 \& is-CST-stream-base(fu-el1) -> error
is-GET-st(tr) ->
get-control-format(tr, efo):
\end{verbatim}
\begin{verbatim}
copy(tr, x)
is-PUT*st(tr) -> put-control-format(tr, efo)
\end{verbatim}
\begin{verbatim}
Ref.: is-intg-val 9-3(5)
\end{verbatim}
\begin{verbatim}
copy 11-121(327)
\end{verbatim}

\begin{verbatim}
Ref.: This instruction executes a single control format item. The consistency test between the file or the string and the format has been made by the predicate is-format-match in eval-format.
\end{verbatim}

(227) \texttt{get-control-format}(tr, efo) =
\begin{verbatim}
(is-BSPACE \& is-SPACT)(ft1) -> get-space(tr, w1)
(is-RCCL \& is-COL)(ft1) -> get-column(tr, w2, x)
\end{verbatim}
\begin{verbatim}
is-SKIP(ft1) -> get-skip(tr, w2, x)
\end{verbatim}
\begin{verbatim}
Ref.: get-space 11-114(308)
get-column 11-114(31C)
get-skip 11-114(309)
\end{verbatim}
(228) \texttt{put-control-format}(\texttt{tr},\texttt{efo}) =
\begin{align*}
& \quad (\text{is-SPACE} \lor \text{is-SPACE}) (ft_1) \rightarrow \text{put-space}(\text{tr},w_1) \\
& \quad (\text{is-BCOL} \lor \text{is-COL}) (ft_1) \rightarrow \text{put-column}(\text{tr},w_1) \\
& \quad \text{is-SKIP}(ft_1) \land w_1 < 1 \rightarrow \text{put-file}(\text{tr},1,\text{CR}) \\
& \quad \text{is-SKIP}(ft_1) \rightarrow \text{put-skip}(\text{tr},w_2,0) \\
& \quad \text{is-LINE}(ft_1) \rightarrow \text{put-line}(\text{tr},w_2,0) \\
& \quad \text{is-PAGE}(ft_1) \rightarrow \text{put-page}(\text{tr})
\end{align*}

Ref.:
\begin{align*}
& \text{put-space} \ 11-96(257) \\
& \text{put-column} \ 11-97(258) \\
& \text{put-file} \ 11-94(253) \\
& \text{put-skip} \ 11-97(259) \\
& \text{put-line} \ 11-97(261) \\
& \text{put-page} \ 11-99(262)
\end{align*}

(229) \texttt{eval-format}(\texttt{tr}) =
\begin{align*}
& \quad (\text{is-0} \lor \text{is-<}) (s\text{-expand-s-fol}(\text{CI})) \rightarrow \text{upd-format-ci}(s\text{-init-s-fol}(\text{CI})) \\
& \quad \text{is-format-iter}(\texttt{fo}_2) \rightarrow
\end{align*}
\begin{align*}
& \quad \text{upd-format-ci}(\text{fo}_1); \\
& \quad \text{fo}_1:\text{pass-rep-conc}(i,s\text{-format-list}(\text{fo}_1),\text{fo}_2); \\
& \quad \text{i}:\text{eval-intq-expr}(s\text{-rep}(\text{fo}_1)); \\
& \quad \text{is-id}(\text{fo}_1) \rightarrow \text{upd-format-ci}(\text{fo}_2) \\
& \quad \text{is-remote-format}(\text{fo}_1) \rightarrow
\end{align*}
\begin{align*}
& \quad \text{upd-format-ci}(\text{fo}_1); \\
& \quad \text{fo}_1:\text{pass-rep-conc}(1,\text{fo}_1,\text{fo}_2); \\
& \quad \text{fo}_1:\text{verify-remote-format}(0,\text{fo}_2); \\
& \quad \text{i}:\text{pass-op-val}(\text{op}); \\
& \quad \text{op}:\text{eval-ref}(\text{efo}); \\
& \quad \text{is-format-match}(\text{fu-el}_1,\text{s-type}(\text{fo}_4)) \& (\text{is-PAGE}\&\text{s-type}(\text{fo}_4) \Rightarrow \text{is-0}\&\text{s-w}(\text{fo}_4)) \& \\
& \quad (\text{s-type}(\text{fo}_4) \in \{\text{SPACE},\text{BCOL},\text{COL},\text{ECOL}\} \Rightarrow \text{is-0}\&\text{s-w}(\text{fo}_4)) \& \\
& \quad (\text{is-GET}\&\text{s-st}(\text{tr}) \Rightarrow (\text{va}) (\text{is-FLT}\&\text{s-type}\&\text{a}(\text{fo}_4) \Rightarrow \text{is-0}\&\text{p}\&\text{s-e}(\text{fo}_4))) \rightarrow
\end{align*}
\begin{align*}
& \quad \text{pass}(\text{efo}); \\
& \quad \text{upd-format-ci}(\text{fo}_2); \\
& \quad \{s(\text{efo})\}:\text{pass}(s(\text{fo}_4)) \mid (\text{s-1}) (s = \text{s-type-s-1}) \& \text{is-0}\&\text{s}(\text{fo}_4)) \lor \\
& \quad \{s(\text{efo})\}:\text{eval-intq-expr}(s(\text{fo}_1)) \mid \\
& \quad (\text{s-1}) (s = \text{s-w}\&s-1 \lor e = \text{s-0}\&s-1 \lor e = \text{s-p}\&s-1) \& \text{is-0}\&\text{s}(\text{fo}_4))
\end{align*}

\begin{align*}
T & \rightarrow \text{ERROR}
\end{align*}

where:
\begin{align*}
& \text{fo}_4 = \text{head}\&\text{s-expand}\&\text{s-fol}(\text{CI}) \\
& \text{fo}_2 = \text{tail}\&\text{s-expand}\&\text{s-fol}(\text{CI})
\end{align*}

Ref.:
\begin{align*}
& \text{rep-conc} \ 7-17(54) \\
& \text{eval-intq-expr} \ 8-22(60) \\
& \text{op-wal} \ 9-9(36) \\
& \text{eval-ref} \ 8-20(54)
\end{align*}
Note: The third alternative is taken only if a remote format item has been previously substituted into the format list under expansion. Such a substituted remote format item is followed by an identifier, the remote format extent flag. The fifth alternative performs syntactical checking of the format item to be evaluated. It tests also whether the format item is compatible with the file or the string.

(230) \[ \text{upd-format-ci}(\text{fol}) = \]

\[ \text{s-ci} : \mu (\text{ol}; <\text{expand}; \text{f-ol}; \text{fol}) \]

(231) \[ \text{verify-remote-format}(\text{n}, \text{fol}) = \]

\[ \text{is-n}(\text{n}) \land \text{is-format-den}(\text{den}_1) \land \text{s-ba}(\text{den}_1) = \text{MA} \land \text{s-spp}(\text{CS}) = \text{s-sun}(\text{den}_1) \land \text{length}(\text{fol}) \]

\[ \text{Et} \quad \text{s-ident}(\text{den}_1) \neq \text{elem}(\text{i}, \text{fol}) \rightarrow \]

\[ \text{PASS} : \text{c-format-list}(\text{den}_1) \wedge <\text{s-ident}(\text{den}_1)> \]

\[ \text{T} \rightarrow \text{error} \]

where:

\[ \text{den}_1 = \text{n}(\text{EN}) \]

for:is-value(\text{v})

Ref.: \text{is-format-den} 3-12(26)
      \text{is-value} 9-3(2)

Note: The components \text{s-ba} and \text{s-ident} of a format label denotation serve as an indication for the block activation in which the label is declared, and as a unique identification of the remote format in case of multiple label prefixes. Substitution of the remote format item is substitution of the remote format list followed by the extent flag.

(232) \[ \text{is-format-match}(\text{fu-el}, \text{type}) = \]

\[ \text{is-pcol}(\text{type}) \rightarrow \text{is-est-stream-base}(\text{fu-el}) \]

\[ \text{type} \in \{\text{BBIT, BPIC, BSPACE}\} \rightarrow (\text{is-est} \lor \text{is-bit})(\text{stream-base}(\text{fu-el})) \]

\[ \text{type} \in \{\text{PAGF, LINT}\} \rightarrow \text{is-pet-stream-base}(\text{fu-el}) \]

\[ \text{type} \in \{\text{SKIP, CCL}\} \rightarrow (\text{is-cst} \lor \text{is-pet})(\text{stream-base}(\text{fu-el})) \]

\[ \text{type} \in \{\text{FIX, ILT, CFI, CHAR, EIT, FIC, SPACE}\} \rightarrow \]

\[ (\text{is-cst} \lor \text{is-pet} \lor \text{is-char})(\text{stream-base}(\text{fu-el})) \]

Note: The predicate is not applied to remote format type.
(233) \[ \text{stream-base(fu-el)} = \]

\[ \text{is-fu-string(fu-el)} \rightarrow \text{gen-base}s\cdot q(fu-el) \]

\[ \text{BST} \cdot \text{csa}_1 \rightarrow \text{FRT} \]

\[ \text{CST} \cdot \text{csa}_1 \rightarrow \text{CST} \]

\[ \text{BST} \cdot \text{csa}_1 \rightarrow \text{EST} \]

\[ \text{where:} \]

\[ \text{csa}_1 = s\cdot c\cdot r\cdot a(s\cdot p(fu-el)) \]

\[ \text{for: is-fu-el(fu-el)} \]

Ref.: is-fu-string 3-26(99)

is-fu-el 3-25(92)

(234) \[ \text{gen-base}(g) = \]

\[ \text{is-gen}(g) \rightarrow \text{sk-base}\cdot \text{t}s\cdot \text{d}a\cdot \text{s-eva}(g) \]

\[ r\cdot \text{id}(g) \in \{\text{sk-id(SUBSTR)},\text{sk-id(STRING)},\text{sk-id(UWSFIC)}\} \rightarrow \]

\[ \text{sk-base}\cdot \text{t}s\cdot \text{d}a\cdot \text{s-eva}@\text{heads-arg-list}(g) \]

\[ T \rightarrow \text{CHAR} \]

\[ \text{for: (is-gen \lor is-ps-gen)}(g) \]

Ref.: is-gen 3-15(33)

sk-base-1 11-39(105)

is-ps-gen 12-67(169)

\[ \text{Note: This function is applied only to a very restricted class of arguments} \]

(cf. ref-base). In the domain where the function is used, the range is BIT or CHAR.
(235) \[ \text{data-lg(efo)} = \]
\[ \text{ft}_0 \in \{ \text{CHAR,BIT,BBT,FIX,FLT} \} \rightarrow \text{s-w(efo)} \]
\[ (\text{is-PIC(ft}_0) \lor \text{is-BPIC(ft}_0) \land \text{is-bin-pics-pic(efo)}) \land \]
\[ \text{is-correct-pics-pic(efo)} \land \text{-is-CPLX-s-mode-s-pic(efo)} \rightarrow \]
\[ \text{pic-length-s-pic(efo)} \]
\[ \text{is-CPLX(efo)} \land (\text{is-BPIC(efo)} \equiv \text{is-BPIC(efo)}) \land \]
\[ (\text{is-PIC(efo)} \lor \text{is-dec-pics-pic(effo)}) \land \]
\[ (\text{is-PIC(efo)} \lor \text{is-dec-pics-pic(effo)}) \rightarrow \]
\[ (\text{data-lg}(\text{ft}_1) = 0 \land \text{data-lg}(\text{ft}_2) = 0 \rightarrow \]
\[ 0, \text{data-lg}(\text{ft}_1) > 0 \land \text{data-lg}(\text{ft}_2) > 0 \rightarrow \text{data-lg}(\text{ft}_1) + \text{data-lg}(\text{ft}_2), \]
\[ \tau \rightarrow 0 \]
\[ \tau \rightarrow \text{error} \]

where:
\[ \text{ft}_0 = \text{s-type(effo)} \]
\[ \text{ft}_1 = \text{s-real(effo)} \]
\[ \text{ft}_2 = \text{s-imag(effo)} \]
\[ \text{ft}_1 = \text{s-type(efo)} \]
\[ \text{ft}_2 = \text{s-type(efo)} \]

Ref.: \[ \text{is-correct-pic } 9-48(195) \]
\[ \text{pic-length } 9-64(250) \]

(236) \[ \text{is-skipped-format(efo)} = \]
\[ \text{ft}_0 \in \{ \text{CHAR,BIT,BBT} \} \land \text{is-intg-val(w}_0) \land w_0 < 0 \lor \]
\[ \text{ft}_0 \in \{ \text{FIX,FLT,PRE,EPIC,CPLX} \} \land (\text{is-w-data-lg(efo)} \lor \text{data-lg(efo)} < 0) \]

where:
\[ \text{ft}_0 = \text{s-type(effo)} \]
\[ w_0 = \text{s-w(effo)} \]

Ref.: \[ \text{is-intg-val } 9-3(5) \]

Note: This criterion for ignoring a data format item is sufficient for input and has to be amended for output.
11.6.3 STREAM OUTPUT

List-directed and data-directed transmission essentially consists of a conversion to character string type, and of the construction of names in the form of character strings. Edit-directed transmission is separated into data format, control format, and elementary transmission. Elementary transmission is related with control formats by the instructions put-file and put-val-list. It is related with data format and list- or data-directed transmission by the instruction put-val-list only.

Metavariables

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>tr</td>
<td>transmission parameter</td>
</tr>
<tr>
<td>cp</td>
<td>operand</td>
</tr>
<tr>
<td>eda</td>
<td>data attribute</td>
</tr>
<tr>
<td>cl</td>
<td>character string</td>
</tr>
<tr>
<td>is-tr-par</td>
<td></td>
</tr>
<tr>
<td>is-op</td>
<td></td>
</tr>
<tr>
<td>is-Q</td>
<td></td>
</tr>
<tr>
<td>is-eda</td>
<td></td>
</tr>
<tr>
<td>is-Q</td>
<td></td>
</tr>
<tr>
<td>is-char-val-list</td>
<td></td>
</tr>
<tr>
<td>is-opt</td>
<td></td>
</tr>
</tbody>
</table>

Abbreviations

\[
\begin{align*}
fu-el_1 &= (\text{is-nets-file}(\text{tr}) \& \text{is-fu-file}(\text{s-file}(\text{tr}))(\text{SU}) \Rightarrow (\text{s-file}(\text{tr}))(\text{SU}), \\
    & \quad \text{is-nets-string}(\text{tr}) \Rightarrow (\text{s-string}(\text{tr}))(\text{SU}), \\
    & \quad T \Rightarrow \text{error}) \\
eda_0 &= \text{s-dae-eva}(\text{op})
\end{align*}
\]

11.6.3.1 List-directed output

(237) \[
\text{put-list}(\text{tr,op}) = \\
\text{put-val-list}(\text{tr,cl-2}); \\
\text{cl-2:sk-list-1}(\text{cl-1, <DIANE>}); \\
\text{tab}(\text{tr,eda_0,cl-1}); \\
\text{cl-1:pass-list-string}(\text{is-PBT}\text{stream-base}(\text{fu-el}_1),\text{eda_0,cl}); \\
\text{cl:conv-char-string-1}(\text{tr,op})
\]

Ref.: tab 11-122(329)
stream-base 11-84(233)
(238) \texttt{conv-char-string-1}(tr, op) =
\texttt{is-dec-pic(eda)} \lor \texttt{is-sterling-pic(eda)} \lor \texttt{is-string-type(eda)} \land
\neg \texttt{is-BIT-s-base(eda)} \rightarrow
\texttt{PASS:val}(\texttt{s-eval(op)}, \texttt{s-vr(op)})
\texttt{is-bin-pic(eda)} \lor \texttt{is-BIT-s-base(eda)} \lor \texttt{is-arithmetic(eda)} \lor
\texttt{(is-LIST} \land \texttt{is-DATA)} \land \texttt{is-type(tr)} \rightarrow
\texttt{PASS-or-val(op-1)};
\texttt{op-1:convert-1}(\texttt{aggr-scalar}(\texttt{CHAR-EDA}), \texttt{op})
\texttt{is-CHAR-DATA} \land \texttt{is-type(tr)} \rightarrow \texttt{null}
\texttt{is-ALL-DATA} \land \texttt{is-type(tr)} \rightarrow \texttt{PASS:}*
\textbf{Ref.:} \texttt{is-string-type 9-21(92)}
\texttt{val 9-6(18)}
\texttt{op-val 9-8(36)}
\texttt{convert-1 9-25(119)}
\texttt{aggr-scalar 8-23(64)}

(239) \texttt{list-string}(alt, eda, cl) =
\texttt{alt} \land \texttt{is-string-type(eda)} \land \neg \texttt{is-BIT-s-base(eda)} \rightarrow
\texttt{lg_3}(\texttt{CONC cl_3}^{*}\texttt{APOSTR})
\texttt{is-BIT-s-base(eda)} \rightarrow \texttt{APOSTR}^{*}\texttt{cl}^{*}\texttt{APOSTR,B-CHAR}
\texttt{T} \rightarrow \texttt{cl}
\texttt{where:}
\texttt{lg_3} = \texttt{str-length(eda)}
\texttt{cl_3} = (\texttt{is-APOCSTE-else} (i, cl) \rightarrow \texttt{APOSTR APOSTR},
\texttt{T} \rightarrow \texttt{elem(i, cl))})
\texttt{for:} (\texttt{is-2} \lor \texttt{is-1}) (\texttt{alt})
\textbf{Ref.:} \texttt{is-string-type 9-21(92)}
\texttt{str-length 9-41(164)}

11.6.3.2 Data-directed output

**Metavariables**

\texttt{idal} \texttt{is-id-list} \texttt{qualification part of a reference}
\texttt{intgl} \texttt{is-intg-val-list} \texttt{subscript list of a reference}
30 April 1960

ABSTRACT SYNTAX AND INTERPRETATION OF PL/I

(240) \text{rut-data}(tr,idl,intgl,op) = \\
    \text{test-rut-data}(tr,idl,intgl,eda,cl) ; \\
    cl:conv-char-string-1(tr,op)

(241) \text{test-rut-data}(tr,idl,intgl,eda,cl) = \\
    \text{is-\ast}(cl) \rightarrow \text{PASS:} ; \\
    (\text{is-\ast} \lor \text{is-char-val-list})(cl) \rightarrow \\
    \text{put-val-list}(tr,cl_1\text{-<BLANK>);} \\
    \text{is}(tr,eda,cl_1)

where:
    cl_1 = \text{data-string}(idl,intgl,eda,cl)

Ref.: \text{is-char-val 9-3(6)}
    \text{tab 11-122}(329)

(242) \text{data-string}(idl,intgl,eda,cl) = \\
    \text{is-G}(cl) \rightarrow \text{idl-string(idl)} \\
    \text{is-char-val-list}(cl) \rightarrow \\
    \text{idl-string(idl)} \text{-intgl-string(intgl)} \text{-<PO>-list-string(T,eda,cl)}

Ref.: \text{is-char-val 9-3(6)}

(243) \text{idl-string}(idl) = \\
    \text{length(idl)} \\
    \text{CONC (point_1-chl(id_1))}

where:
    id_1 = \text{elem}(i,idl) \\
    \text{point_1} = (i = 1 \rightarrow \langle>, \\
    i > 1 \rightarrow \langle POINT\rangle)

(244) chl(id) = \\
    (\langle cl \rangle) \text{is-char-val-list}(cl) \land \text{mk-id}(cl) = id

for:is-id(id)

Ref.: \text{is-char-val 9-3(6)}

Note: The function yields the character string representation of any identifier. 
The representation is unique and always defined since the function 
\text{mk-id}(cl) is restricted as follows: 
\text{is-id} \land \text{mk-id}(cl) \land (cl-1 \neq cl-2 \Rightarrow \text{mk-id}(cl-1) \neq \text{mk-id}(cl-2)), 
for all cl, cl-1, cl-2 which satisfy the predicate \text{is-char-val-list}.
(245) intgl-string(intgl) =
    is-<>(intgl) -> <>
    length(intgl)
    T <- ( CONC (del_i-signed-int3_i) ) <RIGHT-PAD>

Where:
        del_i = (i = 1 <-<LEFT-PAD>,
                  i > 1 <-<COMMA>)
        signed-int3_i = (v_1 = 0 <-<C-Chop>,
                          v_1 < 0 <-<MINUS> digit-list(abs(v_1)),
                          v_1 > 0 <- digit-list(v_1))
        v_1 = cloc(i,intgl)

(246) digit-list(i) =
    i = 0 -> <>
    i > 0 -> digit-list•trunc(i / 10)•<single-num-char•modulo(i,10)>

for: is-intg-val(i) & i > 0

Ref.: single-num-char 9-32(129)
      is-intg-val 9-3(5)

11.6.3.3 Elementary and edit-directed output

Metavariables

vl is-bit-val-list v is-char-val-list string destined for output

11.6.3.3.1 Elementary output

This section defines data transmission to a string, and transmission of data or
control information to and from a data set by a file.

Metavariables

s is-gen v is-ps-gen scalar string-type generation
    or pseudo generation of the string option or a
    part of it

col,lc,i is-intg-val, > 1 column, lower index, index
Abbreviations

\( s_x = \text{sel}(\text{fu}-x_1) \)
\( s_x = \text{set}(\text{fu}-x_1) \)
\( ds_{-sel_1} = (\text{ds}_{-sel}(\text{title}(\frac{x_1}{x_1}), s_{-ea}(\frac{x_1}{x_1}), s_{-ds}_{-sh}(\frac{x_1}{x_1}))) \cdot s_{-ds}_{-dir} \)
\( ds_1 = s_{-ds} \cdot ds_{-sel_1}(\frac{L_1}{L_1}) \)
\( \text{mp}(\frac{x_1}{x_1}) \)

The abbreviations are explained at the beginning of section 11.3.

(247) \text{rut}-\text{val}-\text{list}(tr, vl) =

\( \text{is-}x \mapsto (vl) \rightarrow \text{null} \)
\( \text{is-}\text{e-string}(tr) \rightarrow \)
\( \text{rut}-\text{val}-\text{list}(tr, \text{tail}(vl)); \)
\( \text{rut}-\text{list}(tr, \text{head}(vl)) \)
\( s_{-col}(\text{fu}-x_1) \leq s_{-lsz}(\text{fu}-x_1) \rightarrow \)
\( \text{rut}-\text{val}-\text{list}(tr, \text{tail}(vl)); \)
\( \text{rut}-\text{file}(tr, s_{-col}(\text{fu}-x_1) + 1, \text{head}(vl)) \)
\( t \rightarrow \)
\( \text{rut}-\text{val}-\text{list}(tr, vl); \)
\( \text{rut}-\text{skip}(tr, 1, 2) \).
(248) `cut-string(tr,v) =

  is-G(v)  →  call-cond(ERROR)

  T  →

  assign(g,v0(<s-eva:s-eva(g0)>,<s-ev:rep(s-eva(g0),v0)>));
  und-fu-elem(<s-string(tr),s-hi,hi1 + 1>)

where:

  V0 =
  adjust-substr-1(s-da•s-eve(g0),val(s-eva(g0),s-FF(g0)(S)),elem(2,arg0),
  elem(3,arg0),<f>)
  arg0 = s-arg-list•substr-gen(s-g(fu-el1),hi1 + 1,1,s-chif(CS))
  g0 = elem(1,arg0)
  hi1 = s-hi(fu-el1)

Ref.:  call-cond 10-18(54)
  assign 8-9(22)
  ret 8-6(19)
  und-fu-elem 11-50(135)
  val 8-6(18)

(249) substr-gen(g,lc,lg,chif) =

  is-gen(g)  →  mk-substr-gen(g,lc,lg)
  s-id(g) = mk-id(STRING)  →  mk-substr-gen(g1,lo,lg)
  s-id(g) = mk-id(SUBSTR) &
  (is-g•elem(3,arg1) v is-0(lg) v lo + lg - 1 <= elem(3,arg1))  →
  mk-substr-gen(g1,elem(2,arg1) + lo - 1,lg1)
  s-id(g) = mk-id(UNSPEC)  →
  mk-substr-gen(p(g1,<s-eva:sub-str-eva(BIT,unspec-length•size-1•s-pp(g1)>)>,lc,lg)
  s-id(g) = mk-id(ONSOURCE)  →  substr-gen(s-onsource(chif),lo,lg,0)
  s-id(g) = mk-id(ONCHAR) & lo = 1 & (is-0(lg) v lg = 1)  →
  substr-gen(s-onsource(chif),s-onchar(chif),1,0)
  s-id(g) = mk-id(ONIDENT) & is-F•s-inputs-onident(chif)  →
  mk-substr-gen(s-gen•s-onident(chif),lo,lg)

  T  →  error

where:

  arg1 = s-arg-list(g)
  g1 = elem(1,arg1)
  lg1 = (is-g•elem(3,arg1)  →  lg,
         is-0(lg)  →  elem(3,arg1) - lo + 1,
         T  →  lg)

  for:is-chif(chif)

cont'd
Ref.: is-gen 3-14(30)
str-eva 7-14(45)
unspec-length 12-42(71)
size-7 3-15(38)
is-chif 3-22(85)

Note: The generation $g$ originates from a declared reference $ref$ for which
$\text{is-base}(ref, AT)$ holds. The function yields an artificial substring
pseudo-generation constructed from $g$ if $lo$ is interpreted as the lower
index and $lg$ as the length of the substring field.

(250) $\text{mk-substr-gen}(g, lo, lg) =$
$$
\mu_0(<\text{id:mk-id(SUBSTR)}, <\text{arg-list:<g,lo,lg>>})
$$

(251) $\text{adjust-substr-1}(eda, vl-0, i, j, vl) =$
$$
(\text{is-VAR-varying}(eda) \land i \leq lg_0 + 1 \lor \text{is-VAR-varying}(eda) \land i \leq lg_0) \land
i \leq j_0 - 1 \leq \text{str-length}(eda) \rightarrow
\text{adjust-substr-2}(eda, vl-0, i, j_0, vl)
$$
$T \rightarrow \Omega$

where:
$$j_0 = (is-0(j) \rightarrow \text{str-length}(eda) - i + 1,$$
$T \rightarrow j)$
$$lg_0 = \text{length}(vl-0)$

Ref.: str-length 9-41(164)

Note: The function roughly yields the substring of length $j$ of $vl$, starting with
the $i$-th element of $vl$. Constraints are given by $eda$ and $vl-0$. If the
substring cannot be appropriately extracted, the indication $\Omega$ is yielded.
ABSTRACT SYNTAX AND INTERPRETATION OF FL/I

(252) \texttt{adjust-substr-2(eda,vl,f,i,j,vl) =}
\begin{align*}
&\text{is-char-pic}(eda) \& \text{lg}_1 = j \& \\
&\text{is-char-pic-match}(\mu_0(<\text{s-field: } \sum_{k=1}^{j} \text{elem}(k+i+1, \text{s-field}(eda))>, vl) \vee \\
&\text{is-\&s-varying}(eda) \& \text{lg}_1 \leq j \& \\
&\text{vl} \Rightarrow \text{vl} \Rightarrow (\text{fill-char-as-base}(eda))^{\text{vl}} \\
&\text{is-\&s-varying}(eda) \& \text{lg}_1 \leq j \Rightarrow \text{vl} \Rightarrow \text{vl} \Rightarrow \text{vl} \\
&\tau \Rightarrow 0
\end{align*}

where:
\begin{align*}
\text{lg}_1 &= \text{length}(vl) \\
\text{lg}_2 &= j - \text{lg}_1 \\
\text{vl}_1 &= \text{LIST} \text{elem}(k,vl-0) \\
\text{lg}_3 &= \text{length}(vl-0) \\
\text{vl}_2 &= \text{LIST} \text{elem}(k,vl-0) \\
\text{vl}_3 &= \text{LIST} \text{elem}(k,vl-0)
\end{align*}

Ref.: is-char-pic-match 9-59(234) \\
fill-char 9-9(33)

Note: If the function yields a string, it is a string of length \text{lg}_0.

(253) \texttt{put-file(tr,col,el) =}
\begin{align*}
&\text{test-inf(tr,inf,el)}; \\
&\text{inf:stream-transmission(tr,el)}; \\
&\text{upd-fu-elem(s-file(tr),s-col,col)}
\end{align*}

\texttt{fcr:is-str-el(el)}

Ref.: upd-fu-elem 11-50(135) \\
is-str-el 11-3(3)
30 April 1969

ABSTRACT SYNTAX AND INTERPRETATION OF PL/I

(254) \[
\text{stream-transmission}(\text{tr}, \text{arg}) = \\
\text{is-0*s-st}(\text{fu-e1}) \rightarrow \text{error} \\
\text{is-Pu*s-st}(\text{tr}) \rightarrow \\
\quad \text{PASS:} s\text{-inf\*write}(mp_1, ds_1, \text{arg}) \\
\quad s = s_0 (\sigma \xi : ds \text{-sel}_1 : \mu_0 (\sigma \xi : ds \text{-ds*write}(mp_1, ds_1, \text{arg}))) \\
\quad s = \text{id}_1 \\
\text{is-intg-val}(\text{arg}) \lor \text{is-char-val}(rd_1) \lor \text{is-scan-class}(s\text{-incr}, arg, rd_1) \rightarrow \\
\quad \text{PASS:} rd_1 \\
\quad s = s_0 (\sigma \xi : ds \text{-sel}_1 : \mu_0 (\sigma \xi : ds \text{-ds*read}(mp_1, ds_1))) \\
\quad s = \text{id}_1 \\
\text{is-scan-class}(s\text{-stop-incr}, arg, rd_1) \rightarrow \\
\quad \text{PASS:} \mu_0 (\sigma \xi : s\text{-stop-inf:} rd_1) \\
\quad s = s_0 (\sigma \xi : ds \text{-sel}_1 : \mu_0 (\sigma \xi : ds \text{-ds*read}(mp_1, ds_1))) \\
\quad s = \text{id}_1 \\
\text{is-scan-class}(s\text{-stop}, arg, rd_1) \rightarrow \\
\quad \text{PASS:} \mu_0 (\sigma \xi : s\text{-stop-inf:} rd_1) \\
\quad s = \text{id}_1 \\
\]

where:
\[
FP_1 = (\text{is-\text{IN}f-s-\text{ds-sel}_1} (\xi) \rightarrow \mu (\xi): \langle \text{test-sel}(f_1, \text{id}) \rangle, \\
T \rightarrow \text{id}) \\
rd_1 = s\text{-inf\*read}(mp_1, ds_1)
\]

Ref.: write 11-15(47) 
\text{is-intg-val} 9-1(5) 
\text{is-char-val} 9-3(6) 
\text{is-scan-class} 11-11(301) 
read 11-12(42) 
tst-sel 11-79(220)

Note: The last alternatives are taken in the case of stream input only (cf. 11.6.4).

The transmission error flag (if any) from the data set is copied into the file directory entry (cf. test-transmit).

(255) \[
\text{test-inf}(\text{tr}, \text{inf}, \text{arg}) = \\
\text{is-\text{IND}(inf) \rightarrow} \\
\quad \text{test-inf}(\text{tr}, \text{inf-1}, \text{arg}); \\
\quad \text{inf-1:retry-stream-transmission(\text{tr}, \text{cond}, \text{arg});} \\
\quad \text{cond:ds-lab(s-file(\text{tr}), EOV-BOV)} \\
\quad \text{is-Pu*s-st(\text{tr}) \rightarrow null} \\
\quad \text{is-GF*s-st(\text{tr}) \rightarrow test-get-inf(\text{tr}, \text{inf})} \\
\quad \text{for:}(\text{is-str-el} \lor \text{is-0} \lor \text{is-\text{IND} \lor \text{is-stop-inf})} (\text{inf})
\]

cont'd
retry-stream-transmission (tr, cond, arg) =

  is-f-cond (cond) -- call-io-cond-1 (cond)
  is-PUTs-st (tr) v is-GETs-st (tr) & is-intg-val v is-ERROR-ENDF*s-end (arg) &
  is-ENDFs-st (fu-el1) --

  int-next-st:
  revert-onfile;
  call-io-cond-1 (μ0 (<s-f:fl>, <s-cond:ENDF>))
  is-ENDFs-st (fu-el1) -- PASS; μ0 (<s-stop-inf:ENDM-SCAN>)
  T -- stream-transmission (tr, arg)

for: (is-Q v is-f-cond) (cond)

Note: The end of file status has been eventually reached by instruction
ds-switch.

11.6.3.3.2 Control formats

Metavariables:

w

is-intg-val

width of a control format item

endp

is-Q v is-ENDP

indication for not raised
or raised endpage on-condition.

Abbreviation

lineₜ = op-values-line (fu-elₜ)

current line number.
ABSTRACT SYNTAX AND INTERPRETATION OF PL/I

(257) \[ \text{put-space}(tr, w) = \]
\[ \text{put-val-list}(tr, \text{LIST} \sum w \text{fill}_i) \]

where:
\[ \text{fill}_i = ((\text{is-BIT} \lor \text{is-BST}) \land \text{stream-base}(fu-el_i) \rightarrow \text{0-BIT}, \]
\[ \text{T} \rightarrow \text{ELINK}) \]

Ref.: stream-base 11-84(233)

(258) \[ \text{put-column}(tr, w) = \]
\[ - (1 \leq w \leq s-\text{lsz}(fu-el_i)) \rightarrow \text{put-column}(tr, 1) \]
\[ w \geq s-\text{col}(fu-el_i) \rightarrow \text{put-space}(tr, w - s-\text{col}(fu-el_i)) \]
\[ w < s-\text{col}(fu-el_i) \rightarrow \]
\[ \text{put-column}(tr, w); \text{put-skip}(tr, 1, 0) \]

(259) \[ \text{put-skip}(tr, w, \text{endp}) = \]
\[ \text{is-\text{ENDP}(endp)} \lor w = 0 \rightarrow \text{null} \]
\[ \text{T} \rightarrow \]
\[ \text{put-skip}(tr, w - 1, \text{endp}-1); \text{endp}-1: \text{start-line}(tr) \]

(260) \[ \text{start-line}(tr) = \]
\[ \text{is-BIT}\land \text{stream-base}(fu-el_i) \land s-\text{psz}(fu-el_i) = \text{line}_i \rightarrow \]
\[ \text{pass}(\text{endp}); \]
\[ \text{call-ic-cond-1}(p_0\langle <s-\text{line}(fu-el_i)>, <s-\text{cond}(\text{endp})>>); \]
\[ \text{upd-fu-elem}(s-\text{file}(tr), s-\text{col}, 1), \]
\[ \text{and-fu-elem}(s-\text{file}(tr), s-\text{line}, \text{bintg-op}); \]
\[ \text{bintg-op: add-one}(s-\text{line}(fu-el_i)) \]
\[ \text{T} \rightarrow \]
\[ \text{put-file}(tr, 1, \text{LABEL}); \]
\[ \text{upd-fu-elem}(s-\text{file}(tr), s-\text{line}, \text{bintg-op}); \]
\[ \text{bintg-op: add-one}(s-\text{line}(fu-el_i)) \]

Ref.: stream-base 11-84(233)
\[ \text{call-ic-cond-1 10-24(72)} \]
\[ \text{upd-fu-elem 11-50(135)} \]
\[ \text{bintg-op 9-10(40)} \]
\[ \text{add-one 11-74(210)} \]

11. INPUT AND OUTPUT 97
(261) \texttt{put-line}(tr,w,endr) =
\begin{align*}
\text{in-INDP}(endr) \land w = \text{line}_1 \land s-col(fu-el_1) = 1 & \rightarrow \text{null} \\
s-tsz(fu-el_1) < \text{line}_1 \land w < \text{line}_1 & \rightarrow \text{put-page}(tr) \\
T & \rightarrow \\
& \text{put-line}(tr,w,endp-1); \\
& \text{endp-1:start-line}(tr)
\end{align*}

(262) \texttt{put-page}(tr) =
\begin{align*}
& \text{put-file}(tr,1,\text{FINL}); \\
& \text{end-fu-elec}(tr,s-line,hintg-op(1))
\end{align*}

\textbf{Ref.:} \texttt{end-fu-elec 11-50(135)}
\texttt{hintg-op 9-10(40)}

11.6.3.3 Data formats

\textbf{Metavariables:}

\begin{align*}
\text{edfc} & \quad \text{evaluated data format item} \\
\text{sgn} & \quad \text{sgn} \in \{<\text{PLUS}>, <\text{MINUS}>\} \quad \text{sign of data field} \\
\text{dl} & \quad \text{is-digit-list} \quad \text{integer in string form} \\
\text{d} & \quad \text{is-intg-val} \quad \text{number of digits after decimal point} \\
\text{s} & \quad \text{is-intg-val} \quad \text{number of significant digits or scaling factor} \\
\text{i} & \quad \text{is-intg-val} \quad \text{index} \\
\text{mode} & \quad \text{is-CPLX} \lor \text{is-REAL} \quad \text{mode}
\end{align*}

\textbf{Abbreviations:}

\begin{align*}
\text{ft}_0 & = \text{s-type (edfc)} \\
\text{w}_0 & = \text{s-w (edfo)} \\
\text{lg}_0 & = \text{length (dl)}
\end{align*}
(263) put-edit(tr,op,edfo) =
    is-skipped-format(edfo) xor is-int-g-val data-lg(edfo) & data-lg(edfo) = 0 ->
    PASS:
    is-CPLX(ft0) ->
    put-edit-1(tr,op,s-imag(edfo),CPLX);
    put-edit-1(tr,op,s-real(edfo),REAL);
    T -> put-edit-1(tr,op,edfo,REAL)

Ref.:  is-skipped-format 11-86(236)
       is-int-g-val 9-3 (5)
       data-lg 11-85(235)

(264) put-edit-1(tr,op,edfo,mode) =
    put-val-list(tr,vl);
    vl:verify-width(vl-1,edfo);
    vl-1:pass-edit-op(op-1,edfo);
    op-1:convert-1(agr-scalar-edit-eda(edfo,mode),op)

Ref.:  convert-1 9-29(119)
       agr-scalar 8-23(64)

(265) edit-eda(edfo,mode) =
    ft0 € [CHAR,BIT,BBIT]  -> μ0 (<s-base:base1>,<s-length:lg1>,<s-varying:*>)
    ft0 € [FIX,FLT]  -> μ0 (<s-mode:mode>,<s-base:DEC>,<s-prec:*>)
    ft0 € [PIC,BPIC]  -> μ(<s-pic(edfo);<s-mode:mode>)

where:
    base1 = (is-BIT(ft0)  -> BIT,
             T -> CHAR)
    lg1 = (is-0(w0)  -> *,
             T -> w0)
    mode1 = (is-CPLX(mode)  -> CPLX,
             T -> 0)
(266)  \( \text{edit-c}_1(\text{op}, \text{edfo}) = \)

\[
\begin{align*}
\text{\textit{ft}}_0 & \in \{ \text{CHAR, BIT} \} \rightarrow \text{op-val}(\text{op}) \\
\text{is-BIT(} \text{ft}_0) & \rightarrow \text{bit-char-conv}(\text{op-val}(\text{op})) \\
\text{is-FLC(} \text{ft}_0) & \& \text{is-his-pic-s-pic}(\text{edfo}) \\
& \rightarrow \\
\text{bit-char-conv}(\text{edit-part}(\text{ft}_0, \text{mode}_0, \text{val}(\text{op-val}(\text{op}), \text{s-bk}(\text{op})))) \\
\text{ft}_0 & \in \{ \text{PIC, BFLC} \} \rightarrow \text{edit-part}(\text{ft}_0, \text{mode}_0, \text{val}(\text{op-val}(\text{op}), \text{s-bk}(\text{op})))) \\
\text{is-FLN}(\text{ft}_0) & \& \text{d}_1 \geq 0 \rightarrow \\
\text{mk-f-field}(\text{sign}_0, \text{d}_1, \text{digit-list}\text{-}trunc(1/2 + \text{abs}(\text{v}_0) \cdot 10 + \text{p}_d) + 1, \text{d}_1, \text{abs}(\text{p}_d)) \\
\text{is-FTT}(\text{ft}_0) & \& \text{s}_1 > 0 \& \text{s}_1 \geq \text{d}_1 \geq 0 \rightarrow \\
\text{mk-m-field}(\text{sign}_0, \text{d}_1, \text{digit-list}\text{-}trunc(1/2 + \text{abs}(\text{v}_0) \cdot 10 + (\text{s}_1 - \text{lg}_1)), \text{d}_1, \text{abs}(\text{lg}_1 - (\text{s}_1 - \text{d}_1))) \\
& \rightarrow \text{error}
\end{align*}
\]

\text{where:}

\[
\begin{align*}
\text{mode}_0 & = \text{s-mode} \& \text{s-da} \& \text{s-ev}(\text{op}) \\
\text{v}_0 & = \text{edit-part}(\text{ft}_0, \text{mode}_0, \text{op-val}(\text{op})) \\
\text{sign}_0 & = (\text{v}_0 \geq 0 \rightarrow \text{<PLUS>}, \text{T} \rightarrow \text{<MINUS>}) \\
\text{d}_1 & = \text{s-d}(\text{edfo}) \\
\text{v}_1 & = (\text{d}_1 = 0 \rightarrow \text{trunc}(\text{v}_0), \text{T} \rightarrow \text{v}_0) \\
\text{p}_d & = (\text{i} \times 2^{s-p-s}(\text{edfo}) \rightarrow \text{d}_1, \text{T} \rightarrow \text{s-p-s}(\text{edfo}) + \text{d}_1) \\
\text{s}_1 & = (\text{i} \times 2^{s-p-s}(\text{edfo}) \rightarrow \text{d}_1 + 1, \text{T} \rightarrow \text{s-p-s}(\text{edfo})) \\
\text{lg}_1 & = (\text{v}_0 = 0 \rightarrow \text{s}_1, \text{T} \rightarrow \text{lg}(10 + (\text{s}_1 - 1) \leq \text{abs}(\text{v}_0) < 10 + \text{i})) \\
\text{sign}_1 & = (\text{lg}_1 = (\text{s}_1 - \text{d}_1) \geq 0 \rightarrow \text{<PLUS>}, \text{T} \rightarrow \text{<MINUS>})
\end{align*}
\]

\text{Ref.:}  \text{op-val 9-9}(36) \\
\text{bit-char-conv 9-35}(138) \\
\text{val 9-6}(18)

\text{Note:}  \text{The last alternative represents the erroneous case of having an improper number of fractional or significant digits. This alternative applies only to fixed-point and floating-point format items, and does not depend on } w_0.
ABSTRACT SYNTAX AND INTERPRETATION OF PL/I

(267) \text{edit-part}(ft, node, v) =
    \text{-is-CHLi}(node) \rightarrow v
    ft \in \{IX, FLT\} \rightarrow \text{imag}(v)
    ft \in \{PIC, BPICT\} \rightarrow \sum_{i=1}^{\lfloor \frac{\log_2 n}{2} \rfloor} \text{elem}(i + \lfloor \frac{\log_2 n}{2} \rfloor, v)

where:
    \lfloor \frac{\log_2 n}{2} \rfloor = \text{length}(v) / 2

for: \text{is-value}(v)

Ref.: is-value 9-3(2)

(268) \text{mk-f-field}(sgn, d, dl) =
    \text{d} = 0 & \log_0 = 0 \rightarrow \langle 0-CHAR \rangle
    \text{d} = 0 \rightarrow \text{sgn-dl}
    \text{d} \geq \log_0 \rightarrow \text{sgn} \langle 0-CHAR \rangle \langle POINT \rangle \langle \sum_{i=1}^{\lfloor \frac{\log_2 n}{2} \rfloor} \text{elem}(i, d, dl) \rangle
    \text{d} < \log_0 \rightarrow \text{sgn} \langle \sum_{i=1}^{\lfloor \frac{\log_2 n}{2} \rfloor} \text{elem}(i, d, dl) \rangle \langle POINT \rangle \langle \sum_{i=1}^{\lfloor \frac{\log_2 n}{2} \rfloor} \text{elem}(i + \lfloor \frac{\log_2 n}{2} \rfloor, d, dl) \rangle

where:
    \lfloor \frac{\log_2 n}{2} \rfloor = \log_2 - \log_0

(269) \text{mk-n-field}(sgn, d, s, dl) =
    \text{d} = 0 & \log_0 = 0 \rightarrow \langle \sum_{i=1}^{s} 0-CHAR \rangle
    \text{T} \rightarrow \text{mk-f-field}(sgn, d, dl)

(270) \text{mk-x-field}(sgn, dl) =
    \text{EXP-SIZE} < \log_0 \rightarrow \text{error}
    \text{T} \rightarrow \langle E-CHAR \rangle \langle sgn \langle \sum_{i=1}^{\lfloor \frac{\log_2 n}{2} \rfloor} 0-CHAR \rangle \rangle \langle dl \rangle

where:
    \lfloor \frac{\log_2 n}{2} \rfloor = \text{EXP-SIZE} - \log_0
(271) \[\text{verify-width}(vl, edfo) = \]
\[
\text{ft} \circ \{\text{FIX, FLT}\} \rightarrow \text{PASS:vl}
\]
\[
w_0 < \text{length}(vl) \rightarrow
\]
\[
\text{PASS}(cl_1);
\]
\[
\text{call-cond}(\text{SIZE})
\]
\[
T \rightarrow \text{PASS:cl}_3
\]
where:
\[
cl_1 = \text{LIST} \ \text{elem}(i - \text{lg}_2 vl)
\]
\[
\text{lg}_2 = w_0 - \text{length}(vl)
\]
\[
cl_2 = \text{LIST} \ \text{BLANK}^{\text{lg}_2} vl
\]

Ref.: \text{call-cond} 10-16(54)
11.6.4 STEPM INPUN

List- and data-directed input occurs in two steps: A scanning step and a conversion step. Depending on the first step, the second step may be skipped. The scanning step in fact is composed of subsequent scanings, each with different arguments. Sections 11.6.4.1 and 11.6.4.2 define the succession of different scanings and the conversion step for list- and data-directed transmission. The definition of the scanning mechanism is given as part of elementary transmission which is defined as a sub-section of edit-directed transmission (11.6.4.3).

Also edit-directed transmission is composed of the above two steps, but the scanning step is very simple. There is only one scanning, and scanning does not depend on the input character or bit just read.

Metavariables

<table>
<thead>
<tr>
<th>tr</th>
<th>is-tr-par</th>
<th>transmission parameter</th>
</tr>
</thead>
<tbody>
<tr>
<td>cl</td>
<td>is-char-val-list v (2) (is-char-val-list (v) x cl = x&lt;ENDM-SCAN&gt;)</td>
<td>input string; string of characters optionally followed by the special indication ENDM-SCAN</td>
</tr>
<tr>
<td>vl</td>
<td>is-char-val-list v is-bit-val-list</td>
<td>input string</td>
</tr>
<tr>
<td>i,j,k</td>
<td>is-int-val</td>
<td>indices</td>
</tr>
</tbody>
</table>

Abbreviations

\[
\text{fu-cl} = (\text{is-nos-file}(\text{tr}) \& \text{is-fu-file}(\text{s-file}(\text{tr}))(\text{EN}) \rightarrow (\text{s-file}(\text{tr}))(\text{EN}), \\
\text{is-nos-string}(\text{tr}) \rightarrow (\text{s-string}(\text{tr}))(\text{EN}), \\
T \rightarrow \text{error})
\]

11.6.4.1 List-directed input

(272) \text{get-list}(\text{tr}) =

\[
\text{test-list-1}(\text{tr},\text{cl}-1); \\
\text{cl}-1: \text{get-val-list}(\text{tr},\mu_0 (\text{<s-end:ERROR-LIST>}),<\text{s-stor-incr:COMMA}>), <\text{s-incr:BLANK}>))
\]

(273) \text{test-list-1}(\text{tr},\text{cl}-1) =

\[
\text{in-COMMA} * \text{last}(\text{cl}-1) \rightarrow \text{PASS}; \\
T \rightarrow \\
\text{sk-tr-op}(\text{tr},\text{cl}); \\
\text{cl} * \text{test-list-2}(\text{tr},\text{cl}-2); \\
\text{cl}-2: \text{get-val-list}(\text{tr},\mu_0 (\text{<s-step:>}), <\text{s-stor-incr:AFOSIS,COMMA,BLANK}>));
\]

11. INPUT AND OUTPUT 103
(274) `update-ic-string(tr) =

  is-nws-file(tr)   => null
  is-nws-string(tr) => update-elem(s-string(tr),s-lo,s-hi(fu-cl) + 1)

Ref.: `update-elem 11-50(135)

Note: The lower index of the data field will be used by conversion on-condition calls (cf. `call-get-conv).

(275) `test-list-2(tr,cl-2) =

  (is-comm v is-endm-scan) * last(cl-2)   => pass-rp-field(tr,cl-2)
  is-blank * last(cl-2)   =>

    pass-rp-field(tr,cl-2); get-val-list(tr,µ₀(<s-stop-incr:{comma}>,<s-incr:{blank}>))

Ref.: `conc 9-52(207)

(276) `test-list-3(tr,cl-3) =

  is-endm-scan * last(cl-3)   => pass-rp-field(tr,cl-3)

T =>

  `test-list-2(tr,cl-2); cl-3:conc(cl-2,cl); cl:conc-cl(tr,cl,µ₀(<s-stop:>{},<s-stop-incr:{apostrophe},comma,blank}>))

Ref.: `conc 9-52(207)

(277) `rp-field(tr,cl) =

  is-lists-type(tr) v is-endm-scan * last(cl)   => &{cl;elem;length(cl)}

T => cl

(278) `ek-rp-tr(tr,cl) =

  is-compl-correct(cl,is-io-prop-rp) =>

    is-data-op(elem(1,parsing(cl,is-io-prop-rp)))

T =>

  `ek-rp-tr(tr,corr-cl); corr-cl:call-get-conv(tr,cl,wrong-pos(cl,is-io-prop-rp))

...
30 April 1969

ABSTRACT SYNTAX AND INTERPRETATION OF PL/I

Ref.: is-compl-correct 11-71(199)
      parsing 11-71(197)
      wrong-pos 11-71(201)

(279) call-get-conv(tr,c1,i) =
        is-ms-file(tr) ⇒ call-conv-cond(c1,i)
        is-as-string(tr) ⇒
        call-conv-cond(substr-gen(c-g(fu-el1),c-lo(fu-el1),length(c1),c-chif(CS)),i)

Ref.: call-conv-cond 10-24(70)
      substr-gen 11-92(249)

Note: In the case of a string option, no dummy will be created for the onsource field available in the on-unit.

(280) is-ic-prop-tp =
      (elem(1):is-ic-prop-data>,
       elem(2):is-BLANK v is-COMMA v is-SIMIC v is-<>)

(281) is-ic-prop-data(dt) =
      (is-io-data & is-e-exponent)(dt)

Ref.: is-io-data 11-67(169)
      is-e-exponent 11-72(205)

(282) io-data-op (dt) =
      is-io-arithm-const (dt) ⇒ mk-get-op (num-sda (dt),num-val (dt))
      is-io-cplx-expr (dt) ⇒
      eval-cplx (op-1,op-2);
      op-2:io-data-op (mu(elem(3,dt);<elem(3):<>));
      op-1:io-data-op (mu(dt;<elem(3):<>))
      is-io-char-string(dt) ⇒ mk-get-op (CHAR,v1)
      is-io-bit-string(dt) ⇒ mk-get-op (BIT,v2)

where:
      v1 = LIST
           \( \sum_{i=1}^{lg_0} (is-ic-string-aposstr (v_0) ⇒ \text{APOSTR.} \)
           T \rightarrow v_0 \)
      lg_0 = length(elem(2,dt))
      v_0 = elem(1,elem(2,dt))
      v_2 = LIST
            \( \sum_{i=1}^{lg_0} \text{single-char-bit}(v_0) \)

for:is-io-data(dt)

cont'd
Ref.: is-ic-arithm-const 11-67(171)

num-val 11-72(206)
is-ic-cplx-expr 11-67(172)
eval-cplx 12-46(84)
is-ic-char-string 11-67(174)
is-ic-bit-string 11-68(175)
is-ic-string-apost 11-69(169)
single-char-bit 9-34(135)
is-ic-data 11-67(169)

(203) \( \text{mk-get-op} (eda, v) = \)

\( (\text{is-BIT} \lor \text{is-BEIT}) (eda) \rightarrow \text{mk-get-op} (\text{bit-eda} \cdot \text{length}(v), v) \)

\( \text{is-CHAR}(eda) \rightarrow \text{mk-get-op} (\text{char-eda} \cdot \text{length}(v), v) \)

\( \text{is-edata}(eda) \rightarrow \)

\( \text{mk-op} (\text{eva}_0, v) ;\)

\( \text{vr} = \text{test-rep} (\text{eva}_0, v) \)

where:
\( \text{eva}_0 = \text{aggr-scalar}(eda) \)

for: (is-num-val \lor is-char-val-list \lor is-bit-val-list) (v)

Ref.: bit-edata 9-12(59)
char-edata 9-12(58)
\( \text{mk-ct} 9-9(35) \)
\( \text{test-rep} 9-7(27) \)
\( \text{aggr-scalar} 8-23(64) \)
\( \text{is-num-val} 9-3(3) \)
\( \text{is-char-val} 9-3(6) \)
\( \text{is-bit-val} 9-4(11) \)

(204) \( \text{num-edata}(dt) = \)

\( \text{is-I-CHAR}\cdot\text{elem}(3, dt) \rightarrow \mu(\text{real-edata}\cdot\text{elem}(2, dt) ;\langle s\cdot\text{mode}\cdot\text{CPLX} \rangle) \)

\( T \rightarrow \text{real-edata}\cdot\text{elem}(2, dt) \)

for: is-ic-arithm-const (dt)

Ref.: is-ic-arithm-const 11-67(171)

(205) \( \text{real-edata}(dt) = \)

\( \mu(\text{eda}_1 ;\langle s\cdot\text{prec}\cdot\text{min}(\text{max-prec}(\text{eda}_1), s\cdot\text{prec}(\text{eda}_1)) \rangle) \)

where:
\( \text{eda}_1 = \text{real-app-edata}(dt) \)

for: is-ic-real-const (dt)

Ref.: max-prec 9-12(61)
is-ic-real-const 11-68(178)
(286) \( \text{real-app-eda}(dt) = \)

\( (\text{is-io-decimal-float} \lor \text{is-io-binary-float})(dt) \rightarrow \)

\( \mu(\delta(\text{real-app-eda-elem}(1, dt); <s-scale: f}; <s-base: b_1 >, <s-scale: FLT>)) \)

\( \text{is-io-binary-fixed}(dt) \rightarrow \mu(\text{real-app-eda-elem}(1, dt); <s-base: BIN>) \)

\( T \rightarrow \)

\( \mu_e(<s-mode: \text{REAL}>, <s-base: DEC>, <s-scale: FLT>, \)

\( <s-rec:length=\text{unnest-cl-elem}(1, dt) + \log_3, <s-scale-f: \log_3>) \)

where:

\( b_1 = (\text{is-io-decimal-float}(t) \rightarrow \text{DEC}, \)
\( T \rightarrow \text{BIN}) \)

\( \log_3 = (\text{is-ll-elem}(3, dt) \rightarrow 0, \)
\( T \rightarrow \text{length=\text{unnest-cl-elem}(3, dt)}) \)

for: \( \text{is-io-real-const}(dt) \)

Ref.: \( \text{is-io-decimal-float} 11-68(160) \)
\( \text{is-io-binary-float} 11-69(187) \)
\( \text{is-io-binary-fixed} 11-69(184) \)
\( \text{unnest-cl} 11-71(198) \)
\( \text{is-io-real-const} 11-69(178) \)

11.6.4.2 Data-directed input

Metavariables

\( \text{ref-list} \quad \text{is-ref-list} \quad \text{data list} \)

\( \text{io-ref} \quad \text{is-io-basic-ref} \quad \text{name as appearing in the input string} \)

(287) \( \text{get-data-lp}(tr, \text{ref-list}) = \)

\( \text{test-data-l}(tr, cl-1, \text{ref-list}); \)
\( \text{cl-1: get-real-list}(tr, \mu_0(<s-end:ERROR-ENDF>, <s-stop-incr:{COMMA, SEMIC}>), <s-incr:{BLANK}>) \)
(288) \( \text{test-data-1}(\text{tr,cl-1,ref-list}) = \)
    \( \text{is-COMMA} \cdot \text{last}(\text{cl-1}) \rightarrow \text{get-data-lp(\text{tr,ref-list})} \)
    \( \text{is-SEMIC} \cdot \text{last}(\text{cl-1}) \rightarrow \text{null} \)
    \( T \rightarrow \)
    \( \text{test-name}(\text{tr,cl,ref-list}) ; \)
    \( \text{cl-1: \text{get-data-2(\text{tr,cl-2})}} ; \)
    \( \text{cl-2: get-val-list(\text{tr,} \mu_{0} (\text{<s-stop: (SEMIC), <s-stop-incr: (APOST, EQ)}>))} \)

(289) \( \text{test-data-2}(\text{tr,cl-2}) = \)
    \( (\text{is-SEMIC} \lor \text{is-IO} \lor \text{is-INDM-SCAN}) \cdot \text{last}(\text{cl-2}) \rightarrow \text{PASS:cl-2} \)
    \( T \rightarrow \)
    \( \text{test-data-3(\text{tr,cl-3})} ; \)
    \( \text{cl-3: \text{conc}(\text{cl-2},\text{cl})} ; \)
    \( \text{cl-2: get-val-list(\text{tr,} \mu_{0} (\text{<s-stop: (SEMIC), <s-stop-incr: (APOST, EQ)}>))} \)

Ref.: \( \text{conc 9-52(207)} \)

(290) \( \text{test-data-3}(\text{tr,cl-3}) = \)
    \( \text{is-INDM-SCAN} \cdot \text{last}(\text{cl-3}) \rightarrow \text{PASS:cl-3} \)
    \( T \rightarrow \)
    \( \text{test-data-2(\text{tr,cl-2})} ; \)
    \( \text{cl-2: \text{conc}(\text{cl-3},\text{cl})} ; \)
    \( \text{cl-3: get-val-list(\text{tr,} \mu_{0} (\text{<s-stop: (SEMIC), <s-stop-incr: (APOST, EQ)}>))} \)

Ref.: \( \text{conc 9-52(207)} \)

(291) \( \text{test-name}(\text{tr,cl,ref-list}) = \)
    \( \text{is-compl-correct}(\delta(\text{cl}; \text{elem} \cdot \text{length}(\text{cl})), \text{is-ic-basic-ref}) \land \)
    \( (\exists i) (\text{io-idl}(i \mu_{0} (\text{io-ref}_{1}) = \text{s-id-list} \cdot \text{elem}(i, \text{ref-list})) \rightarrow \)
    \( \text{get-data-lp}(\text{tr,io-idl}; \text{io-ref}_{1}, \text{s-n}(\text{io-ref}_{1}), \text{const1}) ; \)
    \( \text{ccn1: \text{ak-cond-nl}(\text{io-cl}(\text{io-ref}_{1})))} \)
    \( \text{is-n* file}(\text{tr}) \rightarrow \)
    \( \text{get-data-lp}(\text{tr,ref-list}) ; \)
    \( \text{call-ic-cond-1}(\mu_{0} (\text{<s-fis-f(fu-el1)>, <s-cond:NAMIR, <s-datafield-s-chif:cl>>}) \)
    \( \text{is-n* string}(\text{tr}) \rightarrow \)
    \( \text{get-data-lp}(\text{tr,ref-list}) ; \)
    \( \text{call-cond (ERROR)} \)

where:
    \( \text{io-ref}_{1} = \text{parzing}(\delta(\text{cl}; \text{elem} \cdot \text{length}(\text{cl})), \text{is-ic-basic-ref}) \)
    \( \text{ref}_{1} = \text{elem}(\text{bi})(\text{io-idl}(\text{io-ref}_{1}) = \text{s-id-list} \cdot \text{elem}(i, \text{ref-list})), \text{ref-list}) \)

cont'd
Ref.: is-compl-correct 11-71(199)
is-ic-basic-ref 11-69(190)
call-ic-cond-1 10-24(72)
call-cond 10-18(54)
parsing 11-71(197)

(292) io-idl(io-ref) =
  \langle \text{mk-id\^{unnest}}\text{-cl}\text{-elem}(1,\text{ioc-ref})\rangle
  \text{LIST} \text{mk-id}\text{-unnest}\text{-cl}\text{-elem}(4,\text{elem}(i,\text{elem}(2,\text{ioc-ref})))

where:
  \text{lg}_i = \text{length}\text{-elem}(2,\text{ioc-ref})

Ref.: unnest-cl 11-71(198)

(293) io-sl(io-ref) =
  \text{is-\text{-BLANK}}\text{-list} (\text{sp}_3) \rightarrow < >
  T \rightarrow \langle \text{elem}(4,\text{sp}_3)\rangle \text{LIST} \text{elem}(4,\text{elem}(i,\text{elem}(5,\text{sp}_3)))

where:
  \text{sp}_3 = \text{elem}(3,\text{ioc-ref})
  \text{lg}_3 = \text{length}\text{-elem}(5,\text{sp}_3)

(294) \text{mk-const-cl}(s_l) =
  \text{is-\text{-}\langle\rangle}(s_l) \rightarrow \text{PASS}\langle\rangle
  \text{is-size-cond} (\text{eda}_4, \text{v}_1) \rightarrow \text{call-cond}(\text{size})
  T \rightarrow \\
  \text{conc} (\langle\text{v}_0 (\langle\text{s-data-\text{eda}_4}, <\text{s-v:}\text{v}_4>) >, \text{const}_1\rangle; \\
  \text{const}_1: \text{mk-const-cl}(\text{tail}(s_l))

where:
  \text{eda}_4 = \text{num-eda}\text{-head}(s_l)
  \text{v}_4 = \text{num-val}\text{-head}(s_l)

for: is-io-sub-list(s_l)

Ref.: is-size-cond 9-6(28)
call-cond 10-18(54)
conc 9-52(207)
num-val 11-72(206)
is-ic-sub 11-70(175)
(295) \[ \text{get-data} \rightarrow (\text{tr}, \text{idl}, n, \text{constl}) = \]

\[
\begin{align*}
\text{call-check-cond} \langle \text{ref} \rangle; \\
\text{count-convert-assign} (\text{tr}, g, \text{op}, 0); \\
\text{cp}: \text{mak-pp-ep} (\text{tr}, cl); \\
\text{cl}: \text{test-data} \rightarrow (\text{tr}, cl-4); \\
\text{cl-4}: \text{get-val-list} (\text{tr}, \mu_0 (\langle s\text{-stop-incr:COMMA}, s\text{-incr:BLANK} \rangle)); \\
\text{upd}: \text{lo-string} (\text{tr}); \\
g: \text{eval-ref-gen} (\text{ref})
\end{align*}
\]

where:

\[ \text{ref} = \mu_0 (\langle s\text{-id-list:idl}, s\text{-n:n}, s\text{-const-list:constl} \rangle) \]

for: is-id-list-1 (idl), is-n (n), is-const-list (constl)

Ref.: call-check-cond 10-21(64)
       count-convert-assign 11-80(223)
       eval-ref-gen 8-28(82)

(296) \[ \text{test-data} \rightarrow (\text{tr}, cl-4) = \]

\[
\begin{align*}
\text{is-SEMIC} \lor \text{is-COMMA} \lor \text{is-ENDM-SCAN} \cdot \text{last} (cl-4) \rightarrow \text{PASS:rp-field} (cl-4) \\
\text{T} \rightarrow \\
\text{test-data} \rightarrow (\text{tr}, cl); \\
\text{cl}: \text{get-val-list} (\text{tr}, \mu_0 (\langle s\text{-stop:SEMIC}, s\text{-stop-incr:APOSTE,COMMA,BLANK} \rangle)); \\
\text{upd}: \text{lo-string} (\text{tr})
\end{align*}
\]

(297) \[ \text{test-data} \rightarrow (\text{tr}, cl-5) = \]

\[
\begin{align*}
\text{is-SEMIC} \lor \text{is-COMMA} \lor \text{is-BLANK} \lor \text{is-ENDM-SCAN} \cdot \text{last} (cl-5) \rightarrow \\
\text{PASS:rp-field} (cl-5) \\
\text{T} \rightarrow \\
\text{test-data} \rightarrow (\text{tr}, cl-6); \\
\text{cl-6}: \text{conc} (cl-5, cl); \\
\text{cl}: \text{get-val-list} (\text{tr}, \mu_0 (\langle s\text{-stop:}, s\text{-stop-incr:APOSTE} \rangle))
\end{align*}
\]

Ref.: conc 9-52(207)

(298) \[ \text{test-data} \rightarrow (\text{tr}, cl-6) = \]

\[
\begin{align*}
\text{is-ENDM-SCAN} \cdot \text{last} (cl-6) \rightarrow \text{PASS:rp-field} (cl-6) \\
\text{T} \rightarrow \\
\text{test-data} \rightarrow (\text{tr}, cl-5); \\
\text{cl-5}: \text{conc} (cl-5, cl); \\
\text{cl}: \text{get-val-list} (\text{tr}, \mu_0 (\langle s\text{-stop:SEMIC}, s\text{-stop-incr:APOSTE,COMMA,BLANK} \rangle))
\end{align*}
\]

Ref.: conc 9-52(207)
11.6.4.3 Elementary and edit-directed input

11.6.4.3.1 Elementary input

Elementary input yields a string which is returned by `get-val-list` or control information returned by `get-file`. The argument `arg` defines how scanning is to be performed. If `arg` is an integer value, the underlying scan is edit-directed.

Metavariables

- `arg` is-intg-val \(\forall\) scanning argument
- `is-scan-arg`
- `el` is-bit-val \(\forall\) single input element
- `is-char-val`
- `is-LDEE` \(\forall\) `is-y` \(\forall\)
- `is-stop-inf`

(299) `get-val-list(tr, arg) =`

\[
\text{is-intg-val}(arg) \& arg \leq 0 \rightarrow \text{PASS} <\>
\]

\[
T \rightarrow \text{set-i}(tr, <\>, 0, arg)
\]

Ref.: `is-intg-val 9-3(5)`

(300) `is-scan-arg =`

\[
\langle s-end:is-ERROR-ENDE \& is-y > ,
\langle s-stop:is-char-val-set \& is-y > ,
\langle s-stop-increment:is-char-val-set \& is-y > ,
\langle s-increment:is-char-val-set \& is-y > \rangle,
\]

Ref.: `is-char-val 9-3(6)`

Note: The component `s-end` indicates whether an on-condition call has to be made if the end of the input stream or string is reached (eventually after data set switching has been tried). From the following two "stop"-components at least one must be non-empty. The component `s-stop` contains those characters, the input of which should terminate the scan without incrementing the position of the data set. The component `s-stop-increment` contains those characters, the input of which should terminate the scan after incrementing the position. The component `s-increment` contains the characters for which the scanning and incrementing should continue.

(301) `is-scan-class(v, arg, el) =`

\[
\text{is-set}(arg) \& el \in v(arg) \& is-y(arg) \& (v) (is-set)(arg) = el \& v(arg))
\]

for: \(v = \langle s-stop, s-stop-increment, s-increment, is-char-val(el) \rangle\)

Ref.: `is-char-val 9-3(6)`
Note: The scanning argument arg is such that every character e\(1\) belongs to exactly one of the scanning classes defined by the domain of \(\sigma\).

(302) \texttt{get-1(tr, vl, e\(1\), arg) =}
\begin{align*}
& (\text{is-0 } \lor \text{is-LIL}(e\(1\)) \rightarrow \\
& \text{get-1(tr, vl, e\(1\)-1, arg)}; \\
& e\(1\)-1: \text{get-2(tr, arg)} \\
& \text{is-intg-val(arg) } \& \text{arg } = 1 \rightarrow \text{PASS: vl}^c<e\(1\)> \\
& \text{is-intg-val(arg) } \rightarrow \text{get-1(tr, vl}^c<e\(1\)>\,\emptyset, \text{arg } - 1) \\
& \text{is-stop-inf(e\(1\)) } \rightarrow \text{PASS: vl}^c<\text{s-stop-inf(e\(1\)>)} \\
& T \rightarrow \text{get-1(tr, vl}^c<e\(1\)>\,\emptyset, \text{arg})
\end{align*}
Ref.: is-intg-val 9-3(5)

(303) \texttt{is-stop-inf =}
\begin{align*}
(\text{s-stop-inf: is-END-SCAN } \lor \text{is-char-val})
\end{align*}
Ref.: is-char-val 9-3(6)

Note: Stopping information is returned by the instructions \texttt{get-string} or \texttt{retry-stream-transmission} if the end of the input stream or string has been reached during scanning, and the component s-end of the scanning argument is empty.

(304) \texttt{get-2(tr, arg) =}
\begin{align*}
& \text{is-nres-string(tr) } \rightarrow \\
& \text{get-string(tr, vl, arg)}; \\
& \text{vl: pass-op-val(op)}; \\
& \text{op: s-g(fu-e\(1\))} \\
& \text{is-nres-file(tr) } \rightarrow \text{get-file(tr, arg)}
\end{align*}
Ref.: \texttt{cp-val 9-9(36)}
\texttt{gen-op-1 6-49(136)}

Note: The scalar generation or pseudo-generation corresponding to the string option cannot be of numeric picture type.
(305) \textbf{get-string}(tr,vl,\text {arg}) =

\begin{align*}
&\text{is-intg-val } \land \text{is-ERROR-END\text{-end}}(\text{arg}) \land h_{i_1} \geq \text{length}(vl) \rightarrow \text{call-cond}(\text{ERROR}) \\
h_{i_1} \geq \text{length}(vl) \rightarrow \text{PASS}:\mu_0(<\text{s-stop-inf}\text{-IBM-SCAN}>) \\
\text{is-intg-val}(\text{arg}) \land \text{is-scan-class}(\text{s-incr},\text{arg},rd_1) \rightarrow \\
\text{PASS}(rd_1); \\
\text{upd-fu-elem}(\text{s-string}(tr),s-hi,h_{i_1} + 1) \\
\text{is-scan-class}(\text{s-stop-incr},\text{arg},rd_1) \rightarrow \\
\text{PASS}\mu_0(<\text{s-stop-inf}\text{-rd}_1>) \\
\text{is-scan-class}(\text{s-stop},\text{arg},rd_1) \rightarrow \text{PASS}\mu_0(<\text{s-stop-inf}\text{-rd}_1>)
\end{align*}

where:

\begin{align*}
h_{i_1} &= s-hi(fu-el_1) \\
rd_1 &= \text{elem}(hi_1 + 1, v1)
\end{align*}

Ref.: 

\begin{align*}
\text{is-intg-val} 9-3(5) \\
\text{call-cond} 10-18(54) \\
\text{upd-fu-elem} 11-50(135)
\end{align*}

(306) \textbf{get-file}(tr,\text{arg}) =

\begin{align*}
\text{test-inf}(tr,\text{inf},\text{arg}); \\
\text{inf:stream-transmission}(tr,\text{arg})
\end{align*}

Ref.: 

\begin{align*}
\text{test-inf} 11-95(255) \\
\text{stream-transmission} 11-94(254)
\end{align*}

(307) \textbf{test-get-inf}(tr,\text{el}) =

\begin{align*}
\text{is-\text{el}}(\text{el}) \rightarrow \text{error} \\
\text{is-stop-inf}(\text{el}) \rightarrow \text{PASS}:\text{el} \\
\text{is-LDL}(\text{el}) \rightarrow \\
\text{PASS}(\text{el}); \\
\text{upd-fu-elem}(\text{s-file}(tr),\text{s-col},1) \\
(\text{is-char-val} \lor \text{is-bit-val})(\text{el}) \rightarrow \\
\text{PASS}(\text{el}); \\
\text{csey}(tr,\text{el}); \\
\text{upd-fu-elem}(\text{s-file}(tr),\text{s-col},\text{s-col}(\text{fu-el}_1) + 1)
\end{align*}

Ref.: 

\begin{align*}
\text{upd-fu-elem} 11-50(135) \\
\text{is-char-val} 9-3(6) \\
\text{is-bit-val} 9-4(11) \\
\text{csey} 11-121(327)
\end{align*}
11.6.4.3.2 Control formats

**Metavariables:**

<table>
<thead>
<tr>
<th>el</th>
<th>is-bit-val v is-char-val</th>
<th>single input element</th>
</tr>
</thead>
<tbody>
<tr>
<td>w</td>
<td>is-intg-val</td>
<td>width of control format item</td>
</tr>
</tbody>
</table>

(308) \[ \text{get-space}(tr, w) = \]
\[ \text{get-val-list}(tr, w) \]

(309) \[ \text{get-skip}(tr, w, el) = \]
\[ w = 0 \rightarrow \text{null} \]
\[ \text{is-LDEL}(el) \rightarrow \text{get-skip}(tr, w - 1, \emptyset) \]
\[ (\text{is-char-val} \lor \text{is-bit-val} \lor \text{is-Q})(el) \rightarrow \]
\[ \text{get-skip}(tr, w, el-1); \]
\[ el-1: \text{get-file}(tr, w) \]

Ref.:  is-char-val 9-3(6)
is-bit-val 9-4(11)

Note: The last alternative arises for a bit value only in connection with a bitstream column format.

(310) \[ \text{get-column}(tr, w, el) = \]
\[ w < s-col(fu-el_1) \rightarrow \]
\[ \text{get-column}(tr, w, el); \]
\[ \text{get-skip}(tr, 1, \emptyset) \]
\[ w = s-col(fu-el_1) \rightarrow \text{null} \]
\[ \text{is-LDEL}(el) \rightarrow \text{get-skip}(tr, 1, \emptyset) \]
\[ (\text{is-char-val} \lor \text{is-bit-val} \lor \text{is-Q})(el) \rightarrow \]
\[ \text{get-column}(tr, w, el-1); \]
\[ el-1: \text{get-file}(tr, w) \]

Ref.:  is-char-val 9-3(6)
is-bit-val 9-4(11)

Note: If positioning should be to the current line and the actual linesize is exceeded, skipping to the first column of the next line will occur.
11.6.4.3 Data formats

**Metavariables**

- \( ft \) \( \text{is-f-const} \lor \text{is-TLT} \) fixed-point or floating-point format type
- \( \text{const} \) \( \text{is-f-const} \lor \text{is-e-const} \lor \text{is-ENAN} \lor \text{is-} \) floating-point or fixed-point data field
- \( d.p \) \( \text{is-ittg-val} \) number of digits following decimal point, scaling factor
- \( edfo \) evaluated data format iter

**Abbreviation**

\( ft_c = \text{s-type} (edfo) \)

\[ (311) \quad \text{out-edit}(tr, edfo) = \begin{array}{l}
ft_c \in \{\text{CHAR, EHT, EMT}\} \land \text{is-0-s-w}(edfo) \rightarrow \text{ERROR} \\
\text{is-skipped-format}(edfo) \rightarrow \text{PASS}\* \\
T \rightarrow \\
\phantom{T \rightarrow } \text{ek-edit-cp}(tr, vl, edfo); \\
\phantom{T \rightarrow } \text{vl: get-val-list}(tr, \text{data-lg}(edfo)); \\
\phantom{T \rightarrow } \text{end-lg-string}(tr) \\
\end{array} \]

**Ref.:** is-skipped-format 11-86(238) 
\quad data-lg 11-85(235)

**Note:** The data field returned by the last alternative has exactly the length \( \text{data-lg} \) (\( edfo \)).
(312) \[ \text{sk-edit-op}(tr, vl, edfo) = \]
\[ \begin{align*}
\text{(is-< - is-*)}(vl) & \rightarrow \text{PASS: vl} \\
\text{(is-FLX - is-FLT)}(ft_0) & \rightarrow \\
& \text{sk-get-op}(eda, vl) \\
& \text{vos: pass-f-e-eda}(ft_0, \text{const}, \text{s-d (edfo), p_1}); \\
& \text{vl: pass-f-e-val}(ft_0, \text{const}, \text{s-d (edfo), p_1}); \\
& \text{const: verify-f-e(tr, ft_0, vl)} \\
& \text{is-FLT}(ft_0) \rightarrow \\
& \text{sk-get-op}(BIT, vl) \\
& \text{bl: verify-b(tr, vl)} \\
\text{(is-CHAR - is-BBIT)}(ft_0) & \rightarrow \text{sk-get-op}(ft_0, vl) \\
\text{is-EPIC}(ft_0) & \rightarrow \text{sk-get-op}(s-pic(edfo), vl) \\
\text{is-PIC}(ft_0) & \rightarrow \\
& \text{sk-get-op}(s-pic(edfo), vl) \\
& \text{vl-1: verify-2(tr, s-pic(edfo), vl)} \\
& \text{is-CFLX}(ft_0) \rightarrow \\
& \text{eval-cplx(cp-1, cp-2):} \\
& \text{cp-2: sk-edit-op(vl_2, s-imag(edfo));} \\
& \text{cp-1: sk-edit-op(vl_1, s-real(edfo))} \\
\end{align*} \]

where:
\[ p_1 = (\text{is-0-s-t-s(edfo)} \rightarrow 0, \text{tr} \rightarrow \text{s-p-s(edfo)}) \]
\[ \begin{align*}
vl_1 & = \text{LIST elem}(i, vl) \\
& \text{LIST elem}(i + l_{g_1}, vl) \\
\end{align*} \]
\[ l_{g_1} = \text{data-lg*s-real(edfo)} \\
\text{data-lg*s-imag(edfo)} \]

for: (is-* \text{is-char-val-list)(vl)}

Ref.: 
\[ \text{eval-cplx \ 12-46(94)} \]
\[ \text{data-lg \ 11-85(235)} \]
\[ \text{is-char-val \ 9-3(5)} \]

(313) \[ \text{verify-f-e(tr, ft, cl)} = \]
\[ \begin{align*}
\text{is-compl-correct(cl, pred_1)} & \rightarrow \text{PASS: elem(2, parsing(cl, pred_1))} \\
\text{T} & \rightarrow \\
& \text{verify-f-e(tr, ft, corr-cl)} \\
& \text{corr-cl: call-get-conv(tr, cl, wrong-pos(cl, pred_1))} \\
\end{align*} \]

where:
\[ \begin{align*}
pred_1 & = (\text{is-FLX(ft)} \rightarrow \text{is-f-data,} \\
& \text{is-FLT(ft)} \rightarrow \text{is-e-data}) \\
\end{align*} \]
ABSTRACT SYNTAX AND INTERPRETATION OF PL/I

Ref.: is-compl-correct 11-71(197)
      parsing 11-71(197)
      wrong-rec 11-71(201)

(314) is-f-data =
        is-FLANK-list \& is-f-const-data

(315) is-f-const-data =
        (is-fi-arithm-const(dt) \& is<>\&<elem(3,dt) \& is-io-decimal-fixed\&<elem(2,dt))

Ref.: is-io-arithm-const 11-67(171)
      is-io-decimal-fixed 11-68(179)

(317) is-e-data =
        (is-fi-arithm-const(dt) \& is<>\&<elem(3,dt) \&
         (is/io-decimal-fixed \& is/io-decimal-float)(<elem(2,dt)))

Ref.: is-fi-arithm-const 11-67(171)
      is/io-decimal-fixed 11-68(179)
      is-io-decimal-float 11-68(180)

(319) f-e-eda(ft, const, d, p) =
        is-fi-arithm-const(const) \rightarrow \mu (intg-eda(1):<s-scale-f:d - p>)
        is-FLT(ft) \rightarrow e\&\&a\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&\&##
(320) \[ i-e-val(f, c, d, p) = \]
\[
\begin{align*}
&\text{is-io-arithm-const}(c) \rightarrow 0 \\
&\text{is-FLT}(f) \land \text{is-POINT-elem}(2, \text{elem}(1, \text{elem}(2, c))) \lor \text{is-FIX}(f) \land \\
&\text{is-POINT-elem}(2, \text{elem}(2, c)) \rightarrow \\
&\text{real-val}(c) \cdot 10^p \\
&T \rightarrow \text{real-val}(c) \cdot 10^{(p - d)}
\end{align*}
\]

Ref.: \text{is-io-arithm-const} 11-67(171)
\text{real-val} 11-72(207)

(321) \[ \text{verify-b}(t, c) = \]
\[
\begin{align*}
&\text{is-corr-correct}(c, \text{is-b-data}) \rightarrow \text{PASS}: \text{char-bit-list}(<\text{elem}(2, d)t_1>, \text{elem}(3, d)t_2)) \\
&T \rightarrow \\
&\text{verify-b}(t, \text{corr-c}) ; \\
&\text{corr-c} : \text{call-set-corr}(t, c, \text{wrong-pos}(c, \text{is-b-data}))
\end{align*}
\]

where:
\[ d_t = \text{carsing}(c, \text{is-b-data}) \]

Ref.: \text{is-corr-correct} 11-71(199)
\text{wrong-pos} 11-71(201)
\text{carsing} 11-71(197)

(322) \[ \text{is-b-data} = \]
\[
\begin{align*}
&(<\text{elem}(1), \text{is-Blank-list}>, \\
&<\text{elem}(2), \text{is-ic-bit-char}>, \text{, elem}(3), \text{is-ic-bit-char-list}>, \\
&\text{, elem}(4), \text{is-Blank-list}>)
\end{align*}
\]

Ref.: \text{is-ic-bit-char} 11-68(177)

(323) \[ \text{char-bit-list}(c) = \]
\[
\begin{align*}
&\text{LIST}\quad \text{single-char-bit}\text{-elem}\text{(i, c)} \\
&\text{L} - t
\end{align*}
\]

where:
\[ \text{lg}_i = \text{length}(c) \]

for: \text{is-ic-bit-char-list}(c)

Ref.: \text{single-char-bit} 9-34(135)
\text{is-ic-bit-char} 11-68(177)
ABSTRACT SYNTAX AND INTERPRETATION OF PL/I

(324) \texttt{verify-p}(tr,pic,cl) =
\begin{align*}
\text{is-p-match}(pic,cl) \land \text{is-bin-pic}(pic) &\rightarrow \text{PASS:char-bit-list}(cl) \\
\text{is-r-match}(pic,cl) &\rightarrow \text{PASS:cl} \\
\text{T} &\rightarrow \\
\text{verify-p}(tr,\text{corr-cl})
\end{align*}
\begin{align*}
\text{corr-cl} &\coloneqq \text{call-get-comp}(tr,cl,i_0) \\
\text{where:} \\
\begin{cases}
  i_0 &\coloneqq (\forall i \geq 1 \land \text{is-p-leftmatch}(pic,cl-i_1) \land \\
  \quad (\forall j \leq j < i \rightarrow \text{is-p-leftmatch}(pic,cl-j_1))) \\
  cl-i_1 &\coloneqq \text{LIST}_{k=1}^{\text{LIST}_{j=1}^{\text{LIST}_{k=1}}^{\text{elem}(k,cl)}} \\
  cl-j_1 &\coloneqq \text{LIST}_{k=1}^{\text{LIST}_{j=1}^{\text{LIST}_{k=1}}^{\text{elem}(k,cl)}}
\end{cases}
\end{align*}

(325) \text{is-p-match}(pic,cl) =
\begin{align*}
\text{is-bin-pic}(pic) \land \text{is-jo-bit-char-list}(cl) &\rightarrow \\
\text{is-pic-match}(pic,\text{char-bit-list}(cl)) \\
\text{is-bin-pic}(pic) &\rightarrow \text{T} \\
\text{T} &\rightarrow \text{is-pic-match}(pic,cl)
\end{align*}

\text{Ref.:} \quad \text{is-jo-bit-char} 71-68(177) \\
\text{is-pic-match} 9-63(246)

(326) \text{is-p-leftmatch}(pic,cl) =
\begin{align*}
(\exists \text{list})(\text{is-list}(\text{list}) \land \text{is-p-match}(pic,cl\text{-list}))
\end{align*}
11.7 SPECIAL CASES OF SYSTEM TRANSMISSION

This section defines interpretation of copy option, spacing to tabulators, data-directed output by check standard system on-condition, and the starting of a new page through not-signalled endpage standard system on-condition.

Metavariable:

tr is-tr-car transmission parameter
11.7.1 COPY DEFINITION

(327) \texttt{copy(tr,inf) =}
\begin{align*}
\text{is-\texttt{str-copy}(tr) } & \rightarrow \text{null} \\
\text{is-\texttt{t}(inf) } & \\
\text{def-\texttt{copy}(w); } & \\
\text{u:int-\texttt{str-int-oper} } & \\
\text{t } & \rightarrow \\
\text{put-val-list(tr-p,inf-char1>); } & \\
\text{r-file(tr-r):int-\texttt{str-print-open}; } & \\
\text{tr-p:pass(wu(<t-\texttt{str-op}>)}) & \\
\end{align*}

\textbf{where:}
\begin{align*}
\text{inf-char1 } = (\text{is-char-val(}\text{inf} ) \rightarrow \text{inf,} \\
\text{is-bit-val(}\text{inf} ) \rightarrow \text{single-bit-char}(\text{inf})
\end{align*}

\textbf{fer:}\texttt{is-\texttt{str-s-str}(tr), is-\texttt{t}(inf) \lor is-\texttt{char-val} \lor is-\texttt{bit-val}(\text{inf})}

\textbf{Ref.:}
\begin{itemize}
\item \texttt{int-\texttt{str-int-oper} 11-26(76)}
\item \texttt{int-\texttt{val-list} 11-91(247)}
\item \texttt{is-char-val 9-3(6)}
\item \texttt{is-bit-val 9-4(17)}
\item \texttt{single-bit-char 9-35(139)}
\end{itemize}

\textbf{Note:} The first alternative specifies no copying at all. The second alternative is normally taken before a whole data field is copied. The last alternative is the copying of a single character or bit, the latter being converted to character.

(328) \texttt{def-\texttt{copy}(w) =}
\begin{itemize}
\item \texttt{for:is-t(w)}
\end{itemize}

\textbf{Note:} This instruction is implementation defined. It may cause additional actions on that standard system print file on which copying will follow.
11.7.2 Positioning to Tabulators

(329) \( \text{tab}(\text{tr}, \text{eda}, \text{cl}) = \)

\[
\begin{align*}
\text{is-num}(u_4) & \Rightarrow \text{is-FET} \cdot \text{stream-base} \cdot u_4(\text{FU}) \rightarrow \text{null} \\
\text{is-FET} \cdot \text{next-tab}(1) & \land \text{next-tab}(1) > \text{lsz} \Rightarrow \text{error} \\
\text{is-num-type}(\text{eda}) & \land \text{is-int-val} \cdot \text{next-tab}(\text{col}_1) \\
& \land \text{next-tab}(\text{col}_1) + \text{length}(\text{cl}) - 1 \leq \text{lsz} \lor \text{is-num-type}(\text{eda}) \land \\
& \text{is-int-val} \cdot \text{next-tab}(\text{col}_1) \land \text{next-tab}(\text{col}_1) \leq \text{lsz} \Rightarrow \\
\text{put-file}(\text{tr}, \text{next-tab}(\text{col}_1), \text{TABLE}) \\
\text{T} & \Rightarrow \\
\text{put-file}(\text{tr}, \text{next-tab}(1), \text{TABLE}); \\
\text{put-skip}(\text{tr}, 1, \text{E})
\end{align*}
\]

where:

\( u_4 = \text{s-file}(\text{tr}) \)
\( \text{fu-el}_1 = (\text{is-fu-el} \cdot u_4(\text{FU}) \rightarrow u_4(\text{FU}), \text{E} \Rightarrow \text{error}) \)
\( \text{col}_1 = \text{s-col}(\text{fu-el}_1) \)
\( \text{lsz}_1 = \text{s-lsz}(\text{fu-el}_1) \)

for: \( \text{is-FET} \cdot \text{s-file}(\text{tr}), (\text{is-FET} \land \text{is-eda})(\text{eda}), \text{is-char-val-list}(\text{cl}) \)

Ref.:

\( \text{stream-base} \; 11-84(233) \)
\( \text{is-num-type} \; 9-21(91) \)
\( \text{is-int-val} \; 9-3(5) \)
\( \text{put-file} \; 11-94(253) \)
\( \text{put-skip} \; 11-97(259) \)
\( \text{is-fu-el} \; 3-25(92) \)
\( \text{is-char-val} \; 9-3(6) \)

Note: The first alternative specifies no spacing to tabulators for string options and for non-print files. The third alternative specifies spacing to the next tabulator in the current line, the fourth alternative starts a new line and then spaces to the next tabulator. The second alternative tests for at least one tabulator within the range specified by the linesize.

(330) \( \text{next-tab}(\text{col}) = \)

\[
\begin{align*}
\neg(\exists i) (i \in \text{TAB-SET} \land \text{col} \leq i) & \rightarrow \text{F} \\
\text{T} & \rightarrow (\exists i) (i \in \text{TAB-SET} \land \text{col} \leq i \land (\forall j) (\text{col} \leq j < i \Rightarrow j \notin \text{TAB-SET}))
\end{align*}
\]

for: \( \text{is-int-val}(\text{col}), \text{is-int-val}(i), \text{is-int-val}(j) \)

Ref.:

\( \text{is-int-val} \; 9-3(5) \)

Note: \( \text{TAB-SET} \) is a set of positive integers (cf. axiom in section 11.3.3). The function next-tab yields the next tabulator position out of \( \text{TAB-SET} \) if there is any which is greater or equal to \( \text{col} \) (which is the number of the column to be occupied next).
11.7.3 CHECK STANDARD SYSTEM ACTION

(331) \texttt{syst-check-exec(ref) =}

\begin{verbatim}
put-val-list(tr,\langle\text{SEQ}IC\rangle);
expand-exp(tr,ref);
s-file(tr):pass(u);
und-fu-elem(u,s-count,hint-op(0));
\textit{int-st-pret-open};
tr:pass(\mu_0(\langle s-st:PUT\rangle,\langle s-type:CHECK-DATA\rangle))
\end{verbatim}

for: is-ref(ref)

Ref.: \begin{itemize}
\item put-val-list 11-91(247)
\item expand-exp 11-77(215)
\item und-fu-elem 11-30(135)
\item hint-op 9-10(46)
\item int-st-pret-open 11-26(76)
\end{itemize}

Note: Besides opening the standard system print file of the current task, the
initiating actions are performed like for a put statement (cf. section
11.4.3). The special type CHECK-DATA is used in the case distinction of
\texttt{conv-char-string-1} (cf. section 11.6.3.1). Notice that the reference \texttt{ref}
is guaranteed to be suitable for data-directed output, i.e.
is-data-dir-ref (ref, AT) holds or \texttt{ref} refers to a label or entry constant.

11.7.4 ENDPAGE STANDARD SYSTEM ACTION

(332) \texttt{syst-endpage-exec(f) =}

\begin{verbatim}
is-open-file(f,IP) & is-PRT\texttt{-stream-base\_u}_1(\mu) ->
\textit{put-page}(\mu_0(\langle s-st:PUT\rangle,\langle s-file:u_1\rangle))
\end{verbatim}

\texttt{T -> error}

where:
\begin{itemize}
\item \texttt{u_1 = asscc-u(f,IP)}
\end{itemize}

for: (is-s-st-pret \lor is-n)(f)

Ref.: \begin{itemize}
\item is-open-file 11-24(71)
\item stream-base 11-80(233)
\item put-page 11-98(262)
\item asscc-u 11-24(76)
\item is-s-st-pret 3-24(90)
\end{itemize}

Note: No implicit opening is performed. If there is no file or it is non-print,
the instruction is erroneous.
11.8 DISPLAY TRANSMISSION

The evaluation of the display, event, and reply options (s-idto) is defined in connection with record I/O in section 11.4.1. Attaching of an I/O event is described in section 11.4.2. This section defines only the output of the display message and the input of any reply message.

Metavariables

<table>
<thead>
<tr>
<th>cl</th>
<th>is-char-val-list v</th>
<th>message</th>
</tr>
</thead>
<tbody>
<tr>
<td>n</td>
<td>is-n</td>
<td>name of a message</td>
</tr>
<tr>
<td>s</td>
<td>is-gen v</td>
<td>destination of the reply message</td>
</tr>
<tr>
<td>i</td>
<td>is-intg-val index for a message in the message part</td>
<td></td>
</tr>
</tbody>
</table>

(333) \( \text{display}(cl,n) = \)

\[
\text{set}(\text{u} \in \text{display}(s) \leftarrow \text{u} \in \text{name}(s) ,
\text{message : list elem(i,cl)o})
\]

where:
\( l_{g_1} = \min(\text{length}(cl),\text{MESS-LENGTH}) \)

(334) \( \text{wait-is-reply}(g,n) = \)

\( \text{is-}\text{vr}(g) \rightarrow \text{null} \)

\( (3i)(n = \text{name} \in \text{elem}(i,g)) \land \text{is-vr}(vr) \rightarrow \text{g-g} : vr \)

\( (3i)(n = \text{name} \in \text{elem}(i,g)) \rightarrow \text{error} \)

\( T \rightarrow \)

\( \text{wait-for-reply}(g,n); \)

\( \text{set-wait-state}(n) \)

where:
\( vr_1 = \text{assign-idto}(q,cl_1,s-\text{cbif}(CS),S,Pd) \)

\( cl_1 = \text{elem}(L_i)(n = \text{name} \in \text{elem}(i,g),g) \)

Ref.: \( \text{is-vr} 3-15(32) \)

assign-idto 11-62(161)

Note: The assignment to the reply option will not cause on-condition calls. The last alternative is taken as long as the component s-reply(M) is not changed by an environmental influence in a way that the second alternative can be taken (cf. 3.7.2, is-reply-step).

124 11. INPUT AND OUTPUT
(335) \[ \text{get-wait-state}\_7(n) = \]

\[ (\forall i \; (n = s\text{-name}\_s\text{-elem}(i, n) \rightarrow \text{null}) \]

\[ \text{wait} \rightarrow s\text{-te}_{\mu}(\text{wait}; <s\text{-wait:}>) \]

Note: This instruction may also set an event specification into the wait state.
The first six sections of this chapter handle references to built-in functions which occur in expressions to be evaluated, the seventh deals with the assignment to pseudo-variables and the evaluation of pseudo-generators. The first four sections describe the actions which are common to all built-in functions: The syntactic expansion of an aggregate reference into scalar ones (section 12.1), the evaluation of a scalar reference (section 12.2), the determination of data attributes (section 12.3), and built-in functions as values of entry-parameters (section 12.4). Section 12.5 contains a survey of all built-in functions by a table giving their characteristic properties. Section 12.6 describes the evaluation of the individual built-in functions.

12.1 SYNTACTIC EXPANSION OF BUILT-IN FUNCTION REFERENCES

References to built-in functions may be part of aggregate expressions, aggregate expressions are treated by syntactic expansion into a sequence of scalar expressions (cf. 8.1) where each step in the expansion process is accomplished by application of the modifying instruction $\texttt{mod}$ to the text of the expressions. On modifyling references to built-in functions the distinction is to be made between expanding arguments and non-expanding arguments of the built-in functions. An aggregate expression in an expanding argument place is modified according to the rules for modification of sub-expressions of an expression. Expressions in non-expanding argument places are not changed.

The modification of built-in function references is made by the instruction $\texttt{mod-builtin (ref, eva, i)}$ where ref is the text of the reference, eva is the evaluated aggregate attribute of the master variable for the expansion, and i is the number of the sub-part, which is to be referenced by the modified reference (or a sub-aggregate name in case of a BY NAME expansion, cf. 8.1.1). An argument place is expandable if $\texttt{arg-class (id, i) = EXPND}$ where id is the identifier of the built-in function and i is the number of the argument except the case of $\texttt{id = mk-id(MIN)}$ or $\texttt{id = mk-id(MAX)}$ where $\texttt{i = 1}$ (cf. 12.5.1.3).

Metavariables:

- $\texttt{ref}$ is-\texttt{ref} the reference to a built-in function
- $\texttt{eva}$ is-\texttt{eva} the evaluated aggregate attribute of the master variable
- $\texttt{i}$ is-\texttt{intg-val} or is-\texttt{id}

12. BUILT-IN FUNCTIONS AND PSEUDO-VARIABLES 1
12.2 EVALUATION OF BUILT-IN FUNCTION REFERENCES

This section describes the evaluation of a built-in function reference occurring in a scalar expression to be evaluated (cf. 8.2.3). If it is not passed to an entry-variable (cf. 12.9), the reference to a built-in function is evaluated in two steps: First the evaluation of the argument list, and second, the activation of the individual built-in function. The latter is performed by the instruction eval-built-in-1 which performs the case distinction according to the different built-in identifiers and is defined by the table given in section 12.5.

The argument evaluation constructs a list of "built-in arguments", for each argument expression occurring in the text of the reference one built-in argument. The evaluation of a single argument is governed by the function arg-type(id,i), defined by the table in section 12.5, which specifies for each individual built-in function identifier id and argument position i the type of the built-in argument. The possible types are: an operand, an operand list, an integer value, evaluated aggregate attributes, a generation, the text of the argument expression itself. The function arg-test(id,i), defined by the table in 12.5, id being the built-in function identifier and i the argument position, governs in part the argument evaluation and tests - during the argument evaluation - the arguments for correctness. For arguments of the type operand or operand list the single operands are converted to (incomplete) data attributes specified by the function arg-eda(id,i), which also is defined by the table in section 12.5 for each
individual built-in identifier id and argument position i. Finally, the length of
the argument list is tested by means of the function \textit{lmin(id)} which is defined for
each built-in identifier id by the table in section 12.5 and denotes the minimum
length of the argument list and the function \textit{arg-type(id,i)}, which yields \texttt{G} if i
exceeds the maximum length.

### Metavariables

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>is-id</td>
</tr>
<tr>
<td>i</td>
<td>is-intg-val</td>
</tr>
<tr>
<td>ref</td>
<td>is-ref</td>
</tr>
<tr>
<td>expr</td>
<td>is-expr</td>
</tr>
<tr>
<td>expr</td>
<td>is-or</td>
</tr>
<tr>
<td>cpl</td>
<td>is-cpl</td>
</tr>
</tbody>
</table>

(3) \texttt{eval-built-in(ref)} =

\begin{align*}
\texttt{eval-built-in-1(id_0,bal);} \\
\texttt{bal:builtin-arg-list(arg-list\_s\_ar(ref),id_0,0)}
\end{align*}

where:
\begin{align*}
id_0 = \text{head\_s\_id-list(ref)}
\end{align*}

Ref.: \texttt{eval-built-in-1 12-17(36)} \texttt{arg-list 8-4(6)}

(4) \texttt{builtin-arg-list(expr-list,id,i)} =

\begin{align*}
is\texttt{-<}(expr-list) & \quad \text{if } i \geq \text{lmin(id)} \rightarrow \text{PASS:<>} \\
is\texttt{-<}(expr-list) & \quad \rightarrow \text{error} \\
T & \rightarrow \\
& \text{fl-list(ba,bal);} \\
& \texttt{bal:builtin-arg-list(tail(expr-list),id,i+1);} \\
& \texttt{ba:builtin-arg(head(expr-list),id,i_0)}
\end{align*}

where:
\begin{align*}
i_0 = \\
\text{id } \in [\text{mk-id(MIN)},\text{mk-id(MAX)}] \rightarrow 1 \\
T \rightarrow i + 1
\end{align*}

for: \texttt{is-expr-list(expr-list)}

Ref.: \texttt{lmin 12-16(32)}
(5) \[ \text{builtin-arg}(id,i,expr) = \]
\[ \text{is-OP-arg-type}(id,i) \rightarrow \text{builtin-arg-op}(id,i,expr) \]
\[ \text{is-OP*arg-type}(id,i) \rightarrow \text{builtin-arg-op}(id,i,expr) \]
\[ \text{is-INTC-arg-type}(id,i) \rightarrow \text{builtin-arg-intc}(id,i,expr) \]
\[ \text{is-GEN-arg-type}(id,i) \rightarrow \text{builtin-arg-ifa}(expr) \]
\[ \text{is-GEN-arg-type}(id,i) \rightarrow \text{builtin-arg-gen}(id,i,expr) \]
\[ \text{is-INT-arg-type}(id,i) \rightarrow \text{pass}:expr \]
\[ \text{is-GEN-arg-type}(id,i) \rightarrow \text{error} \]

Ref.: arg-type 12-15(34)

(6) \[ \text{builtin-arg-op}(id,i,expr) = \]
\[ \text{test-builtin-arg-op}(op, \text{arg-test}(id,i)); \]
\[ \text{convert}(\text{aggr-scalar-arg-eda}(id,i), \text{op-1}, \text{area}_2); \]
\[ \text{op-1}: \text{test-eval-exp}(expr,id,1) \]

where:
\[ \text{area}_2 = s\text{-area}\text{*final-ret-type}\text{*da}\text{*aggr-exp}(expr,AT) \]

Ref.: arg-test 12-16(36)
\[ \text{convert} 8-8(17) \]
\[ \text{aggr-scalar} 8-23(64) \]
\[ \text{arg-eda} 12-15(35) \]
\[ \text{final-ret-type} 8-28(80) \]
\[ \text{aggr-exp} 8-23(62) \]

(7) \[ \text{test-eval-exp}(expr,id,i) = \]
\[ \text{is-EXPND-arg-class}(id,i) \rightarrow \text{eval-exp-2}(expr) \]
\[ T \rightarrow \text{eval-exp}(expr,2) \]

Ref.: arg-class 12-10(33)
\[ \text{eval-exp-2} 8-79(50) \]
\[ \text{eval-exp} 8-10(24) \]

(8) \[ \text{test-builtin-arg-op}(op,test) = \]
\[ (\text{is-REAL}(test) \Rightarrow \text{is-REAL*modal-op-da}(op)) \& \]
\[ (\text{is-CPLX}(test) \Rightarrow \text{is-CPLX*modal-op-da}(op)) \rightarrow \]
\[ \text{pass}(op) \]
\[ T \rightarrow \text{error} \]

for\: test \in \{\text{REAL},\text{CPLX},T\}

Ref.: cp-da 9-9(37)
(9) built-in-arg-orl(id,i,expr) =
    test-cpl(aggr-expr(expr,AT),arg-test(id,i)) ->
    expr->pass-list-dop(dop):
      dop:eval-dummy-expr(expr,1,aggr)
    expr->pre-eval(expr)

where:
  aggr = insert-edas(erg-edas(id,i),aggr-expr(expr,AT))

Ref.: aggr-expr 8-23(62)
      arg-test 12-16(36)
      list-dop 6-37(732)
      eval-dummy-expr 6-24(63)
      pre-eval 8-11(26)
      arg-edas 12-15(35)

(10) test-cpl(aggr,test) =

is-AGR(test) & is-array(aggr) \& is-LIN-ARR(test) \& is-scalar-s-elem(aggr) \& is-LIN(test) & (is-scalar-s-elem(aggr) \& is-scalar(aggr)) \& is-T(test)

(11) insert-edas(eda,aggr) =

is-array(aggr) --> g(aga,<$s-elem:insert-edas(eda, s-elem(aggr))$>)

is-struc(naggr) --> \[\frac{\text{length}(aggr)}{\text{length}(aggr)}\] \(v_0(\text{<s-aggr:insert-edas(eda, s-aggr-elem(i,aggr))>})\)

is-scalar(aggr) --> aggr-scalar-complete-tg(eda, s-da(aggr))

for: is-edas(eda), is-aggr(aggr)

Ref.: aggr-scalar 8-23(64)
      complete-tg 9-37(149)

(12) built-in-arg-intq(id,i,expr) =

is-EXP\&arg-class(id,i) --
    test-intq-nup(v, arg-test(id,i));
    eval-intq-expr(1)(expr)

is-TERM\&arg-class(id,i) --
    test-intq-nup(v, arg-test(id,i));
    eval-intq-expr(expr)

is-CONST\&arg-class(id,i) \& test-intq(expr, arg-test(id,i)) --
    pass:intq-val(expr)

cont'd
Ref.:  
12-14(33)  
12-16(36)  
8-22(60)  
6-10(27)

(13) \( \text{test-intg} (\text{expr}, \text{test}) = \)

\[ \begin{align*}
& \text{is-INTG(test)} \rightarrow \text{is-intg-const (expr)} \\
& \text{is-S-INTG(test)} \rightarrow \text{is-signed-intg (expr)}
\end{align*} \]

\text{for: } \text{test} \in \{ \text{INTG, S-INTG} \}

Ref.:  
6-10(27)

(14) \( \text{eval-intg-expr-1} (\text{expr}) = \)

\[ \begin{align*}
& \text{pass-op-val} (\text{op}) : \\
& \text{cp: convert-1 (aggr-scalar \{BINTG-EXA\}, \text{op-1})} \\
& \text{op-1: eval-expr-2 (expr)}
\end{align*} \]

Ref.:  
9-3(36)  
8-23(64)  
8-19(50)

(15) \( \text{test-intg-num} (v, \text{test}) = \)

\[ \begin{align*}
& \text{is-INTG(test)} \rightarrow (1 \leq v) \rightarrow \text{PASS:v} \\
& T \rightarrow \text{ERROR}
\end{align*} \]

\text{for: } \text{test} \in \{ \text{INTG, S-INTG} \}, \text{is-intg-val (v)}

Ref.:  
9-3(5)

(16) \( \text{builtin-arg-eva} (\text{expr}) = \)

\[ \begin{align*}
& \text{array-eva-expr} (\text{expr-1}) : \\
& \text{expr-1: pre-eval (expr)}
\end{align*} \]

Ref.:  
8-26(76)  
8-11(26)

(17) \( \text{builtin-arg-gen} (\text{id}, i, \text{expr}) = \)

\[ \begin{align*}
& \text{is-EXPR-arg-class (id, i) \& is-var-ref (expr, AT)} \rightarrow \text{eval-ref-gen-1 (expr)} \\
& \text{is-var-ref (expr, AT)} \rightarrow \text{eval-ref-gen (expr)}
\end{align*} \]

Ref.:  
12-14(33)  
8-9(21)  
8-28(63)  
8-28(62)

6 12. BUILT-IN FUNCTIONS AND PSEUDOC-VARIABLES
12.3 THE AGGREGATE ATTRIBUTES OF BUILT-IN FUNCTION REFERENCES

The aggregate attributes of the aggregate returned by a built-in function reference are evaluated by the function `aggr-builtin(id, expr-list, at)` where id is the identifier and expr-list is the argument list of the built-in function reference and at is the attribute directory. The function is needed in the definition of the function `aggr-ref(ref, at)` in 8.2.5.

If all expanding arguments of the built-in function reference are scalar then the operand returned is also scalar. The aggregate attributes of the returned scalar are given by `aggr-scalar-builtin(id, list, at)`, where id is the identifier of the built-in function reference, list contains arguments corresponding to the arguments of the reference, and at is the attribute directory. These elements are either data attributes (the data attributes of the corresponding argument for expanding argument places, or the data attributes of the base element of the corresponding argument array, for array argument places), or aggregate attributes (of the corresponding argument in an aggregate argument place), or integer values (the value of the corresponding argument if the argument requires an integer constant, or asterisks * if the corresponding argument is irrelevant for the aggregate attributes of the returned operand.

The function `da-builtins` is defined by the table in chapter 12.5.

**Metavariables**

- `id` is-id: the built-in function identifier
- `expr` is-expr: an argument position of the built-in function
- `i` is-intg-val: an argument position of the built-in function
- `expr-list` is-expr-list: the list of argument expressions
- `at` is-at: the attribute directory

(18) \[ \text{aggr-builtin}(id, \text{expr-list}, at) = \]
\[ \text{aggr-builtin-1}(id, \text{list}_1, at) \]

where:
- \[\text{list}_1 = \text{LIST} \text{ list-elem} (\text{arg-class}(id, i_1), \text{elem}(i, \text{expr-list}), at)\]
- \[i_1 = \{ \text{id} \in \{\text{mk-id}(\text{MAX}), \text{mk-id}(\text{MIN})\} \rightarrow 1, \]
\[i \rightarrow i\]

Ref.: arg-class 12-14(33)
(19) \[ \text{list-eler}(\text{arg-cl}, \text{expr}, \text{at}) = \]
\[ \text{is-EXPR}(\text{arg-cl}) \rightarrow \text{aggr-expr}(\text{expr}, \text{at}) \]
\[ \top \rightarrow \text{expr} \]

\[ \text{for:} \text{arg-cl} \in \{ \text{EXPR}, \text{CONST}, \text{ACT}, \text{ARK}, \text{IRK}, \text{IRKII} \} \]

\[ \text{Ref.:} \quad \text{aggr-expr} \ 8-23(62) \]

(20) \[ \text{aggr-built-in-1}(\text{id}, \text{list}, \text{at}) = \]
\[ \text{length}(\text{list}) \]
\[ \text{Et} \quad (\text{is-EXPR}\text{arg-class}(\text{id},\text{i1}) \Rightarrow \text{is-scalar}\text{elem}(\text{i},\text{list})) \Rightarrow \]
\[ \text{aggr-built-in-2}(\text{id}, \text{list}, \text{at}) \]

\[ (\text{ii}) (\text{is-EXPR}\text{arg-class}(\text{id},\text{i1}) \& \text{is-array}\text{elem}(\text{i},\text{list})) \& \]
\[ \text{length}(\text{list}) \]
\[ \text{Et} \quad (\text{is-EXPR}\text{arg-class}(\text{id},\text{i1}) \& \text{is-array}\text{elem}(\text{i},\text{list})) \Rightarrow \]
\[ (\text{i}) \quad \text{dim}(\text{elem}(\text{i}, \text{list})) = \text{dim-list} \Rightarrow \]
\[ \text{ke}(<\text{e-ld}>, <\text{e-ub}>) \]
\[ <\text{e}\text{-elem}>, \text{aggr-built-in-1}(\text{id}, \text{Llist} \text{array}\text{elem}(\text{id}, \text{i1}, \text{elem}(\text{i}, \text{list})), \text{at}) > \]
\[ \top \rightarrow \text{aggr-built-in-contd-1}(\text{id}, \text{list}, \text{at}) \]

\[ \text{where:} \]
\[ \text{m1} = \text{mil-set} \{ \text{i} | \text{is-EXPR}\text{arg-class}(\text{id},\text{i1}) \& \text{is-array}\text{elem}(\text{i},\text{list}) \} \]
\[ \text{dim-list} = \text{dim}\text{elem}(\text{m1}, \text{list}) \]
\[ \text{i1} = (\text{id} \epsilon \{ \text{mk-id(MAX)}, \text{mk-id(MIN)} \} \rightarrow 1, \]
\[ \top \rightarrow 1) \]

\[ \text{Ref.:} \quad \text{arg-class} \ 12-14(33) \]
\[ \text{dim} \ 8-25(66) \]
(21)  \textit{aggr-builtin-ccntd-1}(id, list, at) =

\begin{align*}
&\left(\exists i\right) \left( \text{is-EXPND\textbullet arg\textbullet class}(id, i_1) \land \text{is-struct\textbullet elem}(i, \text{list}) \right) \land \\
&\neg \left(\exists i\right) \left( \text{is-EXPND\textbullet arg\textbullet class}(id, i_1) \land \text{is-array\textbullet elem}(i, \text{list}) \right) \\
&\text{Et} \left( \text{is-EXPND\textbullet arg\textbullet class}(i, i_1) \land \text{is-struct\textbullet elem}(i, \text{list}) = i = 1 \right) \\
&\text{length}\cdot\text{elem}(i, \text{list}) = \text{length}_1 \rightarrow \\
&\text{length}_1 || \text{list} \bigcup_{j=1}^{\text{length}(\text{list})} \text{struct•elem}(i, \text{list}) \\
&\text{T} \rightarrow \text{error}
\end{align*}

where:

\begin{align*}
&n_1 = \text{min\textbullet set}\{i \mid \text{is-EXPND\textbullet arg\textbullet class}(id, i_1) \land \text{is-struct\textbullet elem}(i, \text{list})\} \\
&\text{length}_2 = \text{length}\cdot\text{elem}(n_1, \text{list}) \\
&i_1 = (id \in \{\text{mk\textbullet id}(\text{MAX}), \text{mk\textbullet id}(\text{MIN})\} \rightarrow 1, \\
&\text{T} \rightarrow i)
\end{align*}

Ref.: arg-class 12-14(33)

(22)  \textit{array•elem}(id, i, el) =

\begin{align*}
&\text{is-EXPND\textbullet arg\textbullet class}(id, i) \land \text{is-array}(el) \rightarrow \text{s•elem}(el) \\
&\text{T} \rightarrow \text{el}
\end{align*}

for: is-aggr\textbullet (el)

Ref.: arg-class 12-14(33)

(23)  \textit{struct•elem}(id, i, j, el) =

\begin{align*}
&\text{is-EXPND\textbullet arg\textbullet class}(id, i) \land \text{is-struct}(el) \rightarrow \text{s•aggr•elem}(j, el) \\
&\text{T} \rightarrow \text{el}
\end{align*}

for: is-aggr\textbullet (el)

Ref.: arg-class 12-14(33)

(24)  \textit{aggr-builtin-2}(id, list, at) =

\begin{align*}
&\text{aggr•scalar•da-builtin}(id, \text{LIST} \bigcup_{i=1}^{\text{length}(\text{list})} \text{adjust•elem}(id, i_1, \text{elem}(i, \text{list}), at))
\end{align*}

where:

\begin{align*}
&i_1 = (id \in \{\text{mk\textbullet id}(\text{MIN}), \text{mk\textbullet id}(\text{MAX})\} \rightarrow 1, \\
&\text{T} \rightarrow i)
\end{align*}
12.4 BUILT-IN FUNCTIONS AS VALUES OF ENTRY VARIABLES

In some instances in PL/I (passing of arguments to an entry parameter) it is possible to assign a built-in function as value to an entry variable. This is possible only for the float generic built-in functions, characterized by the predicate is-float-generic-built-in. In this case the generic selection of the generic built-in function is performed before the assignment according to the parameter descriptors of the entry variable, only the selected member of the built-in function is assigned, and a later call of the entry variable is possible only if the argument of the call match the parameter descriptors of the selected member.

This is realized in the formal definition by constructing a denotation for the selected member of a built-in function, which consists of the built-in identifier and the data attributes of the parameters allowed in a call. This denotation is constructed by the instruction builtin-sel and entered under a newly created unique name n into the denotation directory NW immediately before the assignment (cf. 8.2.2). The unique name n is assigned as value to the entry variable.

When an entry variable is called, the instruction int-call-1 distinguishes whether the value of the entry variable leads to a proper entry denotation or to the denotation of a built-in function (cf. 6.2.1). In the entry case, the instruction call-proc is executed which performs the proper procedure call (cf. 6.2.2). In the built-in case, the instruction call-built-in is executed which has to simulate all actions which are usually performed by the instruction call-proc within the called procedure (in particular by the corresponding return statement, cf. 6.2.4), namely: Testing of the passed arguments against the types of the parameters required, evaluation of the return expression (i.e. the value of the built-in function), allocation and assignment of a dummy for the returned value and freeing of the storage of the passed dummy arguments.
Metavariables

id is-id

(26) is-flat-generic-builtin(id) =

id ∈

{id, mk-id(ACOS), mk-id(ASIN), mk-id(ATAN), mk-id(ATAND), mk-id(ATANF), mk-id(COS),
mk-id(COSD), mk-id(COSH), mk-id(EXP), mk-id(EXP), mk-id(LOG),
mk-id(LOGF10), mk-id(LOGF2), mk-id(SIN), mk-id(SIND), mk-id(SIND), mk-id(SQRT),
mk-id(TAN), mk-id(TAND))

(27) builtin-sel(id, attr-list, n) =
builtin-match(id, attr-list) ->

PASS:n
g→ uθ (p Ug: n: θ (s-id:id), <s-da-list:da-list> ))

T → error

where:

length(attr-list)
da-list = list s-da•s-aggr•elea(i, attr-list)

for:is-descr-list(attr-list), is-n(b)

(28) builtin-match(id, attr-list) =

~is-V(attr-list) & (lg 1 = 1 ∨ lg 1 = 2 ∧ id ∈ {mk-id(ATAN), mk-id(ATAND)}) ∨

lg 1 ≤ it is-Vs•s-scale•s-da•s-aggr•elea(i, attr-list)

where:

lg 1 = length(attr-list)

for:is-descr-list(attr-list)

(29) call-builtin(id, da-list, argl, b) =

length(argl) = length(da-list) ->

allocate-assgn(b, cp);

call-eval-builtin-1(id, opl);

{(elem(i)) (opl)}; install-builtin-arg(elem(i, argl)), elem(i, da-list)) 1

1 ≤ i ≤ length(argl)

T → error

for:is-da-list(da-list), is-n(b)

cont'd
12.5 TABLE OF BUILT-IN FUNCTIONS

12.5.1 EXPLANATION

The following table (cf. 5.2) lists all PL/I built-in functions and gives a survey of their characteristic properties: admissible numbers and kinds of arguments and data attributes of the result values. Furthermore, it lists the instructions, defined in section 12.6 which after suitable preparation of arguments evaluate the individual built-in functions.

It will be spoken of one table contained in 5.2; the tables occurring there are to be looked upon as one single table: The table on a right hand page is the continuation of the table on the left hand page, this "compound table" is continued by the "table" on the next two pages.

To remain within the range of definitional tools used throughout the present document (cf. chapter 1), the table may be read as a definition of the functions link(id), arg-class(id,i), arg-edat(id,i), arg-test(id,i), da-built-in(id,i) and the instruction eval-built-in(id,list) by means of conditional expressions. These functions and the instruction are used in sections 12.1, 12.2, and 12.3. The first two columns, the "argument columns" id and i, determine the propositions of the conditional expressions while the other columns, the "result columns", contain the expressions or instructions defining the mentioned functions or instructions, respectively. Thereby, of course, the argument column i has to be neglected for link(id), da-built-in(id,list), and eval-built-in(id,list) since they do not depend on i.

The last line of the table, having a T in the argument columns, defines some of the functions for all cases not listed before. It is used to guarantee handling of errors in accordance with section 1.3 of the present document.

So, for instance, the definition of the function arg-type reads:

arg-type(id,i) =
  id = mk-id(BIT) & i=1  →  OP
  id = mk-id(BIT) & i=2  →  INTG
  id = mk-id(BOOL) & i=1  →  OP
  ...
  id = mk-id(FIXED) & i=3  →  INTG

12 12. BUILT-IN FUNCTIONS AND PSLODC-VARIAELES
id = mk-id(LINENO) & i=1 → OP
T → 0

or, the definition of the instruction \texttt{eval-built-in} reads:

\begin{verbatim}

\texttt{eval-built-in} =

id = mk-id(BIF) → eval-bit(op1, k2)
id = mk-id(BGCL) → eval-bgcl(op1, op2, op3)

id = mk-id(FIXED) → eval-fixed(op1, k2, k3)

id = mk-id(LINENO) → eval-lineno(op1)
id = mk-id(TIME) → eval-time
\end{verbatim}

For more details see the following descriptions of the single columns of the table.

\textbf{12.5.1.1 The argument columns id and i}

The first column contains the built-in function identifiers in their concrete representation. To use these identifiers in the formal definition, one has to apply the function \texttt{mk-id} to them in order to have objects satisfying the predicate \texttt{is-id}.

The first column of the table is also used to define the predicate \texttt{is-built-in} where the expression \texttt{is-builtin(id)} is true, if \texttt{id} is a built-in function identifier:

\begin{equation}
\texttt{is-builtin(id)} = \texttt{id} \in \texttt{BIF-ID-SET}
\end{equation}

\text{for:} \texttt{is-id(id) \& is-id-set(BIF-ID-SET)}

Note: \texttt{BIF-ID-SET} is the set of all identifiers that can be produced by applying the function \texttt{mk-id} to an entry in the first column of the table with the exception of the last entry, namely T:

\begin{equation}
\texttt{BIF-ID-SET} = \{\texttt{mk-id(BIF), mk-id BOOL}, \ldots, \texttt{mk-id(LINENO), mk-id(TIME)}\}
\end{equation}

The second column contains integer values denoting for each single built-in function the different possible argument positions. They are used, as described in the above example, as argument for the functions \texttt{arg-class}, \texttt{arg-type}, \texttt{arg-edn}, and \texttt{arg-test} which describe properties of the single arguments of the different built-in functions. Since the built-in functions MAX and MIN have an indefinite number of arguments of the same kind, for these two functions only the entry with \texttt{i=1} is contained in the table.
12.5.1.2  Number of arguments of built-in functions

Several built-in functions may have a variable number of arguments. In these cases the last argument or arguments are optional. The maximum length of an argument list for each built-in function is determined by the number of entries in the column i, except for the two built-in functions MAX and MIN for which no maximum length exists. The minimum length of the argument list (and thereby the information which arguments are optional) is given by the third column of the table.

This column defines the function

\[(32) \text{lniz}(id) = \text{for:is-id}(id)\]

Note: The range of this function is the set of integer values \([0, 1, 3, 4]\). The default value of this function for all non-built-in identifiers is 1. This value guarantees that erroneous programs are rejected by the instructions \text{builtin-arg-list} or \text{builtin-arg} in section 12.2.

12.5.1.3  The classes of arguments of built-in functions

The possible arguments of built-in functions may be categorized into five classes which are characterized by the four elementary objects \text{EXPND}, \text{CONST}, \text{AGGR}, \text{ARR}, \text{IRREL}. This categorization considers the way how the different arguments of built-in functions influence the syntactic expansion of aggregate expressions to scalar expressions (cf. 8.1.2 and especially 12.1) and the aggregate attributes of the result of an expression (cf. 8.2.5 and especially 12.3):

\text{EXPND}. The arguments belonging to this class may be aggregate expressions. They influence the syntactic expansion of expressions in the same way as if they occurred as direct subexpressions of them (cf. 8.1). In the final scalar expressions to be evaluated, they occur only as scalar arguments. The data attributes of these scalar arguments, after appropriate conversion (cf 12.5.1.5) influence the data attributes of the result of the built-in function.

\text{CONST}. For arguments belonging to this class the value of the argument influences the data attributes of the result. Actually, such arguments are restricted to be integer constants and a test for this is made by \text{builtin-arg-intg} in the case arg-class(id,i) = \text{CONST} by the predicate test-intg: as can be seen from the table 12.5.1.4, \text{arg-type} is INTG or \text{s-INTG}, whenever arg-class is \text{CONST} (cf. 12.5.1.4, 12.5.1.6)

\text{AGGR}. The arguments belonging to this class may be aggregate expressions (like aggregate arguments of programmed function procedures). The data attributes of their scalar components influence - after appropriate conversion - the data attributes of the result.

\text{ARR}. The same holds as for \text{AGGR} with the restriction that the arguments must not be structures.

\text{IRREL}. The arguments belonging to this class are irrelevant both for syntactic expansion and for the determination of the data attributes of the result.

The classification into these four classes is given by the fourth column of the table. It defines the function

\[(33) \text{arg-class}(id,i) = \text{for:is-id}(i),is-intg-val(i)\]

cont'd
Ref.: is-intg-val 9-3(5)

Note: The range of this function is the set of five elementary objects (EXPR, CONST, AGGR, RES, IRREL) and the null object & as default values for erroneous cases.

12.5.1.4 The types of arguments of built-in functions

On evaluation of a built-in function reference, the arguments are evaluated by a general mechanism (cf. 12.2) and passed to the individual instruction for the built-in function. Each evaluated argument is one of the following six types of objects: an operand, an operand list (the scalar operands resulting from an aggregate expression), an integer value, evaluated aggregate attributes of an aggregate expression, a generation, the uncharged text of the argument expression itself.

Which of these six types of arguments is passed is given for each individual argument by the fifth column of the table. It defines the function

(34) \[ \text{arg-type(id,i)} = \]

for:is-id(i),is-intg-val(i)

Ref.: is-intg-val 9-3(5)

Note: The range of this function is the set of the six elementary objects (OF, CEL, INTG, EVA, GEN, TEXT) and the null object & as default value for erroneous cases.

12.5.1.5 Conversion of arguments of built-in functions

In those cases where evaluated operands or operand lists of the arguments are passed to the instruction of a built-in function, before passing these operands are converted to specified (incomplete) aggregate attributes. The target aggregate attributes for this conversion are found by applying the function aggr-scalar to the incomplete evaluated data attribute (cf 9.2.1) contained in the sixth column of the table. An asterisk denotes that no conversion takes place.

This column defines the function

(35) \[ \text{arg-eda(id,i)} = \]

for:is-id(i),is-intg-val(i)

Ref.: is-intg-val 9-3(5)

Note: The range of this function is the set of objects {* , PIR, OFFSET, FILE, EVENT, TASK, STRING-EDA, BYT-EDA, bit-eda(4), AR-EDA, PLT-EDA}, cf. 9.2.1.

12.5.1.6 Testing of arguments of built-in functions

During the evaluation of arguments of built-in functions by the general mechanism described in section 12.2, some general tests are performed. These tests supplement the tests performed automatically by the instructions convert-1, eval-exp-2, eval-req-gen, etc. and the special tests performed by the individual instructions of the built-in functions.

These tests are specified by the following elementary objects contained in the seventh column of the table:
REAL  The converted argument has to be of real mode.
CPLX  The converted argument has to be of complex mode.
ARE  The argument has to be an array expression.
LIN-ARE  The argument has to be a one dimensional array expression.
LIN  The argument has to be a one dimensional array expression or a scalar expression.
INTG  The argument has to be an unsigned integer constant.
S-INTG  The argument has to be a possibly signed integer constant.
T  The argument may be any expression (the test performed has no influence).
*  No test is performed during the argument evaluation.

This column defines the function

(36)  \[ \text{arg-test}(id,i) = \]

\[ \text{for:is-id}(i), \text{is-intg-val}(i) \]

Ref.:  is-intg-val 9-3(5)

Note:  The range of this function is the set of elementary objects \{REAL, CPLX, ARE, LIN-ARE, LIN, INTG, S-INTG, T, *\}.

12.5.1.7 Data attributes of built-in functions

The last but one column of the table gives for each built-in function an expression which determines its data attributes. This expression depends on the data attributes or values of the arguments of the built-in function (cf. 12.5.1.3).

This column defines the function

(37)  \[ \text{da-builtin}(id,list) = \]

\[ \text{for:is-id}(id), \text{is-builtin-arg-descr-list}(list) \&
\quad (\text{is-builtin-arg-descr } = \text{is-da } \lor \text{is-aggr } \lor \text{is-intg-val } \lor \text{is-*}) \]

Ref.:  is-intg-val 9-3(5)

Note:  The range of this function consists of scalar data attributes, i.e. objects satisfying the predicate is-da. Its argument list consists of the descriptors of the single arguments of the built-in function: for classes EXPR and ARE generally the scalar data attributes, for AGGR a general aggregate attribute, for CONST the integer value, and for class IRREL the object * (cf. 12.5.1.3).
12.5.1.8 Evaluation of the individual built-in functions

The last column of the table lists the instructions which after evaluation of the arguments evaluate the values of the individual built-in functions (cf. 12.2). These instructions are listed in section 12.6, see the references under the table. Arguments of these instructions are the evaluated arguments of the built-in function reference (cf. 12.5.1.4). For the built-in functions MAX and MIN the instructions have one argument which is the evaluated list of arguments. Each of these instructions returns an operand representing the result value with an aggregate attribute having a da-component defined by the function da-built-in (with exception of string-lengths and similar entities) and a density-component which is the default density corresponding to the result of da-built-in.

The last column defines the instruction

(38) \texttt{eval-built-in-1}(id,bal) =

\begin{align*}
\text{for:} & \text{id}(id), \text{is-built-in-arg-list(bal)} & & \\
& (\text{is-built-in-arg} = \text{is-op} \lor \text{is-op-list} \lor \text{is-intg-val} \lor \text{is-eval} \lor \text{is-gen} \lor \text{is-expr}) & & \\
\text{Ref.:} & \text{is-op 9-9(34)} & & \\
& \text{is-intg-val 9-3(5)} & & \\
& \text{is-gen 3-14(30)} & & \\
\end{align*}

Note: The instruction returns an operand, i.e. an object satisfying the predicate is-op.

12.5.2 THE TABLE

Abbreviations

- \texttt{op_n} = \texttt{elem(n,list)}, for \( n = 1, 2, 3 \) used if the \( n \)-th element of list is an operand
- \texttt{op_l_n} = \texttt{elem(n,list)}, for \( n = 1, 2 \) used if the \( n \)-th element of list is an operand list
- \texttt{k_n} = \texttt{elem(n,list)}, for \( n = 1, 2, 3, 4 \) used if the \( n \)-th element of list is an integer value
- \texttt{gen_n} = \texttt{elem(n,list)}, for \( n = 1, 2 \) used if the \( n \)-th element of list is a generation
- \texttt{t_1} = \texttt{elem(1,list)} used if the first element of list is the text of an expression
- \texttt{op_l_0} = \texttt{list} used for MAX and MIN if the list is an operand list
- \texttt{da_n} = \texttt{elem(n,list)} used if the \( n \)-th element of list is a scalar data attribute
- \texttt{da-list_0} = \texttt{list} used for MAX and MIN if the elements of list consist of scalar data attributes
- \texttt{eva_1} = \texttt{elem(1,list)} used if the first element of list consists of a evaluated aggregate attribute
aggr₁ = elem(1,list)  

used if the first element of list consists  
of an aggregate attribute  

The table is split up into several parts corresponding to the specific groups  
of built-in functions. To cope with the preceding descriptions, these parts  
should be looked upon as one continuous table.
### String-Handling Built-in Functions

<table>
<thead>
<tr>
<th>id</th>
<th>i1min(id)</th>
<th>arg-class(id,i)</th>
<th>arg-type(id,i)</th>
<th>arg-eda(id,i)</th>
<th>arg-test(id,i)</th>
</tr>
</thead>
<tbody>
<tr>
<td>BIT</td>
<td>1</td>
<td>EXPND</td>
<td>OP</td>
<td>STRING-EDA</td>
<td>T</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>EXPND</td>
<td>OP</td>
<td>STRING-EDA</td>
<td>T</td>
</tr>
<tr>
<td>BIG</td>
<td>1</td>
<td>EXPND</td>
<td>OP</td>
<td>STRING-EDA</td>
<td>T</td>
</tr>
<tr>
<td>LOW</td>
<td>1</td>
<td>EXPND</td>
<td>OP</td>
<td>STRING-EDA</td>
<td>T</td>
</tr>
<tr>
<td>INDEX</td>
<td>1</td>
<td>EXPND</td>
<td>OP</td>
<td>STRING-EDA</td>
<td>T</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>EXPND</td>
<td>OP</td>
<td>STRING-EDA</td>
<td>T</td>
</tr>
<tr>
<td>LENGTH</td>
<td>1</td>
<td>EXPND</td>
<td>OP</td>
<td>STRING-EDA</td>
<td>T</td>
</tr>
<tr>
<td>LOW</td>
<td>1</td>
<td>EXPND</td>
<td>OP</td>
<td>STRING-EDA</td>
<td>T</td>
</tr>
<tr>
<td>REPEAT</td>
<td>1</td>
<td>EXPND</td>
<td>OP</td>
<td>STRING-EDA</td>
<td>T</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>EXPND</td>
<td>OP</td>
<td>STRING-EDA</td>
<td>T</td>
</tr>
<tr>
<td>STRING</td>
<td>1</td>
<td>AGGR</td>
<td>OP</td>
<td>STRING-EDA</td>
<td>T</td>
</tr>
<tr>
<td>SUBSTR</td>
<td>1</td>
<td>EXPND</td>
<td>TEXT</td>
<td>STRING-EDA</td>
<td>T</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>EXPND</td>
<td>INTG</td>
<td>STRING-EDA</td>
<td>T</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>EXPND</td>
<td>INTG</td>
<td>STRING-EDA</td>
<td>T</td>
</tr>
<tr>
<td>TRANSIT</td>
<td>1</td>
<td>STYND</td>
<td>OP</td>
<td>STRING-EDA</td>
<td>T</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>EXPND</td>
<td>OP</td>
<td>STRING-EDA</td>
<td>T</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>EXPND</td>
<td>OP</td>
<td>STRING-EDA</td>
<td>T</td>
</tr>
<tr>
<td>UNSTEP</td>
<td>1</td>
<td>EXPND</td>
<td>GEN</td>
<td>STRING-EDA</td>
<td>T</td>
</tr>
<tr>
<td>VEDITY</td>
<td>1</td>
<td>EXPND</td>
<td>OP</td>
<td>STRING-EDA</td>
<td>T</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>EXPND</td>
<td>OP</td>
<td>STRING-EDA</td>
<td>T</td>
</tr>
<tr>
<td>id</td>
<td>da-builtin(id, list)</td>
<td>eval-builtitp-1(id, list)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>------</td>
<td>----------------------</td>
<td>---------------------------</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BIT</td>
<td>complete-tg(EIT-EDA, da₁)</td>
<td>eval-bit(op₁, k₁)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BOOL</td>
<td>da-bool(da₁, da₂)</td>
<td>eval-bool(op₁, op₂, op₃)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CHAR</td>
<td>complete-tg(CHAR-EDA, da₁)</td>
<td>eval-char(op₁, k₁)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>HIGH</td>
<td>CHAR-EDA</td>
<td>eval-high(k₁)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>INDEX</td>
<td>BINTG-EDA</td>
<td>eval-index(op₁, op₂)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LENGTH</td>
<td>BINTG-EDA</td>
<td>eval-length(op₁)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LOW</td>
<td>CHAR-EDA</td>
<td>eval-low(k₁)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>REPEAT</td>
<td>μ(da₁; s-length<em>da:</em>&gt;)</td>
<td>eval-repeat(op₁, k₁)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>STRING</td>
<td>da-string(arg₁)</td>
<td>eval-string(op₁)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SUBSTR</td>
<td>μ(da₁; s-varying:VAR&gt;)</td>
<td>eval-substr(t₁, k₂, k₃)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TRANSLATE</td>
<td>da₁</td>
<td>eval-translate(op₁, op₂, op₃)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>UNSPEC</td>
<td>BIT-EDA</td>
<td>eval-unspec(gen₁)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>VERIFY</td>
<td>BINTG-EDA</td>
<td>eval-verify(op₁, op₂)</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Ref.:  
- eval-bit 12-36(47)  
- eval-bool 12-37(48)  
- eval-char 12-37(52)  
- eval-high 12-38(53)  
- eval-index 12-38(54)  
- eval-length 12-38(56)  
- eval-low 12-38(57)  
- eval-repeat 12-39(58)  
- eval-string 12-39(60)  
- eval-substr 12-39(62)  
- eval-translate 12-41(66)  
- eval-unspec 12-42(70)  
- eval-verify 12-43(73)
## Arithmetic Generic Built-in Functions

<table>
<thead>
<tr>
<th>id</th>
<th>ilmin(id)</th>
<th>arg-class(id,i)</th>
<th>arg-type(id,i)</th>
<th>arg-edata(id,i)</th>
<th>arg-test(id,i)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>ABS</strong></td>
<td>1</td>
<td>EXPED</td>
<td>OP</td>
<td>AR-EDA</td>
<td>T</td>
</tr>
<tr>
<td><strong>ADD</strong></td>
<td>1</td>
<td>EXPED</td>
<td>OP</td>
<td>AR-EDA</td>
<td>T</td>
</tr>
<tr>
<td>2</td>
<td>EXPED</td>
<td>OP</td>
<td>AR-EDA</td>
<td>T</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>CONST</td>
<td>INTG</td>
<td>*</td>
<td>INTG</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>CONST</td>
<td>INTG</td>
<td>*</td>
<td>S-INTG</td>
<td></td>
</tr>
<tr>
<td><strong>BEIN</strong></td>
<td>1</td>
<td>EXPED</td>
<td>OP</td>
<td>AR-EDA</td>
<td>T</td>
</tr>
<tr>
<td>2</td>
<td>EXPED</td>
<td>OP</td>
<td>AR-EDA</td>
<td>T</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>EXPED</td>
<td>OP</td>
<td>AR-EDA</td>
<td>T</td>
<td></td>
</tr>
<tr>
<td><strong>CEIL</strong></td>
<td>1</td>
<td>EXPED</td>
<td>OP</td>
<td>AR-EDA</td>
<td>REAL</td>
</tr>
<tr>
<td><strong>CEIL</strong></td>
<td>2</td>
<td>EXPED</td>
<td>OP</td>
<td>AR-EDA</td>
<td>REAL</td>
</tr>
<tr>
<td><strong>CONJG</strong></td>
<td>1</td>
<td>EXPED</td>
<td>OP</td>
<td>AR-EDA</td>
<td>CFLX</td>
</tr>
<tr>
<td><strong>DECIMAL</strong></td>
<td>1</td>
<td>EXPED</td>
<td>OP</td>
<td>AR-EDA</td>
<td>T</td>
</tr>
<tr>
<td><strong>DECIMAL</strong></td>
<td>2</td>
<td>EXPED</td>
<td>OP</td>
<td>AR-EDA</td>
<td>T</td>
</tr>
<tr>
<td>3</td>
<td>EXPED</td>
<td>OP</td>
<td>AR-EDA</td>
<td>T</td>
<td></td>
</tr>
<tr>
<td><strong>DIVIDE</strong></td>
<td>1</td>
<td>EXPED</td>
<td>OP</td>
<td>AR-EDA</td>
<td>T</td>
</tr>
<tr>
<td>2</td>
<td>EXPED</td>
<td>OP</td>
<td>AR-EDA</td>
<td>T</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>EXPED</td>
<td>OP</td>
<td>AR-EDA</td>
<td>T</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>EXPED</td>
<td>OP</td>
<td>AR-EDA</td>
<td>T</td>
<td></td>
</tr>
<tr>
<td><strong>FIXED</strong></td>
<td>1</td>
<td>EXPED</td>
<td>OP</td>
<td>AR-EDA</td>
<td>T</td>
</tr>
<tr>
<td>2</td>
<td>EXPED</td>
<td>OP</td>
<td>AR-EDA</td>
<td>T</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>EXPED</td>
<td>OP</td>
<td>AR-EDA</td>
<td>T</td>
<td></td>
</tr>
<tr>
<td><strong>FLOAT</strong></td>
<td>1</td>
<td>EXPED</td>
<td>OP</td>
<td>AR-EDA</td>
<td>T</td>
</tr>
<tr>
<td>2</td>
<td>EXPED</td>
<td>OP</td>
<td>AR-EDA</td>
<td>T</td>
<td></td>
</tr>
<tr>
<td>cont'd...</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### Built-In Functions and Pseudo-Variables

<table>
<thead>
<tr>
<th>id</th>
<th>da-builtin(id,list)</th>
<th>eval-builtin-1(id,list)</th>
</tr>
</thead>
<tbody>
<tr>
<td>ABS</td>
<td>da-abs(da_1)</td>
<td>eval-abs(op_1)</td>
</tr>
<tr>
<td>ADD</td>
<td>da-infix-1(da_1,da_2,k_3,k_4,ADD)</td>
<td>eval-add(op_1,op_2,k_3,k_4)</td>
</tr>
<tr>
<td>BIN</td>
<td>da-bin(da_1,k_2,k_3)</td>
<td>eval-bin(op_1,k_2,k_3)</td>
</tr>
<tr>
<td>CEIL</td>
<td>da-trunc(da_1)</td>
<td>eval-ceil(op_1)</td>
</tr>
<tr>
<td>CPLX</td>
<td>new(CPLX,pref(da_1,da_2))</td>
<td>eval-cplx(op_1,op_2)</td>
</tr>
<tr>
<td>CONJG</td>
<td>da_1</td>
<td>eval-conjg(op_1)</td>
</tr>
<tr>
<td>DEC</td>
<td>da-dec(da_1,k_2,k_3)</td>
<td>eval-dec(op_1,k_2,k_3)</td>
</tr>
<tr>
<td>DIV</td>
<td>da-infix-1(da_1,da_2,k_3,k_4,DIV)</td>
<td>eval-divide(op_1,op_2,k_3,k_4)</td>
</tr>
<tr>
<td>FIXED</td>
<td>da-fixed(da_1,k_2,k_3)</td>
<td>eval-fixed(op_1,k_2,k_3)</td>
</tr>
<tr>
<td>FLOAT</td>
<td>da-float(da_1,k_3)</td>
<td>eval-float(op_1,k_2)</td>
</tr>
</tbody>
</table>

**Ref.:**
- eval-abs 12-44 (77)
- eval-add 12-45 (86)
- eval-bin 12-45 (81)
- eval-ceil 12-45 (83)
- eval-cplx 12-46 (84)
- eval-conjg 12-46 (86)
- eval-dec 12-46 (87)
- eval-divide 12-46 (89)
- eval-fixed 12-46 (95)
- eval-float 12-47 (92)

Cont'd.
<table>
<thead>
<tr>
<th>id</th>
<th>ilmin(id)</th>
<th>arg-class(id,i)</th>
<th>arg-type(id,i)</th>
<th>arg-eda(id,i)</th>
<th>arg-test(id,i)</th>
</tr>
</thead>
<tbody>
<tr>
<td>FLOOR</td>
<td>1</td>
<td>EXPND</td>
<td>OP</td>
<td>AR-EDA</td>
<td>REAL</td>
</tr>
<tr>
<td>IMAG</td>
<td>1</td>
<td>EXPND</td>
<td>OP</td>
<td>AR-EDA</td>
<td>CPLX</td>
</tr>
<tr>
<td>MIN</td>
<td>1</td>
<td>2 EXPND</td>
<td>OP</td>
<td>AR-EDA</td>
<td>T</td>
</tr>
<tr>
<td>MOD</td>
<td>1</td>
<td>2 EXPND</td>
<td>OP</td>
<td>AR-EDA</td>
<td>REAL</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>EXPND</td>
<td>OP</td>
<td>AR-EDA</td>
<td>REAL</td>
</tr>
<tr>
<td>MULTPLY</td>
<td>1</td>
<td>3 EXPND</td>
<td>OP</td>
<td>AR-EDA</td>
<td>T</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>EXPND</td>
<td>OP</td>
<td>AR-EDA</td>
<td>T</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>CONST</td>
<td>INTG</td>
<td>*</td>
<td>S-INTG</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>3 EXPND</td>
<td>OP</td>
<td>AR-EDA</td>
<td>T</td>
</tr>
<tr>
<td>PRECISION</td>
<td>2</td>
<td>2 EXPND</td>
<td>OP</td>
<td>AR-EDA</td>
<td>S-INTG</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>CONST</td>
<td>INTG</td>
<td>*</td>
<td>S-INTG</td>
</tr>
<tr>
<td>REAL</td>
<td>1</td>
<td>EXPND</td>
<td>OP</td>
<td>AR-EDA</td>
<td>CPLX</td>
</tr>
<tr>
<td>ROUND</td>
<td>1</td>
<td>2 EXPND</td>
<td>TEXT</td>
<td>*</td>
<td></td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>CONST</td>
<td>INTG</td>
<td>*</td>
<td>S-INTG</td>
</tr>
<tr>
<td>SIGN</td>
<td>1</td>
<td>1 EXPND</td>
<td>OP</td>
<td>AR-IDA</td>
<td>REAL</td>
</tr>
<tr>
<td>TRUNC</td>
<td>1</td>
<td>1 EXPND</td>
<td>OP</td>
<td>AR-EDA</td>
<td>REAL</td>
</tr>
<tr>
<td>id</td>
<td>eval-builtin(id, list)</td>
<td>eval-builtin-1(id, list)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>------------</td>
<td>------------------------</td>
<td>-------------------------</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FLOOR</td>
<td>da-trunc(da_1)</td>
<td>eval-floor(op_1)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IMAG</td>
<td>new(REAL, da_1)</td>
<td>eval-imag(op_1)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MAX</td>
<td>pref-1(Q, da-list_0)</td>
<td>eval-max(op_1_o)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MIN</td>
<td>pref-1(Q, da-list_0)</td>
<td>eval-min(op_1_o)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MOD</td>
<td>da-mod(da_1, da_2)</td>
<td>eval-mod(op_1, op_2)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BUILTIN</td>
<td>da-infix-1(da_1, da_2, k_3, k_4, MULT)</td>
<td>eval-multiply(op_1, op_2, k_3, k_4)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PRECISION</td>
<td>da-prec(da_1, k_2, k_3)</td>
<td>eval-prec(op_1, k_2, k_3)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FBEC</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>REAL</td>
<td>new(REAL, da_1)</td>
<td>eval-real(op_1)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ROUND</td>
<td>da-round(da_1)</td>
<td>eval-round(t_1, k_2)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SIGN</td>
<td>BINTG-EDA</td>
<td>eval-sign(op_1)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TRUNC</td>
<td>da-trunc(da_1)</td>
<td>eval-trunc(op_1)</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Ref.:
- eval-floor 12-47(04)
- eval-imag 12-47(95)
- eval-max 12-48(96)
- eval-min 12-48(97)
- eval-mod 12-48(98)
- eval-multiply 12-49(101)
- eval-prec 12-49(102)
- eval-real 12-49(104)
- eval-round 12-50(105)
- eval-sign 12-51(109)
- eval-trunc 12-51(110)
### Mathematical Generic Built-In Functions

<table>
<thead>
<tr>
<th>id</th>
<th>idmin(id)</th>
<th>arg-class(id,i)</th>
<th>arg-type(id,i)</th>
<th>arg-eda(id,i)</th>
<th>arg-test(id,i)</th>
</tr>
</thead>
<tbody>
<tr>
<td>ASIN</td>
<td>1</td>
<td>EXPND</td>
<td>OP</td>
<td>FLT-EDA</td>
<td>T</td>
</tr>
<tr>
<td>ATAN</td>
<td>1</td>
<td>EXPND</td>
<td>OP</td>
<td>FLT-EDA</td>
<td>T</td>
</tr>
<tr>
<td>ATANH</td>
<td>1</td>
<td>EXPND</td>
<td>OP</td>
<td>FLT-EDA</td>
<td>REAL</td>
</tr>
<tr>
<td>COS</td>
<td>1</td>
<td>EXPND</td>
<td>OP</td>
<td>FLT-EDA</td>
<td>T</td>
</tr>
<tr>
<td>COSD</td>
<td>1</td>
<td>EXPND</td>
<td>OP</td>
<td>FLT-EDA</td>
<td>REAL</td>
</tr>
<tr>
<td>CSCH</td>
<td>1</td>
<td>EXPND</td>
<td>OP</td>
<td>FLT-EDA</td>
<td>T</td>
</tr>
<tr>
<td>ERF</td>
<td>1</td>
<td>EXPND</td>
<td>OP</td>
<td>FLT-EDA</td>
<td>REAL</td>
</tr>
<tr>
<td>ENF</td>
<td>1</td>
<td>EXPND</td>
<td>OP</td>
<td>FLT-EDA</td>
<td>T</td>
</tr>
<tr>
<td>ENF2</td>
<td>1</td>
<td>EXPND</td>
<td>OP</td>
<td>FLT-EDA</td>
<td>T</td>
</tr>
<tr>
<td>ERF</td>
<td>1</td>
<td>EXPND</td>
<td>OP</td>
<td>FLT-EDA</td>
<td>T</td>
</tr>
<tr>
<td>LOG</td>
<td>1</td>
<td>EXPND</td>
<td>OP</td>
<td>FLT-EDA</td>
<td>T</td>
</tr>
<tr>
<td>LOG10</td>
<td>1</td>
<td>EXPND</td>
<td>OP</td>
<td>FLT-EDA</td>
<td>T</td>
</tr>
<tr>
<td>LOG2</td>
<td>1</td>
<td>EXPND</td>
<td>OP</td>
<td>FLT-EDA</td>
<td>REAL</td>
</tr>
<tr>
<td>SIN</td>
<td>1</td>
<td>EXPND</td>
<td>OP</td>
<td>FLT-EDA</td>
<td>T</td>
</tr>
<tr>
<td>SIND</td>
<td>1</td>
<td>EXPND</td>
<td>OP</td>
<td>FLT-EDA</td>
<td>REAL</td>
</tr>
<tr>
<td>SINH</td>
<td>1</td>
<td>EXPND</td>
<td>OP</td>
<td>FLT-EDA</td>
<td>T</td>
</tr>
<tr>
<td>SQRT</td>
<td>1</td>
<td>EXPND</td>
<td>OP</td>
<td>FLT-EDA</td>
<td>T</td>
</tr>
<tr>
<td>TAN</td>
<td>1</td>
<td>EXPND</td>
<td>OP</td>
<td>FLT-EDA</td>
<td>T</td>
</tr>
<tr>
<td>TAND</td>
<td>1</td>
<td>EXPND</td>
<td>OP</td>
<td>FLT-EDA</td>
<td>T</td>
</tr>
<tr>
<td>TANH</td>
<td>1</td>
<td>EXPND</td>
<td>OP</td>
<td>FLT-EDA</td>
<td>T</td>
</tr>
</tbody>
</table>
### Abstract Syntax and Interpretation of PL/I

<table>
<thead>
<tr>
<th>id</th>
<th>da-builtins(id,list)</th>
<th>eval-builtins(id,list)</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACOS</td>
<td>da_1</td>
<td>eval-acos(op_1)</td>
</tr>
<tr>
<td>ASIN</td>
<td>da_1</td>
<td>eval-asin(op_1)</td>
</tr>
<tr>
<td>ATAN</td>
<td>nref(da_1,da_2)</td>
<td>eval-atan(op_1,op_2)</td>
</tr>
<tr>
<td>ATANH</td>
<td>da_1</td>
<td>eval-atanh(op_1)</td>
</tr>
<tr>
<td>COS</td>
<td>da_1</td>
<td>math-bif(mk-id(COS),op_1)</td>
</tr>
<tr>
<td>COSD</td>
<td>da_1</td>
<td>math-bif(mk-id(COSD),op_1)</td>
</tr>
<tr>
<td>COSH</td>
<td>da_1</td>
<td>math-bif(mk-id(COSH),op_1)</td>
</tr>
<tr>
<td>ERF</td>
<td>da_1</td>
<td>math-bif(mk-id(ERF),op_1)</td>
</tr>
<tr>
<td>ERFDC</td>
<td>da_1</td>
<td>math-bif(mk-id(ERFDC),op_1)</td>
</tr>
<tr>
<td>EXP</td>
<td>da_1</td>
<td>math-bif(mk-id(EXP),op_1)</td>
</tr>
<tr>
<td>LOG</td>
<td>da_1</td>
<td>eval-log(op_1)</td>
</tr>
<tr>
<td>LOG10</td>
<td>da_1</td>
<td>eval-log-10(op_1)</td>
</tr>
<tr>
<td>LOG2</td>
<td>da_1</td>
<td>eval-log-2(op_1)</td>
</tr>
<tr>
<td>SIN</td>
<td>da_1</td>
<td>math-bif(mk-id(SIN),op_1)</td>
</tr>
<tr>
<td>SIND</td>
<td>da_1</td>
<td>math-bif(mk-id(SIND),op_1)</td>
</tr>
<tr>
<td>SINH</td>
<td>da_1</td>
<td>math-bif(mk-id(SINH),op_1)</td>
</tr>
<tr>
<td>SQRT</td>
<td>da_1</td>
<td>eval-sqrt(op_1)</td>
</tr>
<tr>
<td>TAN</td>
<td>da_1</td>
<td>math-bif(mk-id(TAN),op_1)</td>
</tr>
<tr>
<td>TAND</td>
<td>da_1</td>
<td>math-bif(mk-id(TAND),op_1)</td>
</tr>
<tr>
<td>TANH</td>
<td>da_1</td>
<td>math-bif(mk-id(TANH),op_1)</td>
</tr>
</tbody>
</table>

Ref.: eval-acos 12-52(112)
eval-asin 12-52(113)
eval-atan 12-52(114)
eval-atanh 12-53(115)
eval-atanh 12-53(116)
math-bif 12-54(123)
eval-log-10 12-54(118)
eval-log-2 12-54(119)
eval-sqrt 12-54(120)

12. BUILT-IN FUNCTIONS AND PSEUDO-VARIABLES 27
### Generic Built-in Functions for Array Manipulation

<table>
<thead>
<tr>
<th>f(id)</th>
<th>id &amp; min(id)</th>
<th>arg-class(id,i)</th>
<th>arg-type(id,i)</th>
<th>arg-eda(id,i)</th>
<th>arg-test(id,i)</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALL</td>
<td>1</td>
<td>ARR</td>
<td>CPL</td>
<td>BIT-ERA</td>
<td>ARR</td>
</tr>
<tr>
<td>ANY</td>
<td>1</td>
<td>ARR</td>
<td>CPL</td>
<td>BIT-ERA</td>
<td>ARR</td>
</tr>
<tr>
<td>DIM</td>
<td>1</td>
<td>IRRILL</td>
<td>EVA</td>
<td>*</td>
<td>*</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>IRRILL</td>
<td>INTG</td>
<td>*</td>
<td>T</td>
</tr>
<tr>
<td>LBOUND</td>
<td>1</td>
<td>IRRILL</td>
<td>EVA</td>
<td>*</td>
<td>*</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>IRRILL</td>
<td>INTG</td>
<td>*</td>
<td>T</td>
</tr>
<tr>
<td>UBOUND</td>
<td>1</td>
<td>IRRILL</td>
<td>EVA</td>
<td>*</td>
<td>*</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>IRRILL</td>
<td>INTG</td>
<td>*</td>
<td>T</td>
</tr>
<tr>
<td>DOLY</td>
<td>1</td>
<td>ARR</td>
<td>CPL</td>
<td>TLT-ERA</td>
<td>LTB-ARR</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>ARR</td>
<td>CPL</td>
<td>TLT-ERA</td>
<td>LTB-ARR</td>
</tr>
<tr>
<td>ERCZ</td>
<td>1</td>
<td>ARR</td>
<td>CPL</td>
<td>AR-ERA</td>
<td>ARR</td>
</tr>
<tr>
<td>SCN</td>
<td>1</td>
<td>ARR</td>
<td>CPL</td>
<td>AR-ERA</td>
<td>ARR</td>
</tr>
<tr>
<td>id</td>
<td>da-builtin(id, list)</td>
<td>eval-builtin-1(id, list)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>----------</td>
<td>----------------------</td>
<td>--------------------------</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ALL</td>
<td>da-all(da₁)</td>
<td>eval-all(op₁₁)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ANY</td>
<td>da-all(da₁)</td>
<td>eval-any(op₁₁)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DIM</td>
<td>BINTG-EDA</td>
<td>eval-dim(eva₁, k₂)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>HBOUND</td>
<td>BINTG-EDA</td>
<td>eval-hbound(eva₁, k₂)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LBOUND</td>
<td>BINTG-EDA</td>
<td>eval-lbound(eva₁, k₂)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>POLY</td>
<td>da-poly(da₁, da₂)</td>
<td>eval-poly(op₁, op₁₂)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PROD</td>
<td>da-prod(da₁)</td>
<td>eval-prod(op₁)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SUM</td>
<td>da-sum(da₁)</td>
<td>eval-sum(op₁)</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Ref.: evalu-all 12-57(124)  
eval-any 12-57(126)  
eval-dim 12-57(127)  
eval-hbound 12-57(128)  
eval-lbound 12-57(129)  
eval-poly 12-57(130)  
eval-prod 12-58(133)  
eval-sum 12-59(135)
### Condition Built-in Functions

<table>
<thead>
<tr>
<th>Name</th>
<th>id</th>
<th>ilmin(id)</th>
<th>arg-class(id,i)</th>
<th>arg-type(id,i)</th>
<th>arg-eds(id,i)</th>
<th>arg-test(id,i)</th>
</tr>
</thead>
<tbody>
<tr>
<td>INAPFDLE</td>
<td></td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IGNATTN</td>
<td></td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ONCHAR</td>
<td></td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ORCODE</td>
<td></td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ORCOUNT</td>
<td></td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ORTITLE</td>
<td></td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ORIPTNT</td>
<td></td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ORRZLT</td>
<td></td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ORRZCC</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ORRZRF</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ALTRU</td>
<td>1</td>
<td>1</td>
<td>TEXT</td>
<td>*</td>
<td>*</td>
<td></td>
</tr>
<tr>
<td>EMPTY</td>
<td></td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NULL</td>
<td></td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NULL0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>OIPSET</td>
<td>1</td>
<td>2</td>
<td>OP</td>
<td>PTR</td>
<td>T</td>
<td></td>
</tr>
<tr>
<td>OIPSET</td>
<td>2</td>
<td></td>
<td>IXEPED</td>
<td>GIN</td>
<td>*</td>
<td></td>
</tr>
<tr>
<td>ORCOUNT</td>
<td>1</td>
<td>2</td>
<td>IXEPED</td>
<td>GIN</td>
<td>OFFSET</td>
<td>T</td>
</tr>
<tr>
<td>FCH</td>
<td>2</td>
<td></td>
<td>IXEPED</td>
<td>GIN</td>
<td>*</td>
<td></td>
</tr>
<tr>
<td>id</td>
<td>do-builtin(id, list)</td>
<td>eval-builtin-1(id, list)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>------------</td>
<td>----------------------</td>
<td>--------------------------</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DATAFIELD</td>
<td>CHAF-EDA</td>
<td>eval-datsfield</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CNATTN</td>
<td>FTR</td>
<td>eval-cnatto</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CNCHAR</td>
<td>CHAF-ELA</td>
<td>eval-cnchar</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CNCODE</td>
<td>BINTG-ELA</td>
<td>eval-cncode</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CNCOUNT</td>
<td>BINTG-ELA</td>
<td>eval-cncount</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CNFILE</td>
<td>CHAF-ELA</td>
<td>eval-cnfile</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ONIDENT</td>
<td>µ(CHAF-EDA; (s-varying; VAR))</td>
<td>eval-cnider</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ONKEY</td>
<td>µ(CHAF-EDA; (s-varying; VAR))</td>
<td>eval-cnkey</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CNLOC</td>
<td>CHAF-ELA</td>
<td>eval-cnloc</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CONSOURCEC</td>
<td>CHAF-ELA</td>
<td>eval-consource</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ADDR</td>
<td>FTR</td>
<td>eval-addr(t.)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>EMPTY</td>
<td>po(<a href="">s-size:+</a>)</td>
<td>eval-empty</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NULL</td>
<td>FTR</td>
<td>eval-null</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NULLC</td>
<td>OFFSET</td>
<td>eval-nulc</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>OFFSET</td>
<td>OFFSET</td>
<td>eval-offset (op, gena)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>POINTER</td>
<td>PTR</td>
<td>eval-ptr (op, gena)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PTR</td>
<td>PTR</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Ref.:  
`eval-datsfield` 12-60(138)  
`eval-cnatto` 12-60 (139)  
`eval-cnchar` 12-60 (140)  
`eval-cncode` 12-60 (142)  
`eval-cncount` 12-60 (143)  
`eval-cnfile` 12-61 (144)  
`eval-cnider` 12-61 (145)  
`eval-cnkey` 12-61 (146)  
`eval-cnloc` 12-61 (147)  
`eval-consource` 12-61 (148)  
`eval-addr` 12-62 (150)  
`eval-empty` 12-62 (151)  
`eval-null` 12-62 (153)  
`eval-nulc` 12-63 (154)  
`eval-offset` 12-63 (156)  
`eval-ptr` 12-63 (157)
### Multitasking Built-In Functions

#### Miscellaneous Built-In Functions

<table>
<thead>
<tr>
<th>id</th>
<th>ilmin(id)</th>
<th>arg-class(id,i)</th>
<th>arg-type(id,i)</th>
<th>arg-eda(id,i)</th>
<th>arg-test(id,i)</th>
</tr>
</thead>
<tbody>
<tr>
<td>COMPLETION</td>
<td>1</td>
<td>EXPME</td>
<td>OP</td>
<td>EVENT</td>
<td>T</td>
</tr>
<tr>
<td>CFSN</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PRIORITY</td>
<td>1</td>
<td>EXPME</td>
<td>OP</td>
<td>TASK</td>
<td>T</td>
</tr>
<tr>
<td>STATUS</td>
<td>1</td>
<td>EXPME</td>
<td>OP</td>
<td>EVENT</td>
<td>T</td>
</tr>
<tr>
<td>ALLOCATION</td>
<td>1</td>
<td>IRREL</td>
<td>TEXT</td>
<td>*</td>
<td>*</td>
</tr>
<tr>
<td>COUNT</td>
<td>1</td>
<td>EXPME</td>
<td>OP</td>
<td>FILE</td>
<td>T</td>
</tr>
<tr>
<td>DATI</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>INAPLID</td>
<td>1</td>
<td>IRREL</td>
<td>TEXT</td>
<td>*</td>
<td>*</td>
</tr>
<tr>
<td>LIMING</td>
<td>1</td>
<td>EXPME</td>
<td>OP</td>
<td>FILE</td>
<td>T</td>
</tr>
<tr>
<td>IMA</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>T</td>
<td>1</td>
<td>O</td>
<td>O</td>
<td></td>
<td></td>
</tr>
<tr>
<td>id</td>
<td>da-builtin(id, list)</td>
<td>eval-builtin-1(id, list)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>-------</td>
<td>----------------------</td>
<td>--------------------------</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>COMPLETION</td>
<td>bit-eda (1)</td>
<td>eval-completion (c1)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CELL</td>
<td>s-da (PRI-EVA)</td>
<td>eval-priority (c1)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IMPORT</td>
<td>BITGE-EDA</td>
<td>eval-status (c1)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ALLOCATION</td>
<td>BITGE-EDA</td>
<td>eval-allocation (t1)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>COUNT</td>
<td>BITGE-EDA</td>
<td>eval-count (c1)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DATE</td>
<td>char-eda (6)</td>
<td>eval-date</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ENABLE</td>
<td>BIT-EDA</td>
<td>eval-enabled (t1)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LINENC</td>
<td>BITGE-EDA</td>
<td>eval-linenc (c1)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TIME</td>
<td>char-eda (9)</td>
<td>eval-time</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Ref.:  
eval-completion 12-64(158)  
eval-priority 12-64(159)  
eval-status 12-64(160)  
eval-allocation 12-64(161)  
eval-count 12-64(162)  
eval-date 12-65(163)  
eval-enabled 12-65(165)  
eval-linenc 12-66(166)  
eval-time 12-66(168)
12.6 EVALUATION OF THE INDIVIDUAL BUILT-IN FUNCTIONS

Section 12.6.2 defines the instructions listed in the last column of the table given in section 12.5.2. These instructions evaluate the individual built-in functions.

Section 12.6.1 defines some general auxiliary functions and an instruction used in 12.6.2 and in the last but one column of the table for determination of data attributes of the result.

Metavariables

<table>
<thead>
<tr>
<th>Metavariable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cp</td>
<td>is-cp</td>
</tr>
<tr>
<td>cpl</td>
<td>is-cpl-list</td>
</tr>
<tr>
<td>vr</td>
<td>is-vr</td>
</tr>
<tr>
<td>da</td>
<td>is-da</td>
</tr>
<tr>
<td>eda</td>
<td>is-edata</td>
</tr>
<tr>
<td>eva</td>
<td>is-eval</td>
</tr>
<tr>
<td>p</td>
<td>is-irtg-val v is-@ a precision or string length if specified</td>
</tr>
<tr>
<td>q</td>
<td>is-irtg-val v is-@ a scale factor, if specified</td>
</tr>
<tr>
<td>i, j, k</td>
<td>is-irtg-val v is-@ an integer value, sometimes optional</td>
</tr>
<tr>
<td>t</td>
<td>is-expr</td>
</tr>
<tr>
<td>ref</td>
<td>is-ref</td>
</tr>
<tr>
<td>gen</td>
<td>is-gen</td>
</tr>
</tbody>
</table>

12.6.1 GENERAL AUXILIARY DEFINITIONS

(39) \[ \text{new}(x, \text{da}) = \]
\[
\begin{align*}
x & \in \{BYII, CPLX\} \rightarrow \mu(\text{da};<\text{mode};x>) \\
x & \in \{BIN, DIC, BIT, CRAB\} \rightarrow \mu(\text{da};<\text{base};x>) \\
x & \in \{FIIT, FLT\} \rightarrow \mu(\text{da};<\text{scale};x>) \\
\text{is-VARYING}(x) & \rightarrow \mu(\text{da};<\text{varying};T>)
\end{align*}
\]

Note: This instruction is used to change a single attribute of \( \text{da} \), it is never used to change from arithmetic to string or vice versa.
30 April 1969

| 12. BUILT-IN FUNCTIONS AND PSEUDO-VARIABLES | 35 |

(40) pref(eda-1,eda-2) =

\[ \text{is-f}(eda-2) \rightarrow eda-1 \]
\[ \text{is-arithm}(eda-1) \land \text{is-arithm}(eda-2) \rightarrow \]
\[ \mu(\text{pref-ar-eda}(eda-1,eda-2) ; <s-prec: pref-prec(eda-1,eda-2)>,<s-scale-f: pref-scale-f(eda-1,eda-2)>) \]
\[ \text{is-string}(eda-1) \land \text{is-string}(eda-2) \rightarrow \]
\[ \mu(\text{pref-str-eda}(eda-1,eda-2) ; <s-length: max(s-length(eda-1),s-length(eda-2))>) \]

\[
\text{for: is-eda(eda-2) } \forall \text{ is-f(eda-2)};
\]

Ref.: pref-ar-eda 9-22(93)
pref-str-eda 9-22(97)

Note: This function completes the functions pref-ar-eda and pref-str-eda by specifying also the preferred precision and scale factor or length of two data attributes. It merges two arithmetic or two string attributes to a complete (evaluated) data attribute with the higher characteristics of both of them.

(41) pref-prec(eda-1,eda-2) =

\[ \text{is-fix}s-scale(eda-1) \land \text{is-fix}s-scale(eda-2) \rightarrow \]
\[ \min(W_0, \max(p_1 - q_1, p_2 - q_2) \times \max(q_1, q_2)) \]
\[ T \rightarrow \max(p_1, p_2) \]

where:
\[ p_1 = s-prec(eda-1) \]
\[ p_2 = s-prec(eda-2) \]
\[ q_1 = s-scale-f(eda-1) \]
\[ q_2 = s-scale-f(eda-2) \]
\[ W_0 = \max(s-prec-pref-ar-eda(eda-1,eda-2)) \]

\[
\text{for: is-arithm(eda-1), is-arithm(eda-2)}
\]

Ref.: max-prec 9-12(61)
pref-ar-eda 9-22(93)

(42) pref-scale-f(eda-1,eda-2) =

\[ \text{is-fix}s-scale(eda-1) \land \text{is-fix}s-scale(eda-2) \rightarrow \]
\[ \max(s-scale-f(eda-1), s-scale-f(eda-2)) \]
\[ T \rightarrow 0 \]

\[
\text{for: is-arithm(eda-1), is-arithm(eda-2)}
\]
pref-1(eda, eda-list) =
   is-<>(eda-list) " eda
   T  " pref-1(pref(head(eda-list), eda), tail(eda-list))

for: is-eda(eda) " is-<>(eda), is-eda-list(eda-list)

Note: Used with eda = 0, this function computes the highest characteristics of a list of data-attributes.

eva-f(m, b, s, p, q) =
   \mu_0(<s-dens:AL>,<s-da:eda-f(m, b, s, p, q)>)

for: (is-REAL v is-CPLX)(m), (is-EIN v is-DEC)(b), (is-FIX v is-FLT)(s)

eda-f(m, b, s, p, q) =
   \mu_0(<s-node:m>, <s-base:b>, <s-scale:s>, <s-prec:p>, <s-scale-f:q>)

for: (is-REAL v is-CPLX)(m), (is-EIN v is-DEC)(b), (is-FIX v is-FLT)(s)

val-op-1(eda, v) =
   mk-cr(aggr-scalar(eda), vr);
   vr: test-rep(aggr-scalar(eda), v)

for: is-value(v)

Ref:  mk-cr 9-9(35)
      aggr-scalar 8-23(64)
      test-rep 9-7(27)
      is-value 9-3(2)

Note: This instruction constructs an operand from given data attributes and value, including a test for the size condition.
Ref.: is-intg-val 9-3(5)
convert-1 9-29(119)
aggr-scalar 8-23(64)

(48) eval-hocl(op-1,op-2,op-3) =

\[ \text{PASS:val-op}(\text{pref}(\text{op-da}(op-1),\text{op-da}(op-2)),
\text{bool}(\text{op-val}(op-1),\text{op-val}(op-2),\text{op-val}(op-3))) \]

Ref.: val-op 9-9(38)
op-da 9-9(37)
op-val 9-9(36)

(49) hocl(v,-1,v-2,v-3) =

\[ \text{length-m}_o \]
\[ \text{l IST } \text{elem-hocl(elem}(i,v)_1,\text{elem}(i,v)_2,\text{v}-3) \]

where:
\[ \text{length-r}_o = \text{max} (\text{length}(v-1),\text{length}(v-2)) \]
\[ v_1 = \text{adjust-string(BIT, length-m}_o ,v-1) \]
\[ v_2 = \text{adjust-string(BIT, length-m}_o ,v-2) \]

for:is-bit-val-list(v-1),is-bit-val-list(v-2),is-bit-val-list(v-3)

Ref.: adjust-string 9-8(32)
is-bit-val 9-4(11)

(50) elem-hocl(bit-1,bit-2,bit-list) =

\[ \text{is-0-BIT(bit-1)} \& \text{is-0-BIT(bit-2)} \rightarrow \text{elem}(1,bit-list) \]
\[ \text{is-0-BIT(bit-1)} \& \text{is-1-BIT(bit-2)} \rightarrow \text{elem}(2,bit-list) \]
\[ \text{is-1-BIT(bit-1)} \& \text{is-0-BIT(bit-2)} \rightarrow \text{elem}(3,bit-list) \]
\[ \text{is-1-BIT(bit-1)} \& \text{is-1-BIT(bit-2)} \rightarrow \text{elem}(4,bit-list) \]

for:is-bit-val(bit-1),is-bit-val(bit-2),is-bit-val-list(bit-list)

Ref.: is-bit-val 9-4(11)

(51) da-hocl(da-1,da-2) =

\[ \mu(\text{BIT-EDA};<s-varying:pref-varying(da-1,da-2)>) \]

Ref.: pref-varying 9-22(99)

(52) sval-char(cp,i) =

\[ \text{is-intg-val}(i) \rightarrow \text{convert-1}(\mu(\text{aggr-scalar(CHAR-EDA)};<s-length>s-da;i),op) \]
\[ \text{is-S}(i) \rightarrow \text{convert-1}(\text{aggr-scalar(CHAR-EDA)},op) \]

cont'd
Ref.: is-int-val 9-3(5)
        convert 9-29(119)
        agr-scalar 8-23(64)

(53) \texttt{eval-high}(k) =

\begin{align*}
\text{PASS:val-op(char-eda(k), \quad \text{LIST} \quad \text{CHAR}_k)}
\end{align*}

where:
\begin{align*}
\text{CHAR}_k = (\text{Lchar}) \text{is-char-val(char)} \land \neg(3\text{char-1}) \text{(collat(char-1) > collat(char))}
\end{align*}

Ref.: val-op 9-9(38)
        char-eda 9-12(58)
        is-char-val 9-3(6)
        collat 9-18(82)

(54) \texttt{eval-index(op-1, op-2)} =

\begin{align*}
s\text{-base}\text{-op-da}(op-1) \neq s\text{-base}\text{-op-da}(op-2) \rightarrow
\text{eval-index(op-11, op-21)};
\quad \text{op-11:convert-1(agr-scalar(CHAR-EDA), op-1)},
\quad \text{op-21:convert-1(agr-scalar(CHAR-EDA), op-2)}
\end{align*}

\begin{align*}
T \rightarrow \text{PASS:bintg-op*index(op-val(op-1), op-val(op-2))}
\end{align*}

Ref.: op-da 9-9(37)
        convert 9-29(119)
        agr-scalar 8-23(64)
        bintg-op 9-10(80)
        op-val 9-9(36)

(55) \texttt{index(str, conf)} =

\begin{align*}
is-[](i\text{-set}_1) \rightarrow 0
\end{align*}

\begin{align*}
T \rightarrow \text{min-set}(i\text{-set}_1)
\end{align*}

where:
\begin{align*}
i\text{-set}_1 = \{i \mid i > 0 \land \text{conf} = \text{substr}(str, i, \text{length(conf))}\}
\end{align*}

\begin{align*}
\text{for:}(is\text{-bit-val-list} \lor is\text{-char-val-list})(str), (is\text{-bit-val-list} \lor is\text{-char-val-list})(conf)
\end{align*}

Ref.: is-bit-val 9-4(11)
        is-char-val 9-3(6)

(56) \texttt{eval-length(op)} =

\begin{align*}
\text{PASS:bintg-op*length*op-val(op)}
\end{align*}

Ref.: bintg-op 9-10(80)
        op-val 9-9(36)
(57) \texttt{eval-low}(k) =
\begin{align*}
\text{PASS:} & \text{val-op(char-ed}a(k), \text{ LIST } \text{CHAR}_0) \\
\text{where:} & \text{CHAR}_0 = (\text{Uchar}) (\text{is-char-val}(\text{char}) \land \neg (\exists \text{char-1}) (\text{collat(char-1)} < \text{collat(char)}))
\end{align*}

Ref.: & \text{val-op 9-9(38)} \\
& \text{char-ed}a 9-12(58) \\
& \text{is-char-val 9-3(6)} \\
& \text{collat 9-18(62)}

(58) \texttt{eval-repeat}(op,i) = \\
\text{eval-repeat-1}(op,op,i)

(59) \texttt{eval-repeat-1}(op-1,op,i) = \\
i \leq 0 \implies \text{PASS:op-1} \\
T \implies \\
\text{eval-repeat-1}(op-2,op,i-1); \\
\text{op-2:infix-op}(op-1,op,CAT)

Ref.: & \text{infix-op 9-13(65)}

(60) \texttt{eval-string}(opl) = \\
(\forall i (1 \leq i \leq \text{length}(opl)) \implies \\
\text{s-base\*op-da\*head}(opl) = \text{s-base\*op-da\*elem}(i,opl)) \implies \\
\text{iterate-infix-op}(\text{head}(opl),\text{tail}(opl),\text{CAT}) \\
T \implies \text{error}

Ref.: & \text{op-da 9-9(37)}

(61) \texttt{da-string}(aggr) = \\
(\exists i \land \neg 2) (\text{is-CHAR-1}(aggr) \land \text{is-\text{FIT-2}}(aggr)) \implies \text{error} \\
(\exists e) (\text{is-CHAR}(aggr)) \implies \mu(\text{CHAR-\text{\#DA}; <s-varying:var_1>) \\
T \implies \mu(\text{\text{\#EA}; <s-varying:var_1>})

where: & \text{var}_1 = (\exists e) (\text{varying\*s}(aggr) = \text{VAR}) \implies \text{VAR}, \\
& T \implies 0

for:is-aggr(aggr).
(62) \( \text{eval-substr}(t, i, j) = \)
\[
\text{is-gen-base}(t, AT) \rightarrow
\]
\[
\text{eval-substr-1}(\text{op}, \text{gen}, i, j);
\]
\[
\text{op} = \text{gen-or-1}(	ext{gen});
\]
\[
\text{gen} = \text{eval-lp}(t);
\]
\[
T \rightarrow
\]
\[
\text{eval-substr-1}(\text{op}, 0, i, j, 0);
\]
\[
\text{op} = \text{gen-expr-1}(\text{agg} - \text{scalar}(\text{string-eq}), \text{op}-1);
\]
\[
\text{op} = \text{eval-expr-2}(t)
\]

Ref.:
\* ref-base 11-38(106)
\* \text{eval-op-1} 6-49(136)
\* \text{eval-lp} 8-9(20)
\* \text{gen-expr-1} 9-29(119)
\* \text{agg} - \text{scalar} 8-23(68)
\* \text{eval-expr-2} 9-19(56)

(63) \( \text{eval-substr-1}(\text{op}, \text{gen}, i, j, \text{cond}) = \)
\[
is-mon(j) \neq 1 \land i \leq k_2 \land \text{is-mon}(j) \neq 1 \land i \leq j \land k_2 + 1 \rightarrow
\]
\[
\text{eval-op}(\text{da}, \text{substr}(\text{op-val}(\text{op}), i, j))
\]
\[
is-gen(\text{gen}) \rightarrow
\]
\[
\text{eval-op}(\text{da}, \text{substr}(\text{op-val}(\text{op}-1), i, j))
\]
\[
\text{op} = \text{gen-or-1}(\text{gen});
\]
\[
\text{gen-expr-cond}(\text{cond})
\]
\[
is-\text{def}(\text{gen}) \rightarrow
\]
\[
\text{eval-op}(\text{da}, \text{substr}(\text{op-val}(\text{op}), i, j))
\]
\[
\text{gen-expr-cond}(\text{cond})
\]

where:
\[
k_2 = \text{length} - \text{op-val}(\text{op})
\]
\[
\text{da} = \mu(\text{op-da}(\text{op}) ; <s - \text{varying}; \text{VAR})
\]

for \( is-\text{def} \land is-\text{dc-string} \)(\text{cond})

Ref.:
\* \text{eval-op} 8-9(38)
\* \text{op-val} 8-9(26)
\* \text{is-gen} 3-14(30)
\* \text{gen-expr-1} 6-49(136)
\* \text{op-da} 9-9(37)
(64) $\text{substr}(\text{str}, i, j) =$
\[ t, \text{LIST elem}(n, \text{str}) \]

where:
\[ i_1 = \max(1, i) \]
\[ l_1 = \text{is-Q}(j) \rightarrow \text{length(str)} \]
\[ T \rightarrow \min(\text{length(str)}, e_1) \]
\[ e_1 = i + j - 1 \]

for: $\text{is-char-val-list(str)}$

Ref.: $\text{is-char-val } 9{-}3(6)$

(65) $\text{substr-cond}(\text{cond}) =$
\[ \text{is-NO-STRG}(\text{cond}) \rightarrow \text{error} \]
\[ T \rightarrow \text{call-cond}(\text{STRG}) \]

for: $(\text{is-Q} \lor \text{is-NO-STRG})(\text{cond})$

Ref.: $\text{call-cond } 10{-}18(54)$

(66) $\text{eval-translate}(op-1, op-2, op-3) =$
\[ \text{is-Q}(op-3) \rightarrow \text{eval-translate}(op-1, op-2, op-tr_4) \]
\[ \neg(s_{-}\text{base}=op-da(op-1) = s_{-}\text{base}=op-da(op-2) = s_{-}\text{base}=op-da(op-3)) \rightarrow \]
\[ \text{eval-translate}(op-11, op-21, op-31); \]
\[ op-11: \text{convert-1}(\text{aggr-scalar}(\text{CHAR-EDA}), op-1), \]
\[ op-21: \text{convert-1}(\text{aggr-scalar}(\text{CHAR-EDA}), op-2), \]
\[ op-31: \text{convert-1}(\text{aggr-scalar}(\text{CHAR-EDA}), op-3) \]
\[ T \rightarrow \text{PASS}:\text{val-op}(op-da(op-1), \text{translate}(str_4, repl_4, pos_4)) \]

where:
\[ \text{str}_4 = op-val(op-1) \]
\[ \text{repl}_4 = op-val(op-2) \]
\[ \text{pos}_4 = op-val(op-3) \]
\[ \text{op-tr}_4 = \text{val-op}(\text{char-eda}*\text{length(TS-STR)}, \text{TR-STR}) \]

Ref.: $\text{op-da } 9{-}9(37)$
\[ \text{convert-1 } 9{-}29(119) \]
\[ \text{aggr-scalar } 8{-}23(64) \]
\[ \text{val-op } 9{-}9(39) \]
\[ \text{op-val } 9{-}9(36) \]
\[ \text{char-eda } 9{-}12(56) \]

(67) $\text{is-char-val-list}(\text{TR-STR})$

Ref.: $\text{is-char-val } 9{-}3(6)$
Note: TR-STR is implementation defined.

\[(68)\]
\[\text{translate}(\text{str}, \text{repl}, \text{pos}) =\]
\[\begin{align*}
\text{length(\text{repl}) < length(\text{pos})} & \rightarrow \\
\text{translate}(\text{str}, \text{adjust-string}(b_0, \text{length(\text{pos})}, \text{repl}), \text{pos}) & \rightarrow \\
\text{length(\text{str})} & \rightarrow \text{LIST} \\
\end{align*}\]
\[\text{elem-trans}(\text{elem}(i, \text{str}), \text{repl}, \text{pos})\]

where:
\[b_0 = (\text{is-char-val-list(\text{str})} \leftarrow \text{CHAR,} \]
\[\rightarrow \text{BIT})\]

\[\text{for: (is-char-val-list } \land \text{ is-bit-val-list)} (\text{str}), (\text{is-char-val-list } \land \text{ is-bit-val-list)} (\text{repl}), (\text{is-char-val-list } \land \text{ is-bit-val-list)} (\text{pos})\]

Ref.: \text{adjust-string} 9-8(32)
\text{is-char-val} 9-3(6)
\text{is-bit-val} 9-4(11)

\[(69)\]
\[\text{elem-trans}(\text{char}, \text{repl}, \text{pos}) =\]
\[\begin{align*}
\text{indx}(\text{pos}, \langle \text{char} \rangle) & = 0 \rightarrow \text{char} \\
\text{T} & \rightarrow \text{elem}(\text{indx}(\text{pos}, \langle \text{char} \rangle), \text{repl}) \\
\end{align*}\]

\[\text{for: (is-char-val } \land \text{ is-bit-val)} (\text{char}), (\text{is-char-val-list } \land \text{ is-bit-val-list)} (\text{repl}), (\text{is-char-val-list } \land \text{ is-bit-val-list)} (\text{pos})\]

Ref.: \text{is-char-val} 9-3(6)
\text{is-bit-val} 9-4(11)

\[(70)\]
\[\text{eval-unspec}(\text{gen}) =\]
\[\begin{align*}
(\text{is-arithm } \land \text{ is-string } \land \text{ is-pic } \land \text{ is-area } \land \text{ is-PTR } \land \text{ is-OFFSET)} & \rightarrow \text{gen-da}(\text{gen}) \rightarrow \\
\text{mk-op}(\text{eva}_1, \text{vr}_1) & \rightarrow \\
\end{align*}\]

where:
\[\text{eva}_1 = \text{str-eva(BIT, unspec-length } \land \text{ size(vr}_1)) \]
\[\text{vr}_1 = s-pp(\text{gen}) (3)\]

Ref.: \text{gen-da} 7-17(53)
\text{mk-op} 9-9(35)
\text{str-eva} 7-14(45)
\text{size} 3-15(33)

\[(71)\]
\[\text{unspec-length}(z) =\]

Note: This function is described in the following:
(72) \[ z = \text{alloc-size}(\text{eva}) > \text{is-intg-val}\cdot\text{unspec-length}(z) \; \& \]
\[ z = \text{alloc-size}\cdot\text{str-eva}(\text{BIT},\text{unspec-length}(z)) \; \& \]
\[ \text{is-bit-val-list}\cdot\text{val}((\text{str-eva}(\text{BIT},\text{unspec-length}\cdot\text{size}(\text{vr}))), \text{vr}) \]

for: is-size(z)

Ref.: alloc-size 7-11(26)
      is-intg-val 9-3(5)
      str-eva 7-74(45)
      is-bit-val 9-4(11)
      val 9-6(18)
      size 3-15(33)
      is-size 3-15(34)

Note: The function \text{unspec-length} determines the length of the bit string corresponding to a given value representation \text{vr}. It depends only on \( z = \text{size}(\text{vr}) \). The last part of the axiom states that every value representation \text{vr} can be interpreted as the value representation of a bit-string.

(73) \text{eval-verify}(\text{op-1}, \text{op-2}) = \]
\[ \text{s-base}\cdot\text{op-da}(\text{op-1}) \neq \text{s-base}\cdot\text{op-da}(\text{op-2}) \rightarrow \]
\[ \text{eval-verify}(\text{op-11}, \text{op-21}); \]
\[ \text{op-11: convert-1}(\text{aggr-scalar}(\text{CHAR-EDA}), \text{op-1}), \]
\[ \text{op-21: convert-1}(\text{aggr-scalar}(\text{CHAR-EDA}), \text{op-2}) \]
\[ \text{T} \rightarrow \text{PASS: bintg-op-verify}(\text{op-val}(\text{op-1}), \text{op-val}(\text{op-2})) \]

Ref.: \text{op-da} 9-9(37)
      \text{convert-1} 9-29(119)
      \text{aggr-scalar} 8-23(84)
      \text{bintg-op} 9-15(40)
      \text{op-val} 9-9(36)

(74) \text{verify}(\text{arg}, \text{str}) = \]
\[ \text{is-}(\text{set}_{\text{i}}) \rightarrow 0 \]
\[ \text{T} \rightarrow \text{min-set}(\text{set}_{\text{i}}) \]

where:\n\[ \text{set}_{\text{i}} = \{ i \mid \text{index}(\text{str}, \text{elem}(i, \text{arg})) = 0 \} \]

for: (is-char-val-list \( \vee \) is-bit-val-list)(\text{arg}), (is-char-val-list \( \vee \) is-bit-val-list)(\text{str})

Ref.: \text{is-char-val} 9-3(6)
      \text{is-bit-val} 9-4(11)
12.6.2.2 Arithmetic generic built-in functions

(75) \[
eval-infix-convert(op-x, op-y, p, q, opr) =
\]

\[
p \leq N_4 \land (is-FLT\cdot s-scale(eda-res_1) \equiv is-q(q)) \rightarrow
\]

\[
\begin{align*}
& convert-1(eva_1, op-r) ; \\
& op-r : \text{infix-op}(op-1, op-2, opr) ; \\
& op-1 : convert-1(eva_1, op-x) , \\
& op-2 : convert-2(eva_1, op-y)
\end{align*}
\]

\[
T \rightarrow \text{error}
\]

where:

\[
\begin{align*}
eda-res_1 &= eda-infix(op-da(op-x), op-da(op-y), opr) \\
eva_1 &= aggr-scalar(eda_0) \\
eda_0 &= complete-tg(eda-f(*,*\cdot *\cdot *p,q), eda-res_1)
\end{align*}
\]

for:is-infix-opr(opr)

Ref.:

\begin{align*}
& convert-1 9-29(119) \\
& infix-op 9-13(65) \\
& convert-2 9-29(120) \\
& eda-infix 9-20(87) \\
& op-da 9-9(37) \\
& aggr-scalar 8-23(64) \\
& complete-tg 9-37(149)
\end{align*}

(76) \[
da-infix-1(da-1, da-2, p, q, opr) =
\]

\[
is-FLT\cdot s-scale(eda-r_1) \equiv is-q(q) \rightarrow complete-tg(eda-f(*,*\cdot *\cdot *p,q), eda-r_1)
\]

\[
T \rightarrow \text{error}
\]

where:

\[
eda-r_1 = res-edata(da-1, da-2, opr)
\]

for:is-infix-opr(opr)

Ref.:

\begin{align*}
& complete-tg 9-37(149) \\
& res-edata 9-23(100)
\end{align*}

(77) \[
eval-abs(op) =
\]

\[
is-REAL\cdot s-mode*op-da(op) \rightarrow val-op-1(op-da(op), abs-op-val(op))
\]

\[
is-CPLX\cdot s-mode*op-da(op) \rightarrow eval-cplx-abs(op)
\]

Ref.:

\begin{align*}
& op-da 9-9(37) \\
& op-val 9-9(36)
\end{align*}

Note: The instruction \texttt{eval-cplx-abs(op)} is in part implementation defined:
(78) \[ \text{eval-arg} \text{ (op) = } \]
\[ \text{P} \text{AS} \text{s} \text{val-op (da-abs op-da (op), v-res)} \]

\[ \text{FOR:} \text{i} \text{S-NUM-VAL (v-res)} \text{ A V-RES} \geq 0 \]

\[ \text{Ref.:} \text{ val-op 9-9(38)} \]
\[ \text{op-da 9-9(37)} \]
\[ \text{is-num-val 9-3(3)} \]

\[ \text{Note: v-res is (an approximation of) } \sqrt{v_1^2 + v_2^2} \text{ where } v_1 \text{ and } v_2 \text{ are the real and imaginary part of op-val(op), respectively.} \]

(79) \[ \text{da-abs (da) = } \]
\[ (\text{is-CPLX-s-mode} \text{ A IS-FIX-s-scale}) (da) \rightarrow \]
\[ \text{complete-tg(eda-f (REAL, * , * , MIN(N1, P1 + 1), G1), da)} \]
\[ T \rightarrow \text{nov (REAL, da)} \]

\[ \text{where:} \]
\[ N_1 = \text{max-prec (da)} \]
\[ P_1 = \text{s-prec (da)} \]
\[ G_1 = \text{s-scale-f (da)} \]

\[ \text{Ref.:} \text{ complete-tg 9-37(149)} \]
\[ \text{max-prec 9-12(61)} \]

(80) \[ \text{eval-add (cp-1, cp-2, p, g) = } \]
\[ \text{eval-infix-convert (op-1, op-2, p, g)} \]

(81) \[ \text{eval-bin (cp, p, g) = } \]
\[ \text{is-FLT-s-scale-op-da (op) A IS-G (g) \rightarrow \text{error}} \]
\[ T \rightarrow \text{convert-1(eda-f(*, BIN, *, p, g), op)} \]

\[ \text{Ref.:} \text{ op-da 9-9(37)} \]
\[ \text{convert-1 9-29(119)} \]

(82) \[ \text{da-bin (da, p, g) = } \]
\[ \text{complete-tg(eda-f(*, BIN, *, p, g), da)} \]

\[ \text{Ref.:} \text{ complete-tg 9-37(149)} \]

(83) \[ \text{eval-ceil (op) = } \]
\[ \text{val-op-1(da-trunc-op-da (op), ceil-op-val (op))} \]

\[ \text{Ref.:} \text{ op-da 9-9(37)} \]
\[ \text{op-val 9-9(36)} \]

12. BUILT-IN FUNCTIONS AND PSEUDO-VARIABLES 45
(84) \( \text{eval-cplx}(\text{op-1}, \text{op-2}) = \)

\[
\text{eval-cplx-1(\text{op-11}, \text{op-21});} \\
\text{cf-11: \text{convert-1(eva-pr1, op-1)},} \\
\text{cf-21: \text{convert-1(eva-pr1, op-2)}}
\]

where:
\[\text{eva-pr1} = \text{aggr-scalar\*pref(op-da(op-1), op-da(op-2))}\]

Ref.: \text{convert-1 9-29(119)} \text{aggr-scalar 8-23(64)} \text{op-da 9-9(37)}

(85) \( \text{eval-cplx-1(\text{op-1}, \text{op-2}) =} \)

\[
\text{PASS:val-op(da1, v1)}
\]

where:
\[\text{da1} = \text{new(CPLX, op-da(op-1))} \]
\[\text{v1} = \text{cplx(cp-val(op-1), op-val(op-2))}\]

Ref.: \text{val-cp 9-9(38) cp-da 9-9(37) op-val 9-9(36)}

(86) \( \text{eval-conjg}(\text{op}) = \)

\[
\text{PASS:val-op(cp-da(op), conjg\*op-val(op))}
\]

Ref.: \text{val-cp 9-9(38) cp-da 9-9(37) conjg 1-2(10) op-val 9-9(36)}

(87) \( \text{eval-dec}(\text{op}, p, q) = \)

\[
\text{is-FLT\*s-scale\*op-da(op) \& \text{is-G(q)} \rightarrow \text{error}} \\
\text{T} \rightarrow \text{convert-1(eva-f(*, DEC, *, p, q), op)}
\]

Ref.: \text{op-da 9-9(37) convert-1 9-29(119)}

(88) \( \text{da-dec(da, p, q)} = \)

\[
\text{complete-tg(eda-f(*, DEC, *, p, q), da)}
\]

Ref.: \text{complete-tg 9-37(149)}

(89) \( \text{eval-divide}(\text{cp-1, cp-2, p, q}) = \)

\[
\text{eval-infix-convert(cp-1, op-2, p, q, DIV)}
\]
Abstract Syntax and Interpretation of PL/I

(90) **eval-fixed**(cp,p,c) =

\[\text{convert-1 (aggr-scalar da-fixed (cp-da (op), p, c), op)}\]

Ref.: \(\text{convert-1 9-25(119)}\)
\(\text{aggr-scalar 8-23(64)}\)
\(\text{op-da 9-9(37)}\)

(91) **da-fixed**(da,p,q) =

\[\text{is-scalar (p) \rightarrow da-fixed (da, p, q)}\]
\[\text{is-scalar (q) \rightarrow da-fixed (da, p, q)}\]
\[T \rightarrow \text{complete-tg (eda-f (*, *, INT, p, c), da)}\]

where:
\[N_1 = (\text{is-scalar-base (da) \rightarrow int-to-DEC},\]
\[\text{is-scalar-base (da) \rightarrow int-to-DEC})\]

Ref.: \(\text{complete-tg 9-37(149)}\)

(92) **eval-float**(cp,p) =

\[\text{convert-1 (aggr-scalar da-float (op-da (op)), p, q)}\]

Ref.: \(\text{convert-1 9-25(119)}\)
\(\text{aggr-scalar 8-23(64)}\)
\(\text{op-da 9-9(37)}\)

(93) **da-float**(da,p) =

\[\text{is-scalar (p) \rightarrow da-float (da, N_1)}\]
\[T \rightarrow \text{complete-tg (eda-f (*, *, FLT, p), da)}\]

where:
\[N_1 = \text{def-prec (da)}\]

Ref.: \(\text{complete-tg 9-37(149)}\)

Note: The function \(\text{def-prec (da)}\) gives the default precision associated with \(da\) for float scale.

(94) **eval-floor**(cp) =

\[\text{eval-op-1 (da-trunc op-da (op), floor op-val (op))}\]

Ref.: \(\text{op-da 9-9(37)}\)
\(\text{op-val 9-9(36)}\)
(95) \texttt{eval-imag}(cpl) =
\begin{align*}
\text{PASS} & : \text{val-op}(da_1, v_1) \\
\text{where:} & \\
& da_1 = \text{new}(\text{REAL}, \text{op-da}(op)) \\
& v_1 = \text{imag} \cdot \text{op-val}(op) \\
\text{Ref.:} & \quad \text{val-op 9-9(36)} \\
& \text{op-da 9-9(37)} \\
& \text{op-val 9-9(36)}
\end{align*}

(96) \texttt{eval-max}(cpl) =
\begin{align*}
\text{val-op-1}(da \text{-pref}_1, v \text{-max}_1) \\
\text{where:} & \\
& \text{da-pref}_1 = \text{pref-1}(0, \text{LIST} \text{op-da} \cdot \text{elem}(i, \text{opl})) \\
& v \text{-max}_1 = \text{max-set}({\text{op-val} \cdot \text{elem}(i, \text{opl}) \mid 1 \leq i \leq \text{length}(\text{opl})}) \\
\text{Ref.:} & \quad \text{op-da 9-9(37)} \\
& \text{op-val 9-9(36)}
\end{align*}

(97) \texttt{eval-min}(cpl) =
\begin{align*}
\text{val-op-1}(da \text{-pref}_1, v \text{-min}_1) \\
\text{where:} & \\
& \text{da-pref}_1 = \text{pref-1}(0, \text{LIST} \text{op-da} \cdot \text{elem}(i, \text{opl})) \\
& v \text{-min}_1 = \text{min-set}({\text{op-val} \cdot \text{elem}(i, \text{opl}) \mid 1 \leq i \leq \text{length}(\text{opl})}) \\
\text{Ref.:} & \quad \text{op-da 9-9(37)} \\
& \text{op-val 9-9(36)}
\end{align*}

(98) \texttt{eval-mod}(cpl_1, op_2) =
\begin{align*}
\text{val-op-1}(da_1, v) \\
\text{v: pass-\text{modulo-}1}(op_1, op_2); \\
\text{op-11: convert-1}(\text{eva}_0, op_1), \\
\text{op-21: convert-1}(\text{eva}_0, op_2) \\
\text{where:} & \\
& \text{eva}_0 = \text{aggr-scalar} \cdot \text{pref-ar-edu}(\text{op-da}(op_1), \text{op-da}(op_2)) \\
& da_1 = \text{da-mod}(\text{op-da}(op_1), \text{op-da}(op_2)) \\
\text{Ref.:} & \quad \text{convert-1 9-29(119)} \\
& \text{aggr-scalar 8-23(64)} \\
& \text{pref-ar-edu 9-22(93)} \\
& \text{op-da 9-9(37)}
\end{align*}
(99) \[ \text{module-1}(\text{op-1}, \text{op-2}) = \]
\[ v_2 = 0 \rightarrow v_1 \]
\[ T \rightarrow \text{module}(v_1, \text{abs}(v_2)) \]

where:
\[ v_1 = \text{cp-val}(\text{op-1}) \]
\[ v_2 = \text{cr-val}(\text{op-2}) \]

Ref.: \( \text{cp-val} \) 9-9(36)

(100) \[ \text{da-mod}(\text{da-1}, \text{da-2}) = \]
\[ \text{is-lt} \in \text{scale}(\text{da-1}) \land \text{is-quot} \in \text{scale}(\text{da-2}) \rightarrow \text{pref}(\text{da-1}, \text{da-2}) \]
\[ T \rightarrow \text{complete-tg}(\text{eda-f}(*, *, *, \text{da-1}, \text{da-2}), \text{pref}(\text{da-1}, \text{da-2})) \]

where:
\[ p_0 = \min(N_0, F_2 - G_2 + G_0) \]
\[ G_0 = \max(G_1, G_2) \]
\[ F_2 = \text{prec}(\text{da-2}) \]
\[ G_1 = \text{scale-f}(\text{da-1}) \]
\[ G_2 = \text{scale-f}(\text{da-2}) \]
\[ N_0 = \max[\text{prec} \cdot \text{pref}(\text{da-1}, \text{da-2})] \]

Ref.: \( \text{complete-tg} \) 9-37(149)
\( \text{max-prec} \) 9-12(61)

(101) \[ \text{eval-multiply}(\text{cp-1}, \text{cr-2}, p, q) = \]
\[ \text{eval-infix-convert}(\text{cp-1}, \text{cr-2}, p, q, \text{MULT}) \]

(102) \[ \text{eval-op}(\text{cr}, f, q) = \]
\[ \text{is-lt} \in \text{scale}(\text{cp-da}(\text{op}) \equiv \text{is-quot}(q) \rightarrow \text{convert-1}(\text{eva-f}(*, *, *, f, q), q) \]
\[ T \rightarrow \text{error} \]

Ref.: \( \text{cp-da} \) 9-9(37)
\( \text{convert-1} \) 9-29(119)

(103) \[ \text{da-prec}(\text{da}, f, q) = \]
\[ \text{is-lt} \in \text{scale}(\text{da}) \land \text{is-quot}(q) \rightarrow \text{complete-tg}(\text{eda-f}(*, *, *, f, q), \text{da}) \]
\[ T \rightarrow \text{error} \]

Ref.: \( \text{complete-tg} \) 9-37(149)

12. BUILT-IN FUNCTIONS AND PSEUDO-VARIABLES 49
eval-real(op) =

\[
\text{PASS} : \text{val-op}(da_1, \text{real-op-val}(op))
\]

where:

\[
da_1 = \text{new}(\text{REAL}, \text{op-da}(op))
\]

Ref.: \[ \text{val-op } 9-9(38) \]
\[ \text{op-val } 9-9(36) \]
\[ \text{op-da } 9-9(37) \]

\( \text{eval-round}(t,i) = \)

\[
\text{eval-round-1(op,i)}; \quad \text{op=} \text{eval-expr-2}(t)
\]

Ref.: \[ \text{eval-expr-2 } 8-19(50) \]

\( \text{eval-round-1(op,i)} = \)

is-num-pic(da_1) \rightarrow

\[
\text{eval-round-1(op-1,i)}; \quad \text{op-1=} \text{eval-round-1(op-r,i)}; \quad \text{op-r=} \text{eval-real}(op), \quad \text{op-2=} \text{eval-round-1(op-im,i)}; \quad \text{op-im=} \text{eval-imag}(op)
\]

is-arithm(da_1) \rightarrow

\[
\text{pass-val-op}(da_3,v); \quad \text{v=} \text{test-fl}(aggr_2, \text{round}(da_1,v_1,i))
\]

is-string-type(da_1) \rightarrow \text{PASS} : \text{op}

\[ \text{T} \rightarrow \text{error} \]

where:

\[
w_1 = \text{op-val}(op) \quad da_4 = \text{op-da}(op)
\]
\[ da_2 = \text{da-round}(da_4) \]
\[ \text{aggr}_2 = \text{aggr-scalar}(da_2) \]

Ref.: \[ \text{is-num-pic } 9-45(173) \]
\[ \text{convert-1 } 9-29(119) \]
\[ \text{aggr-scalar } 8-23(64) \]
\[ \text{val-op } 9-9(38) \]
\[ \text{test-fl } 9-15(72) \]
\[ \text{is-string-type } 9-21(92) \]
\[ \text{op-val } 9-9(36) \]
\[ \text{op-da } 9-9(37) \]

Note: The function \text{round}(eda,v,i) is in part implementation defined; it satisfies the axiom:
30 April 1969

ABSTRACT SYNTAX AND INTERPRETATION OF PL/I

(107) \[(\text{is-arithm} \& \text{is-HEX-modes} \& \text{is-FIX-s-scale})(\text{eda}) \& \]
\[v \& v-0\text{-set}(\text{eda}) \Rightarrow \text{round}(\text{eda},v,i) = \text{sign}(v) \cdot (v \div \text{base}_i + i)\]

where:
\[v \div i = (\text{module}(v \div i) \geq 0 \cdot 5 \rightarrow v \div i + 1, \text{T} \rightarrow v \div i)\]
\[v \div i = \text{abs}(v) \cdot \text{base}_i + 1\]
\[\text{base}_i = (\text{is-FIX-s-base}(\text{eda}) \rightarrow 2, \text{T} \rightarrow 10)\]

for: \text{is-num-val}(v)

Ref.:
\[v-0\text{-set 9-5}(17)\]
\[\text{is-num-val 9-3}(3)\]

Note: For floating-point eva, the round function is implementation defined, since no particular normalisation rules have been assumed.

(108) \[\text{da-round}(\text{da}) =\]
\[(\text{is-num-pic}(\text{da}) \rightarrow \text{da-round}\text{-complete-tg}(\text{AR-EA},\text{da})\]
\[(\text{is-arithm} \& \text{is-FIX-s-scale})(\text{da}) \rightarrow\]
\[\text{complete-tg}(\text{eda}(\text{is-arithm} \cdot \text{is-string-type})(\text{da}) \rightarrow \text{da})\]
\[(\text{is-arithm} \& \text{is-string-type})(\text{da}) \rightarrow \text{da}\]

where:
\[\text{max-prec}(\text{da})\]

Ref.:
\[\text{is-num-pic 9-45}(173)\]
\[\text{complete-tg 9-37}(149)\]
\[\text{is-string-type 9-21}(92)\]
\[\text{max-prec 9-12}(69)\]

(109) \[\text{eval\text{-}sign}(\text{op}) =\]
\[\text{FALL}\text{-bintg-op}(\text{sign})\]

where:
\[\text{sign} = (v_1 = 0 \rightarrow 0, \text{T} \rightarrow \text{sign}(v_1))\]
\[v_1 = \text{op-val}(\text{op})\]

Ref.:
\[\text{bintg-op 9-10}(40)\]
\[\text{op-val 9-9}(36)\]

(110) \[\text{eval\text{-}trunc}(\text{op}) =\]
\[\text{val-op-1}(\text{da-trunc}\cdot \text{op}-\text{da}(\text{op}), \text{trunc}\cdot \text{op}-\text{val}(\text{op}))\]

Ref.:
\[\text{op-da 9-9}(37)\]
\[\text{op-val 9-9}(36)\]
(111) \( da\text{-trunc}(da) = \)
\[
\begin{align*}
\text{is-FLT}(s\text{-scale}(da)) & \rightarrow \text{aggr}\text{-scalar}(da) \\
\text{is-FIX}(s\text{-scale}(da)) & \rightarrow \text{complete-tg}(\text{eda-f}(\#,\#,\#;P_0;0),da)
\end{align*}
\]

where:
\[
\begin{align*}
P_0 & = \min(N_0; \max(F_1 - Q_1 + 1,1)) \\
P_1 & = s\text{-prec}(da) \\
Q_1 & = s\text{-scale-f}(da) \\
N_0 & = \max\text{-prec}(da)
\end{align*}
\]

Ref.:  
aggr\text{-scalar} 8-23(64)  
complete\text{-tg} 9-37(149)  
max\text{-prec} 9-12(61)

12.6.2.3 Mathematical generic built-in functions

The built-in functions defined in this section evaluate approximately the values of mathematical functions. The corresponding algorithms are implementation defined and may depend on both the values and the data attributes of the arguments.

The mathematical built-in functions are evaluated by means of the implementation-defined instruction \textit{math-bif}, one part directly, the other indirectly, cf. the table in 12.5.2.

The instruction \textit{math-bif} is defined with the help of a table.

(112) \( \text{eval-ccos}(op) = \)
\[
\begin{align*}
is\text{-REAL}\text{-s-mode}\text{-op-da}(op) & \& -1 \leq \text{op-val}(op) \leq 1 \rightarrow \text{math-bif}(\text{mk-id}(\text{ACCS-REAL}),op) \\
is\text{-CPLX}\text{-s-mode}\text{-op-da}(op) & \rightarrow \text{math-bif}(\text{mk-id}(\text{ACCS-CPLX}),op) \\
T & \rightarrow \text{error}
\end{align*}
\]

Ref.:  
\text{cp-da} 9-9(37)  
\text{cp-val} 9-9(36)

(113) \( \text{eval-asin}(op) = \)
\[
\begin{align*}
is\text{-REAL}\text{-s-mode}\text{-op-da}(op) & \& -1 \leq \text{op-val}(op) \leq 1 \rightarrow \text{math-bif}(\text{mk-id}(\text{ASIN-REAL}),op) \\
is\text{-CPLX}\text{-s-mode}\text{-op-da}(op) & \rightarrow \text{math-bif}(\text{mk-id}(\text{ASIN-CPLX}),op) \\
T & \rightarrow \text{error}
\end{align*}
\]

Ref.:  
\text{cp-da} 9-9(37)  
\text{cp-val} 9-9(36)

52 12. BUILT-IN FUNCTIONS AND PSEUDO-VARIABLES
(114) \( \text{eval-atan}(cp, cp-1) \) =

\[
\begin{align*}
\text{is-Q}(op-1) & \quad \text{is-REAL-s-mode-op-da}(op) \rightarrow \text{math-bif}(\text{mk-id(\text{ATAN-REAL)}}, op) \\
\text{is-Q}(op-1) & \quad \text{is-CPLX-s-mode-op-da}(op) \quad \text{op-val}(op) \neq \text{cpl}(0,1) \quad \text{op-val}(op) \neq \text{cpl}(0,-1) \rightarrow \\
\text{math-bif}(\text{mk-id(\text{ATAN-CPLX)}}, op) \\
\text{is-CPLX-s-mode-op-da}(op) \vee \text{op-val}(op) = 0 \quad \text{op-val}(op) = 0 \rightarrow \text{error} \\
T & \rightarrow \\
\text{math-bif}(\text{mk-id(\text{ATAN-2)}}, op-0, op-10); \quad \text{op-0:convert-1}(eva_0, op), \\
& \quad \text{op-10:convert-1}(eva_0, op-1)
\end{align*}
\]

where:
\( eVa_0 = \text{aggr-scalar}\text{-}\text{pref}(op-da(op), op-da(op-1)) \)

Ref.: op-da 9-9(37)
op-val 9-9(36)
convert-1 9-29(119)
aggr-scalar 8-23(64)

(115) \( \text{eval-atand}(op, op-1) \) =

\[
\begin{align*}
\text{is-Q}(op-1) & \rightarrow \text{math-bif}(\text{mk-id(\text{ATAN-1)}}, op) \\
\text{op-val}(op) = 0 \quad \text{op-val}(op) = 0 \rightarrow \text{error} \\
T & \rightarrow \\
\text{math-bif}(\text{mk-id(\text{ATAN-2)}}, op-0, op-10); \quad \text{op-0:convert-1}(eva_0, op), \\
& \quad \text{op-10:convert-1}(eva_0, op-1)
\end{align*}
\]

where:
\( eVa_0 = \text{aggr-scalar}\text{-}\text{pref}(op-da(op), op-da(op-1)) \)

Ref.: op-val 9-9(36)
convert-1 9-29(119)
aggr-scalar 8-23(64)
op-da 9-9(37)

(116) \( \text{eval-atanh}(op) \) =

\[
\begin{align*}
\text{is-REAL-s-mode-op-da}(op) \quad \text{abs-op-val}(op) < 1 \rightarrow \text{math-bif}(\text{mk-id(\text{ATANH-REAL)}}, op) \\
\text{is-CPLX-s-mode-op-da}(op) \quad \text{op-val}(op) \neq 1 \quad \text{op-val}(op) \neq -1 \rightarrow \\
\text{math-bif}(\text{mk-id(\text{ATANH-CPLX)}}, op) \\
T & \rightarrow \text{error}
\end{align*}
\]

Ref.: op-da 9-9(37)
op-val 9-9(36)

12. BUILT-IN FUNCTIONS AND PSEUDO-VARIABLES 53
(117) \( \text{eval-log}(op) = \)
\[ \text{is-REAL•s-mode•op-da}(op) \land \text{op-val}(op) > 0 \rightarrow \text{math-bif}(	ext{mk-id}(\text{LOG-REAL}), op) \]
\[ \text{is-CPLX•s-mode•op-da}(op) \land \text{op-val}(op) \neq 0 \rightarrow \text{math-bif}(	ext{mk-id}(\text{LOG-CPLX}), op) \]
\[ T \rightarrow \text{error} \]

Ref.: op-da 9-9(37)
op-val 9-9(36)

(118) \( \text{eval-log-10}(op) = \)
\[ \text{op-val}(op) > 0 \rightarrow \text{math-bif}(	ext{mk-id}(\text{LOG-10-1}), op) \]
\[ T \rightarrow \text{error} \]

Ref.: op-val 9-9(36)

(119) \( \text{eval-log-2}(op) = \)
\[ \text{op-val}(op) > 0 \rightarrow \text{math-bif}(	ext{mk-id}(\text{LOG-2-1}), op) \]
\[ T \rightarrow \text{error} \]

Ref.: op-val 9-9(36)

(120) \( \text{eval-sqrt}(op) = \)
\[ \text{is-REAL•s-mode•op-da}(op) \land \text{op-val}(op) \geq 0 \rightarrow \text{math-bif}(	ext{mk-id}(\text{SQRT-REAL}), op) \]
\[ \text{is-CPLX•s-mode•op-da}(op) \rightarrow \text{math-bif}(	ext{mk-id}(\text{SQRT-CPLX}), op) \]
\[ T \rightarrow \text{error} \]

Ref.: op-da 9-9(37)
op-val 9-9(36)

(121) \( \text{erf}(x) = \)
\[ 2 / \text{sqrt}(\pi) \cdot \int_{0}^{x} \text{exp}(-t^2) dt \]

Note: sqrt denotes the positive square root.

(122) \( \text{arctan-2}(x, y) = \)
\[ y > 0 \rightarrow (\text{uv})(x / y = \tan(v) \land -\pi / 2 < v < \pi / 2) \]
\[ y < 0 \rightarrow (\text{uv})(x / y = \tan(v - \pi, \text{sign}(x)) \land -\pi / 2 < v < \pi / 2) \]
\[ y = 0 \rightarrow \pi / 2 \cdot \text{sign}(x) \]

Note: tan denotes the tangent-function.
(123) $\text{math-bif}(\text{id}, \text{cp}, \text{op-1}) =$

\[
\begin{align*}
\text{is-CFLy-s-mode}(\text{da-res}_0) \land \\
(\text{is-size-cond}(\text{da-real}_1, \text{real}(\text{v-res}_0)) \land \\
\text{is-size-cond}(\text{da-real}_1, \text{iseq}(\text{v-res}_0))) \land \\
\text{is-REAL-s-mode}(\text{da-res}_0) \land \text{is-size-cond}(\text{da-res}_0, \text{v-res}_0) \rightarrow \\
\text{ERROR} \\
T \rightarrow \text{PASS} : \text{val-cp}(\text{da-res}_0, \text{v-impl}_0)
\end{align*}
\]

where:

\[
\begin{align*}
\text{da-res}_0 &= (\text{is-}c(\text{op-1}) \rightarrow \text{cp-}\text{da}(\text{op}) \land \\
T \rightarrow \text{ref}(\text{cp-}\text{da}(\text{op}), \text{op-}\text{da}(\text{op-1}))) \\
\text{da-real}_1 &= \text{new} (\text{REAL}, \text{da-res}_0)
\end{align*}
\]

for: $\text{is-id}(\text{id}), \text{is-cp}(\text{op}), (\text{is-cp} \land \text{is-cp}) (\text{op-1}), \text{s-eva}(\text{cp}) = \\
\text{s-eva}(\text{op-1}), \text{is-FLT-s-scale op-da}(\text{cp})$

Ref.: $\text{is-size-cond} 8-8(26)$

$\text{val-op} 9-9(38)$

$\text{cp-da} 9-9(37)$

$\text{is-cp} 9-9(34)$

Note: $\text{v-res}_0$ is to be found in that line of the table designated by (the concrete representation of) the identifier $\text{id}$; it is the value of the mathematical expression contained in the second column of the table, satisfying the condition stated in the third column, if the expression is not unique. The function names in column two have their usual mathematical meaning.

$v-\text{impl}_0$ is an implementation-dependent approximation of $v-res_0$.

Abbreviations used in the table:

\[
\begin{align*}
v_0 &= \text{cp-val}(\text{op}) \\
v_1 &= \text{cp-val}(\text{op-1}) \\
v-res_0 &= \text{see above}
\end{align*}
\]
<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACOS-REAL</td>
<td>( \arccos(v_0) )</td>
<td>( 0 \leq v_{\text{res}_0} \leq \pi )</td>
</tr>
<tr>
<td>ACOS-CPLX</td>
<td>( \arccos(v_0) )</td>
<td>( -\pi \leq \text{real}(v_{\text{res}_0}) &lt; \pi )</td>
</tr>
<tr>
<td>ASIN-REAL</td>
<td>( \arcsin(v_0) )</td>
<td>( -\pi/2 \leq \text{real}(v_{\text{res}_0}) \leq \pi/2 )</td>
</tr>
<tr>
<td>ASIN-CPLX</td>
<td>( \arcsin(v_0) )</td>
<td>( -\pi/2 &lt; \text{real}(v_{\text{res}_0}) \leq \pi/2 )</td>
</tr>
<tr>
<td>ATAN-REAL</td>
<td>( \arctan(v_0) )</td>
<td>( -\pi/2 &lt; \text{real}(v_{\text{res}_0}) &lt; \pi/2 )</td>
</tr>
<tr>
<td>ATAN-CPLX</td>
<td>( \arctan(v_0) )</td>
<td>( -\pi/2 &lt; \text{real}(v_{\text{res}_0}) \leq \pi/2 )</td>
</tr>
<tr>
<td>ATAN-2</td>
<td>( \arctan-2(v_0,v_1) )</td>
<td></td>
</tr>
<tr>
<td>ATAND-1</td>
<td>( \arctan(v_0) \cdot 180/\pi )</td>
<td>( -90 &lt; v_{\text{res}_0} &lt; 90 )</td>
</tr>
<tr>
<td>ATAND-2</td>
<td>( \arctan-2(v_0,v_1) \cdot 190/\pi )</td>
<td></td>
</tr>
<tr>
<td>ATANF-REAL</td>
<td>( \arctanh(v_0) )</td>
<td>( -\pi/2 &lt; \text{imag}(v_{\text{res}_0}) \leq \pi/2 )</td>
</tr>
<tr>
<td>ATANF-CPLX</td>
<td>( \arctanh(v_0) )</td>
<td></td>
</tr>
<tr>
<td>CCS</td>
<td>( \csc(v_0) )</td>
<td></td>
</tr>
<tr>
<td>CCSD</td>
<td>( \csc(v_0 \cdot \pi/180) )</td>
<td></td>
</tr>
<tr>
<td>CSCH</td>
<td>( \csch(v_0) )</td>
<td></td>
</tr>
<tr>
<td>ERF</td>
<td>( \text{erf}(v_0) )</td>
<td></td>
</tr>
<tr>
<td>ERFC</td>
<td>( 1 - \text{erf}(v_0) )</td>
<td></td>
</tr>
<tr>
<td>EXP</td>
<td>( \exp(v_0) )</td>
<td></td>
</tr>
<tr>
<td>LGT-REAL</td>
<td>( \log(v_0) )</td>
<td>( -\pi &lt; \text{imag}(v_{\text{res}_0}) \leq \pi )</td>
</tr>
<tr>
<td>LGT-CPLX</td>
<td>( \log(v_0) )</td>
<td></td>
</tr>
<tr>
<td>LGT-10-1</td>
<td>( \log_{10}(v_0) )</td>
<td></td>
</tr>
<tr>
<td>LGT-2-1</td>
<td>( \log_{2}(v_0) )</td>
<td></td>
</tr>
<tr>
<td>SIN</td>
<td>( \sin(v_0) )</td>
<td></td>
</tr>
<tr>
<td>SIND</td>
<td>( \sin(v_0 \cdot \pi/180) )</td>
<td></td>
</tr>
<tr>
<td>SINH</td>
<td>( \sinh(v_0) )</td>
<td></td>
</tr>
<tr>
<td>SCPT-REAL</td>
<td>( \sct(v_0) )</td>
<td>( v_{\text{res}_0} \geq 0 )</td>
</tr>
<tr>
<td>SCPT-CPLX</td>
<td>( \sct(v_0) )</td>
<td>( \text{real}(v_{\text{res}<em>0}) \geq 0 ) ( \text{v} ) ( \text{real}(v</em>{\text{res}<em>0}) = 0 ) ( \text{&amp; imag}(v</em>{\text{res}_0}) \geq 0 )</td>
</tr>
<tr>
<td>TAN</td>
<td>( \tan(v_0) )</td>
<td></td>
</tr>
<tr>
<td>TAND</td>
<td>( \tan(v_0 \cdot \pi/180) )</td>
<td></td>
</tr>
<tr>
<td>TANF</td>
<td>( \tanh(v_0) )</td>
<td></td>
</tr>
</tbody>
</table>
12.6.2.4 Generic built-in functions for array manipulation

(124) \texttt{eval-all(opl) =}
\begin{align*}
\texttt{iterate-infix-op(head(opl), tail(opl), AND)}
\end{align*}
\texttt{for: is-real-val(x), is-real-val(y)}
\texttt{Ref.: is-real-val 9-3(4)}

(125) \texttt{da-all(da) =}
\begin{align*}
\mu(\texttt{BIT-IDV;S-varying:S-varying(da)})
\end{align*}

(126) \texttt{eval-any(opl) =}
\begin{align*}
\texttt{iterate-infix-op(head(opl), tail(opl), OR)}
\end{align*}

(127) \texttt{eval-dim(eva,i) =}
\begin{align*}
is-array(eva) \& i = 1 & \rightarrow \texttt{pass-bintg-op}(s-ubd(eva) - s-lbd(eva) + 1) \\
is-array(eva) \& i > 1 & \rightarrow \texttt{eval-dim(s-elem(eva),i - 1)}
\end{align*}
\texttt{T \rightarrow error}
\texttt{Ref.: bintg-op 9-10(40)}

(128) \texttt{eval-hbound(eva,i) =}
\begin{align*}
is-array(eva) \& i = 1 & \rightarrow \texttt{pass-bintg-op}(s-ubd(eva)) \\
is-array(eva) \& i > 1 & \rightarrow \texttt{eval-hbound(s-elem(eva),i - 1)}
\end{align*}
\texttt{T \rightarrow error}
\texttt{Ref.: bintg-op 9-10(40)}

(129) \texttt{eval-lbound(eva,i) =}
\begin{align*}
is-array(eva) \& i = 1 & \rightarrow \texttt{pass-bintg-op}(s-lbd(eva)) \\
is-array(eva) \& i > 1 & \rightarrow \texttt{eval-lbound(s-elem(eva),i - 1)}
\end{align*}
\texttt{T \rightarrow error}
\texttt{Ref.: bintg-op 9-10(40)}
(130) \texttt{eval-rcly}(opl-1,op-1,op-2) = \\
\texttt{eval-rcly-1}(op-1,op-2,\text{tail}(op-1),op-2); \\
\texttt{op-1}: \texttt{convert-1}(\text{ev}_{\text{a}}_{0}\text{head}(op-1)), \\
\texttt{op-2}: \texttt{eval-or-1}(da_{0},1) \\
\text{where:} \\
da_{0} = \text{pref}(op-da\cdot\text{head}(op-1),op-da\cdot\text{head}(op-2)) \\
\text{ev}_{\text{a}}_{0} = \text{aggr-scalar}(da_{0}) \\
\text{Ref.:} \texttt{convert-1} 9-25(119) \\
op-da 8-9(37) \\
\text{aggr-scalar} 8-23(64) \\

(131) \texttt{eval-rcly}(op-1,op-2,op-1,op-2) = \\
is->(op-1) \rightarrow \text{PASS:op-1} \\
T \rightarrow \\
\texttt{eval-rcly-1}(op-11,op-21,\text{tail}(op-1),\text{tail}_{2}); \\
op-11: \texttt{infix-op}(op-1,op-0,\text{AED}); \\
op-0: \texttt{eval-infix-expr}(op-2,\text{head}(op-1,\text{MULT})); \\
op-21: \texttt{eval-infix-expr}(op-2,\text{head}(op-2,\text{MULT}) \\
\text{where:} \\
tail_{2} = (\text{length}(op-2) > 1 \rightarrow \text{tail}(op-2), \\
T \rightarrow op-2) \\
\text{Ref.:} \texttt{infix-op} 9-13(65) \\
\texttt{eval-infix-expr} 9-13(64) \\

(132) da-poly(da-1,da-2) = \\
\text{pref}(da-c_{1},da-c_{2}) \\
\text{where:} \\
da-c_{1} = \text{complete-tg}(\text{FTL-EDA},da-1) \\
da-c_{2} = \text{complete-tg}(\text{FTL-EDA},da-2) \\
\text{Ref.:} \text{complete-tg} 9-37(149) \\

(133) \texttt{eval-prod}(cpl) = \\
\texttt{iterate-infix-op}(\text{head}(c-\text{cpl})\text{tail}(c-\text{cpl}),\text{MULT}); \\
\{\text{elem}(1)(c-\text{cpl}): \texttt{convert-1}(\text{ev}_{\text{a}}_{1}\text{elem}(i,\text{cpl})) \mid 1 \leq i \leq \text{length}(\text{cpl})\} \\
\text{where:} \\
\text{ev}_{\text{a}}_{1} = \text{aggr-scalar}\cdot da-prod\cdot op-da\cdot head(\text{cpl}) \\
\text{Ref.:} \texttt{convert-1} 9-29(119) \\
\text{aggr-scalar} 8-23(64) \\
op-da 8-9(37) \\

58 12. BUILT-IN FUNCTIONS AND PSEUDO-VARIABLES
(134) \( \text{da-prod}(\text{da}) = \)
\[ \text{is-\text{FIX}*s-scale}(\text{da}) \& s\text{-scale-f}(\text{da}) = 0 \rightarrow \text{complete-tg}(\text{eda-f}(\ast,\ast,\text{FIX},N_1,0),\text{da}) \]
\( \text{T} \rightarrow \text{complete-tg}(\text{eda-f}(\ast,\ast,\text{FLT},*),\text{da}) \)

where:
\( N_1 = \text{max-prec}(\text{da}) \)

Ref.:  
\text{complete-tg} 9-37(149)  
\text{max-prec} 9-12(61)

(135) \( \text{eval-sum}(\text{cpl}) = \)
\[ \text{iterate-infix-op}(\text{head}(\text{cpl}),\text{tail}(\text{cpl}),\text{ADD}) ; \]
\[ \{ \text{elem}(1)(\text{cpl}) : \text{convert-1}(\text{eva}_1,\text{elem}(1,\text{cpl})) \mid 1 \leq i \leq \text{length}(\text{cpl}) \} \]

where:
\( \text{eva}_1 = \text{aggr-scalar*da-sum*op-da*head}(\text{cpl}) \)

Ref.:  
\text{convert-1} 9-25(119)  
\text{aggr-scalar} 8-23(64)  
\text{op-da} 9-9(37)

(136) \( \text{da-sur}(\text{da}) = \)
\[ \text{is-\text{FIX}*s-scale}(\text{da}) \rightarrow \text{complete-tg}(\text{eda-f}(\ast,\ast,\ast,N_1,\ast),\text{da}) \]
\( \text{T} \rightarrow \text{da} \)

where:
\( N_1 = \text{max-prec}(\text{da}) \)

Ref.:  
\text{complete-tg} 9-37(149)  
\text{max-prec} 9-12(61)

(137) \( \text{iterate-infix-op}(\text{op},\text{cpl},\text{opr}) = \)
\[ \text{is-\text{<}}(\text{cpl}) \rightarrow \text{PASS}:\text{op} \]
\( \text{T} \rightarrow \)
\[ \text{iterate-infix-op}(\text{op-1},\text{tail}(\text{cpl}),\text{opr}) ; \]
\[ \text{op-1}:\text{infix-op}(\text{op},\text{head}(\text{cpl}),\text{opr}) \]

for: is-infix-opr(\text{opr})

Ref.:  
\text{infix-op} 9-13(65)

12.6.2.5 Condition built-in functions

The values of condition built-in functions, if used in an op-unit, are determined from the contents of the condition built-in function part of the condition state \( s\text{-chif}(\text{CS}) \), cf. 3.5.5 and 10.5. If used outside op-units, the functions return standard values. For the function \( \text{chl}(\text{id}) \), cf. 11.6.3.2.
(138) \texttt{eval-datafield} =
\begin{align*}
is-\text{get-datafield}\&-\text{cs}(CS) \rightarrow \text{PASS:val-op(char-eda(0), \langle \rangle)} \\
T \rightarrow \text{PASS:val-op(char-eda(TATAI-LENGTH), s-datafield}\&-\text{cs}(CS))
\end{align*}
\text{Ref.:} \quad \text{val-op} \ 9-9(38) \\
\quad \text{char-eda} \ 9-12(58)

(139) \texttt{eval-cnattr} =
\begin{align*}
is-\text{get-cnattr}\&-\text{cs}(CS) \rightarrow \text{PASS:val-op(PTR, WPTR)} \\
T \rightarrow \text{PASS:val-op(PTR, s-cnattr}\&-\text{cs}(CS))
\end{align*}
\text{Ref.:} \quad \text{val-op} \ 9-9(38)

(140) \texttt{eval-cnchar} =
\begin{align*}
is-\text{get-cnchar}\&-\text{cs}(CS) \rightarrow \text{PASS:val-op(char-eda(1), \langle ELANE \rangle)} \\
T \rightarrow \text{\texttt{pass-test-cnchar} (op, s-cnchar}\&-\text{cs}(CS)); \\
\quad \text{cp:eval-cnsource}
\end{align*}
\text{Ref.:} \quad \text{val-op} \ 9-9(38) \\
\quad \text{char-eda} \ 9-12(58)

(141) \texttt{test-cnchar(op, i)} =
\begin{align*}
\text{str-length} \&-\text{op-da}(op) < i & \rightarrow \text{error} \\
T \rightarrow &
\begin{align*}
\mu_0(<\langle s-eva:substr-eva(s-eva(op), i, 1) \rangle, \\
\langle s-vr:rep(s-eva(op), elem(i, op-val(op))) \rangle >)
\end{align*}
\end{align*}
\text{Ref.:} \quad \text{str-length} \ 9-41(164) \\
\quad \text{op-da} \ 9-9(37) \\
\quad \text{substr-eva} \ 12-71(180) \\
\quad \text{rep} \ 9-6(19) \\
\quad \text{op-val} \ 9-9(36)

(142) \texttt{eval-cnencode} =
\begin{align*}
is-\text{get-cnencode}\&-\text{cs}(CS) \rightarrow \text{PASS:hintg-op(0)} \\
T \rightarrow \text{PASS:hintg-op(s-cnencode}\&-\text{cs}(CS))
\end{align*}
\text{Ref.:} \quad \text{hintg-op} \ 9-10(80)
(143) \( \text{eval-opcount} = \)
\[ \text{is-\&s-opcount\&s-chif(CS)} \rightarrow \text{PASS:hintg-op}(O) \]
\[ T \rightarrow \text{PASS:hintg-op}(s-opcount\&s-chif(CS)) \]
Ref.: hintg-op 9-10(40)

(144) \( \text{eval-opfile} = \)
\[ \text{is-\&s-opfile\&s-chif(CS)} \rightarrow \text{PASS:val-op}(\text{char-eda}(O),<>) \]
\[ T \rightarrow \text{PASS:val-op}(\text{char-eda}(ID-LENGTH),chl\&s-opfile\&s-chif(CS)) \]
Ref.: val-op 9-9(38)
char-eda 9-12(56)
chl 17-89(244)

(145) \( \text{eval-oident} = \)
\[ \text{is-\&s-oident\&s-chif(CS)} \rightarrow \text{PASS:val-op}(\text{char-eda-var}(O),<>) \]
\[ T \rightarrow \text{gen-op}(s-gen\&s-oident\&s-chif(CS)) \]
Ref.: val-op 9-9(38)
char-eda-var 9-12(60)
gen-op 6-22(59)

(146) \( \text{eval-onkey} = \)
\[ \text{is-\&s-onkey\&s-chif(CS)} \rightarrow \text{PASS:val-op}(\text{char-eda-var}(O),<>) \]
\[ T \rightarrow \text{PASS:val-op}(\text{char-eda-var}(ONY-LENGTH),s-onkey\&s-chif(CS)) \]
Ref.: val-op 9-9(38)
char-eda-var 9-12(60)

(147) \( \text{eval-oloc} = \)
\[ \text{is-\&s-oloc\&s-chif(CS)} \rightarrow \text{PASS:val-op}(\text{char-eda}(O),<>) \]
\[ T \rightarrow \text{PASS:val-op}(\text{char-eda}(ID-LENGTH),chl\&s-oloc\&s-chif(CS)) \]
Ref.: val-op 9-9(38)
char-eda 9-12(56)
chl 17-89(244)

(148) \( \text{eval-onsource} = \)
\[ \text{is-\&s-onsource\&s-chif(CS)} \rightarrow \text{PASS:val-op}(\text{char-eda}(O),<>) \]
\[ \text{is-gen\&s-onsource\&s-chif(CS)} \rightarrow \text{gen-op}(s-onsource\&s-chif(CS)) \]
\[ T \rightarrow \text{eval-onsource-substr}(s-onsource\&s-chif(CS)) \]
cont'd
ABSTRACT SYNTAX AND INTERPRETATION OF PL/I

Ref.: val-cp 9-9(36)
      char-eda 9-12(58)
      is-gen 3-14(3C)
      gen-op 8-22(59)

(149) eval-cons-source-substr(g) =
       eval-substr-1(op, elem(1,s-arg-list(g)), elem(2, s-arg-list(g)),
        elem(3, s-arg-list(g)), NOSTRG);
        cp: gen-cp(elem(1,s-arg-list(g)))

for: is-ps-gen(g) & is-SUBSTR-s-id(g)

Ref.: gen-op 8-22(59)
       is-ps-gen 12-67(169)

12.6.2.6 Based storage built-in functions

(150) eval-addr(ref) =
       is-var-ref(ref,AT) & is-connected(ref,AT) & is-CTRLs-stg-cl(s-n(ref)(AT)) &
       -is-gen-s-head(s-n(ref)(AT)) --
       PASS:val-op(PTR,NPTR)
       is-var-ref(ref,AT) & is-connected(ref,AT) --
       PASS:val-op(PTR,NPTR)
       F:eval-s-op(gen);
       gen:eval-ref-gen(ref)
       T -- ERROR

Ref.: is-var-ref 8-9(21)
       is-connected 8-34(109)
       is-gen 3-14(30)
       val-op 9-9(36)
       eval-ref-gen 8-28(82)

(151) eval-empty =
       PASS:EMPTY-OP

(152) is-op(EMPTY-OP) & is-area-op-da(EMPTY-OP) & is-ALLOC-state-s-vi(EMPTY-OP)

Ref.: is-op 9-9(34)
      op-da 9-9(37)
      alloc-state 3-16(53)

Note: EMPTY-OP is the operand corresponding to the empty area.

(153) eval-null =
       PASS:val-op(PTE,NPTR)

cont'd
Ref.: val-op 9-9(38)

(154) \texttt{eval-nullo} = \texttt{PASS: val-op (OFFSET, NPTR)}

Ref.: val-op 9-9(38)

(155) \texttt{is-ptr-val(NPTR) \& (Wz) \& is-applic(NPTR, z)}

for: \texttt{is-size(z)}

Ref.: is-ptr-val 3-15(36)
\texttt{is-applic 3-15(37)}
\texttt{is-size 3-15(34)}

Note: \texttt{NPTR} is that pointer which is not applicable to any storage.

(156) \texttt{eval-offset(op, gen) =}

\texttt{is-PTR \& op-da (op) \& is-area \& gen-da (gen) \&}
\texttt{(\exists o)(P_1 = o \& P_1 \& o \in alloc-state \& p-a_1 (\exists)) \rightarrow}
\texttt{PASS: val-op (OFFSET, (UO) (p_1 = o \& p-a_1))}

\texttt{T \rightarrow error}

where:
\texttt{P_1 = op-val (op)}
\texttt{P-a_1 = s-pp (gen)}

Ref.: op-da 9-9(37)
\texttt{gen-da 7-17(53)}
alloc-state 3-16(53)
val-op 9-9(38)
\texttt{op-val 9-9(36)}

(157) \texttt{eval-ptr(op, gen) =}

\texttt{is-OFFSET \& op-da (op) \& is-applic (c_x, size-1 (p-a_1)) \rightarrow PASS: val-op (PTR, c_x \& p-a_1)}

\texttt{T \rightarrow error}

where:
\texttt{c_x = op-val (op)}
\texttt{p-a_1 = s-pp (gen)}

Ref.: op-da 9-9(37)
\texttt{is-applic 3-15(37)}
size-1 3-15(38)
val-op 9-9(38)
\texttt{op-val 9-9(36)}
12.6.2.7 Multitasking built-in functions

(158) \(\text{eval-ccmpare}(\text{op}) =\)
\[\text{PASS} : \text{val-op}(\text{bit-eda}(1), \langle s - \text{cosp} \star \text{op-val}(\text{op}) \rangle)\]

Ref.: val-cp 9-9(38)
bit-eda 9-12(59)
cr-val 9-9(36)

(159) \(\text{eval-priority}(\text{op}) =\)
\[\text{convert-1}(\text{PRI-EVA}, \text{op-r});\]
\[\text{op-r} : \text{eval-infix-expr}(\text{op-1}, \text{op-2}, \text{SUBTB});\]
\[\text{cr-1} : \text{convert-pret}(\text{PRI-EVA}, \text{op}),\]
\[\text{cr-2} : \text{convert-pret}(\text{PRI-EVA}, \text{op-2});\]
\[\text{cr-21} : \text{gen-op}(s - tv(TT))\]

Ref.: convert-1 9-29(115)
eval-infix-expr 9-13(64)
convert-pret 5-3(9)
gen-op 8-22(59)

(160) \(\text{eval-status}(\text{cr}) =\)
\[-i - G(\text{op}) \rightarrow \text{PASS} : \text{bintg-op} \star \text{s-status} \star \text{op-val}(\text{op})\]
\[T \rightarrow \]
\[\text{PASS} : \text{bintg-op} (s - \text{status} \star \text{op-val}(\text{op}-1));\]
\[\text{op-1} : \text{gen-op}(s - \text{ev}(TT))\]

Ref.: bintg-op 9-10(40)
op-val 9-9(36)
gen-op 8-22(59)

12.6.2.8 Miscellaneous built-in functions

(161) \(\text{eval-allocate}(\text{t}) =\)
\[-i - \text{ctl-ref}(\text{t}, AT) \rightarrow \text{error}\]
\[\text{is-gen} \star \text{list}(\text{genl} 1) \rightarrow \text{PASS} : \text{bintg-op} \star \text{length}(\text{genl} 1)\]
\[T \rightarrow \text{PASS} : \text{bintg-op} (\text{length}(\text{genl} 1) - 1)\]

where:
\[\text{genl} 1 = s - n(t) (EN)(AG)\]

Ref.: is-ctl-ref 6-19(50)
is-gen 3-14(36)
bintg-op 9-10(40)

64 12. BUILT-IN FUNCTIONS AND PSEUDO-VARIABLES
(162) \texttt{eval-count(op) =}
\begin{verbatim}
    file-bif(s-count, assoc-u(op-val(op) (DN,M)))
\end{verbatim}
Ref.:
\begin{itemize}
    \item assoc-u 11-24(70)
    \item op-val 9-9(36)
\end{itemize}

(163) \texttt{eval-date =}
\begin{verbatim}
    is-valid-date(s-date(DP)) \rightarrow
    PASS:val-op(char-eda(6), num-char-list(6, s-date(DP)))
\end{verbatim}
Ref.:
\begin{itemize}
    \item val-op 9-9(36)
    \item char-eda 9-12(58)
    \item num-char-list 9-32(128)
\end{itemize}

(164) \texttt{is-valid-date(d) =}
\begin{verbatim}
    d > 0 \land
    (\text{month} \in \{1,3,5,7,10,12\} \land 1 \leq \text{day} \leq 31 \land
     \text{month} \in \{4,6,9,11\} \land
     1 \leq \text{day} \leq 30 \lor
     \text{month} = 2 \land
     1 \leq \text{day} \leq 28 \land
     \text{modulo} (\text{year}, 4) = 0 \land
     \text{month} = 2 \land
     \text{day} = 29)
\end{verbatim}
where:
\begin{itemize}
    \item \text{year} = \text{trunc}(d / 10000)
    \item \text{month} = \text{modulo} (\text{trunc}(d / 100), 10)
    \item \text{day} = \text{modulo}(d, 100)
\end{itemize}
for: \texttt{is-real-val(d)}
Ref.:
\texttt{is-real-val 9-3(4)}

Note: \textit{It is assumed, that the program is interpreted between March 1, 1900 and February 28, 2100.}

(165) \texttt{eval-enabled(ref) =}
\begin{verbatim}
    \neg (\text{is-ref(ref)} \land \text{length-s-id-list(ref)} = 1 \land
    \text{is-<s-ptr(ref)} \land \text{is-<s-sl(ref)} \land
    \text{is-<>s-ap(ref)} \land \text{is-attention(s-n(ref) (AT)))}) \rightarrow
    \text{ERROR}
\end{verbatim}
\begin{verbatim}
    (\exists x, i) (1 \leq i \leq \text{length-s-enab-list(FW)} \land
    x = \text{elem}(i, \text{s-enab-list(FW)})) \land
    \text{ident-ref-attn}_1 = \text{id}(x)) \rightarrow
    \text{PASS:val-op(bit-eda(1),1-BIT)}
\end{verbatim}
\begin{verbatim}
    T \rightarrow \text{PASS:val-op(bit-eda(1),0-BIT)}
\end{verbatim}
where:
\begin{itemize}
    \item \text{ref-attn}_1 = \text{eval-attn}(\mu_0 (\langle \text{s-id:head} \rangle \text{s-id-list(ref)}, \text{s-n: n(ref)}), DN,M)
\end{itemize}

cont'd
12.7 PSEUDO-VARIABLES

12.7.1 ASSIGNMENT TO PSEUDO-VARIABLES

References to pseudo-variables in the left part of an assignment statement are evaluated by the instruction `eval-ps-lp(ref), where ref is the text of the reference. The evaluation results in a pseudo-generation. A pseudo-generation satisfies the predicate is-ps-gen:

A pseudo-generation contains the identifier of the pseudo-variable and the list of evaluated arguments.

The assignment of an operand op to a pseudo-generation ps-gen is performed by the instruction `pseudo-assign (ps-gen, op). The definition of this instruction contains the case distinctions for the different PL/I pseudo-variables.
Abstract Syntax and Interpretation of PL/I

**Variables**

<table>
<thead>
<tr>
<th>ref</th>
<th>is-ref</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>is-id</td>
</tr>
<tr>
<td>cp</td>
<td>is-cp</td>
</tr>
<tr>
<td>list</td>
<td>is-list</td>
</tr>
<tr>
<td>gen</td>
<td>is-gen</td>
</tr>
<tr>
<td>ps-gen</td>
<td>is-ps-gen</td>
</tr>
<tr>
<td>ipv</td>
<td>is-char-val-list ∨ is-bit-val-list</td>
</tr>
<tr>
<td>i, j</td>
<td>is-int-val ∨ is-q</td>
</tr>
</tbody>
</table>

(169) is-ps-gen =

(\langle s-id:is-id, s-arg-list:is-list\rangle)

(170) \texttt{eval-ps-lp(ref) =}

(\langle i(id_0 = \text{mk-id(COMPLEX)} \land \text{id_0 = mk-id(CPRX)}) \land \text{length(al_0) = 2} \land \rangle
(\langle s-id:is-id, s-arg-list:is-list\rangle)

(171) \texttt{mk-ps-gen(id_0, list);}

\texttt{\{elem(1)(list):eval-ref-gen-1(elem(i, al_0)) \mid 1 \leq i \leq length(al_0)\}}

\texttt{id_0 = mk-id(SUBSTR) \land (2 \leq length(al_0) \leq 3) \land is-var-ref(head(al_0), &T)} \rightarrow

(172) \texttt{mk-ps-gen(id_0, list);}

\texttt{\{elem(1)(list):eval-ref-gen-1(elem(i, al_0)), elem(2)(list):eval-intg-expr-1(elem(2, al_0)), elem(3)(list):eval-intg-expr-1(elem(3, al_0))}\}}

\texttt{id_0 = mk-id(STRING) \land length(al_0) = 1 \rightarrow}

(173) \texttt{mk-ps-gen(id_0, gen-1);}

\texttt{gen-1:rasp-gen-list(gen);}

\texttt{\{elem(1, al_0))\}}

(\langle i(id_0 = \text{mk-id(CPRX)}) \lor \text{id_0 = mk-id(ONCHAR)} \lor \text{id_0 = mk-id(OMIDENT)}) \land
\texttt{length(al_0) = 0 \rightarrow}

(174) \texttt{mk-ps-gen(id_0, <s-ev(TF)>)}

\texttt{id_0 = mk-id(STATUS) \land length(al_0) = 0 \rightarrow mk-ps-gen(id_0, <s-tv(TF)>)}

\texttt{T \rightarrow Err}

\texttt{where:}

\texttt{id_0 = heads-s-id-list(ref)}

\texttt{al_0 = heads-s-ar(ref)}

\texttt{cert'd}


Ref.: eval-ref-gen-1 8-28(83)
     is-var-ref 8-9(21)
     eval-intg-expr-1 12-6(18)
     gen-list 5-11(33)

(171) \( \text{mk-ps-gen(id, list)} = \)
     PASS: \( \mu_0(<s-id:id>, <s-arg-list:list>) \)

(172) \( \text{pseudo-assign(ps-gen, op) =} \)
     \( \begin{align*}
         \text{id}_0 &= \text{mk-id(COMPLEX) } \lor \text{id}_0 = \text{mk-id(CPLX)} - \\
         &\text{assign-complex} (\text{gen}_1, \text{gen}_2, \text{op-1}); \\
         &\text{op-1: convert-1}(\mu(\text{aggr-scalar} (\text{AR-IDA}); <s-mode-da:CPLX>), \text{op}) \\
         \text{id}_0 &= \text{mk-id(RNUM)} \land \text{is-CPLX}s-mode*gen-da(\text{gen}_1) - \\
         &\text{assign-real}(\text{gen}_1, \text{op-1}); \\
         &\text{op-1: convert-1}(\mu(\text{p-eva}(\text{gen}_1); <s-mode-da:REAL>), \text{op}) \\
         \text{id}_0 &= \text{mk-id(RND)} \land \text{is-CPLX}s-mode*gen-da(\text{gen}_1) - \\
         &\text{assign-real}(\text{gen}_1, \text{op-1}); \\
         &\text{op-1: convert-1}(\mu(\text{p-eva}(\text{gen}_1); <s-mode-da:REAL>), \text{op})
     \end{align*} \)

T \( \leftarrow \) \( \text{pseudo-assign-contin}(\text{ps-gen, op}) \)

Where:
     \( \text{id}_0 = \text{s-id(ps-gen)} \)
     \( \text{gen}_1 = \text{elem}(1, s-arg-list(ps-gen)) \)
     \( \text{gen}_2 = \text{elem}(2, s-arg-list(ps-gen)) \)

Ref.: convert-1 9-29(119)
     aggr-scalar 8-23(64)
     gen-da 7-17(53)

(173) \( \text{pseudo-assign-contin}(\text{ps-gen, op}) = \)
     \( \begin{align*}
         \text{id}_0 &= \text{mk-id(UNSPEC)} \land \\
         &\text{(is-num-type } \lor \text{is-string-type } \lor \text{is-area } \lor \text{is-PTR } \lor \text{is-OFFSET})(\text{gen-da}(\text{gen}_1)) - \\
         &\text{assign}(\text{gen}_1, \text{op-1}); \\
         &\text{op-1: convert-1}(\text{str-eva} (\text{BIT, unspec-length} = \text{alloc-size} = \text{eva}(\text{gen}_1)), \text{op}) \\
         \text{id}_0 &= \text{mk-id(SUBSTR)} \land \text{is-string-type} \land \text{gen-da}(\text{gen}_1) - \\
         &\text{assign-substr}(\text{gen}_1, \text{lpv}, \text{elem}(2, s-arg-list(ps-gen)), \text{elem}(3, s-arg-list(ps-gen)), \text{op,0}); \\
         &\text{lpv: eva}(\text{gen}_1) \\
         \text{id}_0 &= \text{mk-id(STRING)} \land \\
         &\text{id}_0 \\
         &\text{str: eva}(\text{BIT}) - \\
         &\text{id}_0 \\
         &\text{str: eva}(\text{CHR}) - \\
         &\text{id}_0 \\
         &\text{str: eva}(\text{BIT}) - \\
         &\text{id}_0 \\
     \end{align*} \)

\( \text{assign-strn}(1, s-arg-list(ps-gen), \text{str}); \\
     \text{str: eva}(\text{BIT, op-1}); \\
     \text{op-1: convert-1}(\text{eva}_1, \text{op}) \)

\( \text{cont'd} \)
T ::= \textit{pseudo-assign- condi}(ps-gen, op)

where:
\begin{align*}
\text{id}_0 &= \text{s-id}(ps-gen) \\
\text{gen}_0 &= \text{else}(1, s-arg-list(ps-gen)) \\
\text{ly}_0 &= \text{lengths-arg-list}(ps-gen) \\
\text{eval} &= (\text{n-base-gen-da}(\text{gen}_1) = \text{CHAR} \rightarrow \text{aggr-scalar}(\text{CHAR-EQA}), \\
T &= \text{aggr-scalar}(\text{BIT-EQA}))
\end{align*}

Ref.:
\begin{align*}
is-num-type &= 9-21(91) \\
is-string-type &= 9-21(92) \\
gen-da &= 7-17(53) \\
assign &= 8-9(23) \\
convert &= 9-29(119) \\
str-eval &= 7-14(45) \\
unoc-length &= 12-42(71) \\
alloc-size &= 7-17(26) \\
op-val &= 9-9(36) \\
aggr-scalar &= 8-23(64)
\end{align*}

(174) \text{pseudo-assign- condi}(ps-gen, op) =
\begin{align*}
\text{id}_0 &= \text{mk-id}(\text{ONSOURC}) \land \text{id-gen}(g_0) \rightarrow \text{convert-assign}(g_0, \text{op}, 9, 0) \\
\text{id}_0 &= \text{mk-id}(\text{ONSOURC}) \land \text{s-id}(g_0) = \text{mk-id}(\text{SUBST}) \rightarrow \\
\text{assign- subst}(g_1, \text{lpv}, \text{lo}_4, \text{lg}_4, \text{op}, \text{NO-STRG}); \\
\text{lpv} : \text{lpv-val}(g_4) \\
\text{id}_0 &= \text{mk-id}(\text{ONCHAR}) \land \text{id-gen}(y_0) \rightarrow \\
\text{assign- subst}(g_0, \text{lpv}, \text{onchar-s-chif}(\text{CG}), 1, \text{NO-STRG}); \\
\text{lpv} : \text{lpv-val}(g_6) \\
\text{id}_0 &= \text{mk-id}(\text{ONCHAR}) \land \text{s-id}(g_0) = \text{mk-id}(\text{SUBST}) \rightarrow \\
\text{assign- subst}(g_1, \text{lpv}, \text{lo}_4 + \text{onchar-s-chif}(\text{CG}) = 1, 1, \text{op}, \text{NO-STRG}); \\
\text{lpv} : \text{lpv-val}(g_4) \\
\text{id}_0 &= \text{mk-id}(\text{ONIDENT}) \land \text{id-gen}(\text{onil}_1) \rightarrow \\
\text{convert-assign}(\text{gen}(\text{onil}_1), \text{op}, 1, 1) \\
\text{id}_0 &= \text{mk-id}(\text{COMPL}) \lor \text{id}_0 = \text{mk-id}(\text{CPLN}) \rightarrow \\
\text{assign- completion}(\text{gen}_1, \text{op}-1); \\
\text{op}-1: \text{convert-1}(\text{aggr-scalar BIT-EDA}(1), \text{op}) \\
\text{id}_0 &= \text{mk-id}(\text{PRIORITY}) \rightarrow \\
\text{assign- priority}(\text{gen}_1, \text{op}-1); \\
\text{op}-1: \text{convert-1}(\text{PRI-BDA}, \text{op}) \\
\text{id}_0 &= \text{mk-id}(\text{STAT}) \rightarrow \\
\text{assign- status}(\text{gen}_1, \text{op}-1); \\
\text{op}-1: \text{convert-1}(\text{aggr-scalar BITG-EDA}, \text{op}) \\
T &= \text{error}
\end{align*}
where:

\[ id_0 = s-id(ps-gen) \]
\[ gen_1 = \text{elem}(1, s-arg-list(ps-gen)) \]
\[ q_0 = s-onsource\cdot s-chif(CS) \]
\[ g_1 = \text{elem}(1, s-arg-list(q_0)) \]
\[ lo_1 = \text{elem}(2, s-arg-list(g_0)) \]
\[ lg_1 = \text{elem}(3, s-arg-list(g_0)) \]
\[ onid_1 = s-ident\cdot s-chif(CS) \]

Ref.: is-gen 3-14(30)
\[ \text{convert-assign 8-8(16)} \]
\[ \text{convert-1 9-29(119)} \]
\[ \text{aggr-scalar 8-23(64)} \]
\[ \text{tit-cda 9-12(59)} \]

\[(175) \quad \text{le-val}(gen) = \]
\[ \text{pass-op-val}(op); \]
\[ \text{op:gen-op}(gen) \]

Ref.: \[ \text{op-val 9-9(36)} \]
\[ \text{gen-op 8-22(59)} \]

\[(176) \quad \text{assign-complex}(gen-1,gen-2,op) = \]
\[ \text{convert-assign}(gen-1,op-1,0,0); \]
\[ \text{op-1:pass-val-op}(da-op-real_0,v_1); \]
\[ \text{convert-assign}(gen-2,op-2,0,0); \]
\[ \text{op-2:pass-val-op}(da-op-real_0,v_2); \]

where:
\[ da-op-real_0 = \mu(op-da(op); <s-mode:REAL>) \]
\[ v_1 = \text{real\cdot op-val}(op) \]
\[ v_2 = \text{imag\cdot op-val}(op) \]

Ref.: \[ \text{convert-assign 8-8(16)} \]
\[ \text{val-op 9-9(36)} \]
\[ \text{op-da 9-9(37)} \]
\[ \text{op-val 9-9(36)} \]

\[(177) \quad \text{assign-reald}(gen,op) = \]
\[ \text{convert-assign}(gen,op-0,0,0); \]
\[ \text{op-0:pass-val-op}(gen-da(gen),v); \]
\[ v:\text{pass-cplz}(op-val(op),\text{imag\cdot op-val}(op-1)); \]
\[ \text{op-1:gen-op}(gen) \]

Ref.: \[ \text{convert-assign 8-8(16)} \]
\[ \text{val-op 9-9(36)} \]
\[ \text{gen-da 7-17(53)} \]
\[ \text{op-val 9-9(36)} \]
\[ 8-22(59) \]

70 12. BUILT-IN FUNCTIONS AND PSEUDO-VARIABLES
(178) \textbf{assign-izag}(\textit{gen}, \textit{op}) =

\begin{align*}
\text{convert-assign} & (\textit{gen}, \textit{op} - 0, 0, 0); \\
\text{op-1:pass-val-op} & (\textit{gen-da} (\textit{gen}), \textit{v}); \\
\text{v:pass-fplx} & (\text{real}(\textit{op-val} (\textit{op} - 1)), \textit{op-val}(\textit{op})); \\
\text{op-1:gen-op} & (\textit{gen})
\end{align*}

\textbf{Ref.}: convert-assign 8-8(16)
val-cp 8-5(38)
gen-da 7-17(53)
op-val 9-9(36)
gen-op 6-22(59)

(179) \textbf{assign-substr}(\textit{gen}, \textit{lpv}, \textit{i}, \textit{j}, \textit{op}, \textit{cond}) =

\begin{align*}
(\text{is-0}(\textit{j}) \land 1 \leq \textit{i} \leq \textit{k}_0) \lor (\text{is-0}(\textit{j}) \land 1 \leq \textit{i} + \textit{j} \leq \textit{k}_0 + 1) \rightarrow \\
\text{assign-substr-1}(\textit{gen}, \textit{lpv}, \textit{i}, \textit{op}-1); \\
\text{op-1:convert-1}(\textit{substr-eva}(\textit{eva}(\textit{gen}), \textit{i}, \textit{lg}), \textit{op}) \\
\text{j} = 0 \rightarrow \text{null} \\
\text{t} \rightarrow \\
\text{assign-substr-2}(\textit{gen}, \textit{lpv-2}, \textit{i}, \textit{j}, \textit{op}); \\
\text{lpv-2:lp-val}(\textit{gen}); \\
\text{substr-cond}(\textit{cond})
\end{align*}

\textbf{where}:
\begin{align*}
\textit{k}_0 & = \text{length}(\textit{lpv}) \\
\textit{lg}_0 & = \\
\text{is-0}(\textit{j}) & \rightarrow \textit{k}_0 - \textit{i} + 1 \\
\text{t} & \rightarrow \text{j}
\end{align*}

\textbf{for:} (\text{is-no-strc} \lor \text{is-0j}(\textit{cond})

\textbf{Ref.}: convert-1 9-29(115)
substr-cond 12-41(65)

(180) \textbf{substr-eva}(\textit{eva}, \textit{i}, \textit{lg}) =

\begin{align*}
\text{is-stringer-da}(\textit{eva}) & \rightarrow \mu(\textit{eva};<\text{length}\ast\text{da}:\textit{lg}) \\
\text{is-char-pieces-da}(\textit{eva}) & \rightarrow \\
\mu(\textit{eva};<\text{field}\ast\text{da}:\text{\textbf{LIST}}_{\text{m-i}} \text{ elem}(\text{m}, \text{e-field}\ast\text{da}(\textit{eva})))
\end{align*}

\textbf{for:} \text{is-eva}(\textit{eva}), \text{is-intg-val}(\textit{lg})

\textbf{Ref.}: is-intg-val 9-3(5)
ABSTRACT SYNTAX AND INTERPRETATION OF PL/I

30 April 1969

(181) \texttt{assign-substr-1}(\texttt{gen},\texttt{lpv},i,\texttt{op}) = \\
\texttt{assign}(\texttt{gen},\texttt{op})

where:
\begin{align*}
\texttt{op}_0 &= \mu_0(\langle\texttt{s-evs-evs-gen}\rangle,\langle\texttt{m-vr:rep} (\texttt{s-evs-gen},\texttt{str}_0)\rangle) \\
\texttt{str}_0 &= \mu(\texttt{lpv}; \\
& \langle\texttt{elem}(i + k - 1) : \texttt{elem}(k, \text{op-val}(\texttt{op})) | 1 \leq k \leq \text{length} \cdot \text{op-val}(\texttt{cp})\rangle)
\end{align*}

Ref.: \texttt{assign} 8-9(23) \\
\texttt{rep} 9-6(19) \\
\texttt{op-val} 9-9(35)

(182) \texttt{assign-substr-2}(\texttt{gen},\texttt{lpv},i,j,\texttt{cp}) = \\
i = 0 \rightarrow \texttt{null} \\
T \rightarrow \\
\texttt{assign-substr-1}(\texttt{gen},\texttt{lpv},i_{1},\texttt{op-1}); \\
\texttt{op-1:conv-1}(\texttt{substr-evs-s-evs-gen},i_{1},\texttt{lg}_{1}),\texttt{cp})

where:
\begin{align*}
i_{1} &= (i > k_0 \lor i > i + j \lor i + j < 1 \rightarrow 0, \\
i < 1 \rightarrow 1, \\
T \rightarrow i) \\
k_0 &= \text{length}(\texttt{lpv}) \\
\texttt{lg}_{1} &= \text{min}(k_0,i + j) - i_{1} + 1
\end{align*}

Ref.: \texttt{conv-1} 9-29(119)

(183) \texttt{assign-string}(i,\texttt{genl},\texttt{str}) = \\
is-\langle\rangle(\texttt{genl}) \rightarrow \texttt{null} \\
i < \text{length}(\texttt{str}) \rightarrow \\
\texttt{assign-string}(i + 1,\texttt{tail}(\texttt{genl}),\texttt{substr}(\texttt{str},i,1)); \\
\texttt{convert-assign}(\texttt{head}(\texttt{genl}),\texttt{val-op}(\texttt{eva}_{1},\texttt{substr}(\texttt{str},i,1)),\texttt{lg}_{0})

T \rightarrow \\
\texttt{convert-assign-list}(\texttt{genl},\texttt{op-1},0,0); \\
\texttt{op-1:conv-val-op}(\texttt{eva}_{2},\langle\rangle)

where:
\begin{align*}
l_0 &= \text{MIN}(l_1,\text{length}(\texttt{str}) - i + 1) \\
\texttt{eva}_{1} &= (\langle\texttt{is-Char}(\texttt{s-base}\cdot\texttt{gen-da}\cdot\texttt{head}(\texttt{genl})) \rightarrow \texttt{str-ev}(\texttt{CHAR},l_0), \\
T \rightarrow \texttt{str-ev}(\texttt{BIT},l_0)) \\
\texttt{eva}_{2} &= (\langle\texttt{is-Char}(\texttt{s-base}\cdot\texttt{gen-da}\cdot\texttt{head}(\texttt{genl})) \rightarrow \texttt{str-ev}(\texttt{CHAR},0), \\
T \rightarrow \texttt{str-ev}(\texttt{BIT},0))
\end{align*}

\texttt{fcv:}(\texttt{is-char-val-list} \lor \texttt{is-bit-val-list})(\texttt{str})

Ref.: \texttt{substr} 12-49(64) \\
\texttt{convert-assign} 8-8(16) \\
\texttt{convert-assign-list} 8-7(14) \\
\texttt{val-op} 9-9(35) \\
\texttt{gen-da} 7-17(53)

cont'd
ABSTRACT SYNTAX AND INTERPRETATION OF PL/I

str-eva 7-14 (45)
is-char-val 9-3 (6)
is-bit-val 9-4 (11)

(184) assign-correlation(gen, op) =

is-EVENT•gen-da (gen) ->

assign (gen, op-1):
op-1: pass-val-op (EVENT, v);
s-const(v): pass (head, op-val (op));
v: pass-op-val (op-g);

op-g: gen-op (gen)

T -> error

Ref.: gen-da 7-17 (53)
assign 8-9 (23)
val-op 9-9 (38)
op-val 9-9 (36)
gen-op 8-22 (59)

(185) assign-priority (gen, op) =

assign-priority-1(gen, op-1):
op-1: eval-pri-2(s-eva(gen), op)

Ref.: eval-pri-2 5-3 (8)

(186) assign-priority-2 (gen, op) =

is-TASK•gen-da (gen) & -is-active (gen, PA) v (3ths) (s-tv•s-testn (PA) = gen) ->

s-g•el-ass (s-vr (op), s-pp (gen), s)

Ref.: gen-da 7-17 (53)
is-active 5-5 (12)
el-ass 3-16 (46)

Note: Active task and event generations, other than the event generation being assigned to directly, are protected against being affected by the assignment.

(187) assign-status (gen, op) =

is-EVENT•gen-da (gen) & (is-active (gen, PA) v is-active-event (gen, PA)) ->

assign-status-1 (gen, op-1):
op-1: pass-val-op (EVENT, v):
s-status (v): pass-op-val (op):
v: pass-op-val (op-g):
op-g: gen-op (gen)

T -> error

Ref.: gen-da 7-17 (53)
is-active 5-5 (12)
is-active-event 5-5 (11)

cont'd

12. BUILT-IN FUNCTIONS AND PSEUDO-VARIABLES 73
12.7.2 EVALUATION OF PSEUDO-GENERATIONS

The instruction \texttt{ps-gen-op(id,list)} computes an operand from a pseudo-generation \texttt{ps-gen} where \texttt{id = s-id(ps-gen)} and \texttt{list = s-arg-list(ps-gen)}.

\begin{verbatim}
(189) \texttt{ps-gen-op(id,list) =}
     \begin{cases}
     (id = \texttt{mk-id(COMPLEX)} \land \texttt{id = mk-id(CPLX)} \land \texttt{length(list) = 2}) \lor \\
      (id = \texttt{mk-id(SPAL)} \land \texttt{id = mk-id(IMAG)} \land \texttt{id = mk-id(COMPLETION)} \land \\
      \texttt{length(list) = 1} \lor \texttt{id = mk-id(STRING)}) \lor \\
      (id = \texttt{mk-id(CNSOURCE)} \land \texttt{id = mk-id(CNCRAN)} \land \texttt{id = mk-id(GRIDENT)} \land \\
      \texttt{is-<>(list)}) \lor \\
      \texttt{eval-built-in-1(id,opl)}; \\
      \texttt{[elem(i)(opl)}; \texttt{gen-op(elem(i,list)) | 1 \leq i \leq length(list)}
     \end{cases}
\end{verbatim}

where:

\begin{verbatim}
\texttt{gen} = \texttt{is-string-type\texttt{-s-eva}(gen) \rightarrow gen}
T \rightarrow \emptyset
\end{verbatim}

for: \texttt{is-id(id),is-list(list)}

\textbf{Ref.:}
\begin{itemize}
  \item \texttt{eval-built-in-1 12-17(38)}
  \item \texttt{gen-op 8-22(59)}
  \item \texttt{eval-unrec 12-42(70)}
  \item \texttt{eval-substr-1 12-40(63)}
  \item \texttt{is-string-type 9-21(92)}
\end{itemize}
As mentioned in chapter 3.7.4, the optimization rules are described by modifying the set of strict computations of a program in two respects. First, to describe the rules for commoning of expressions (including the definition of the REDUCIBLE attribute), the concept of computation is extended, so that additional valid computations may be derived (section 13.1). Second, the set of strict computations is reduced by rejecting computations which are invalid because of the wrong use of the REORDER attribute or the RECURSIVE attribute in the interpreted program text (section 13.2).

13.1 RULES FOR COMMONING OF EXPRESSIONS: THE REDUCIBLE ATTRIBUTE

To describe the rules for commoning of expressions (including the definition of the REDUCIBLE attribute), the concept of computation as described in chapter 3.7 is extended in such a way that a step does not only depend on the current state, but also on previous states, i.e., on the current computation.

Thus, a computation is a sequence of states

$$\xi(0), \xi(1), \xi(2), \ldots$$

satisfying the following two conditions:

1) \(\xi(0)\) is an initial state of the interpreted program.

2) Each \(\xi(i)\) with \(i > 0\) results from the list of its predecessor states by a valid step including the possibility of commoning of expressions. The validity of such a step, called commoning step, is defined by the predicate \(\text{is-commoning-step}\), i.e., \(\xi(i)\) must satisfy the condition:

$$\text{is-commoning-step} (\langle \xi(0), \xi(1), \ldots, \xi(i-1) \rangle, \xi(i))$$

**Metavariables**

- \(\xi\), \(\xi-1\), \(\xi-2\) is-state
- \(\sigma\), \(\sigma-1\), \(\sigma-2\) is-selector
- \(i, i1, i2, j, k\) is-int-val
- \(x\) is-\(n\) is-op
- \(\text{expr}, \text{expr}-1\) is-\(\text{expr}\)
- \(\text{descr1}, \text{descr}-1\) is-descr-list
- \(\text{opt}\) is-\(v\) is-\(\text{G}\)
- \(\text{cpr}\) is-infix-\(\text{opr}\)
- \(\text{ref}, \text{ref}-1\) is-ref-1
- \(\xi1, \xi1-1\) is-subscr-\(\text{expr}\)-list

---

13.2 OPTIMIZATION
13.1.1 COMMUNING STEP

A communing step consists of three intermediate steps as described in chapter 3.7, with the only difference, that for the first intermediate step, called proper step, there exist two alternatives, i.e., a proper step may be either a normal computation step or possibly a proper communing step.

(1) \( \text{is-communing-step}(cp, \xi) = \)
\[
(3\xi-1, 3\xi-2) \text{ (is-proper-step}(cp, \xi-1) \land \text{is-env-step}(\xi-1, \xi-2) \land \\
\xi = \text{prep-interrupt}(\xi-1, \xi-2))
\]

Ref.: is-env-step 3-31(115)
prep-interrupt 3-32(121)

(2) \( \text{is-proper-step}(cp, \xi-1) = \)
\[
\text{is-comp-step}(\text{last}(cp), \xi-1) \lor \text{is-proper-communing-step}(cp, \xi-1)
\]

Ref.: is-comp-step 3-30(111)

Given the current computation \( cp \), a proper communing step exists under the following conditions:

1) There exists a terminal node \( \sigma \) in the control of the active task of the state \( \xi(k) = \text{last}(cp) \) selecting an instruction for evaluating an expression \( \text{expr} \).

2) There exists a state \( \xi(i) = \text{elem}(i, cp) \) with a terminal node \( \sigma-1 \) in the control of the same task selecting an instruction with the same name for evaluating an expression \( \text{expr} \).

3) There exists a state \( \xi(j) = \text{elem}(j, cp) \) with \( i < j \leq k \), in which the evaluation of \( \text{expr}-1 \) is just finished. I.e., \( \xi(j) \) must be the first state in the block activation of \( \xi(i) \) with empty \( \sigma-1 \) component of the control and must not result from a jump out of a contained block activation.
4) The two expressions expr, expr-1 are common expressions.

The result 5-1 of the proper commoning step is then obtained by replacing in \( \xi(k) \) the instruction in the \( \text{a} \) node by \( \text{naac}(v_1) \), where \( v_1 \) is the value of expr-1, and executing this instruction. The value \( v_1 \) is found in \( \xi(j) \) in the argument positions of the relevant control specified in \( \xi(j-1) \) by the return information of the \( \text{a}-1 \) node.

\[ \text{is-proper-commoning-step}(cp, \xi-1) = \]

\[ (\xi_1, \text{a}-1, i, j) (1 < j < \text{length}(cp)) \& a \in \text{term-node}(c_0) \& \text{a}-1 \in \text{term-node}(c_1) \& \]

\[ \text{is-finished} (\text{a}-1, i, j, cp) \& \text{is-common} (c_0, \text{a}-1, i, cp) \& \]

\[ a-1 = \text{compute} (\text{trans-last}(c_0, \text{a}-1, i, j, cp), c_0) \]

where:

\[ \text{tn-pri}_0 = \text{pri-sched} \text{last}(cp) \]
\[ c_0 = s \ast \text{tn-tri}_0 \ast \text{pa} \ast \text{last}(cp) \]
\[ c_1 = s \ast \text{tn-tri}_1 \ast \text{pa} \ast \text{elem}(i, cp) \]
\[ a_0 = a \ast s \ast \text{tn-tri}_0 \ast \text{pa} \]
\[ a-1_0 = a-1 \ast s \ast \text{tn-tri}_0 \ast \text{pa} \]

Ref.: pri-sched 3-30 (112)

(4) \( \text{trans-last}(a, a-1, j, cp) = \)

\[ (\text{last}(cp), \text{a} \ast \text{tn-tri}_0, \text{a} \ast \text{in} \ast \text{tag}, \text{a} \ast \text{sl} \ast \text{s} : \text{ret-val}(a-1, j, cp)) \]

(5) \( \text{ret-val}(a-1, j, cp) = \)

\[ (ux) ((a-2, i_1, i_2)(x = \text{arg-comp} a \& a-2 = s \ast \text{comp}(ri_1) \& i_1, i_2) \in \text{a-sar}(ri_4)) \]

where:

\[ \text{arg-comp}_4 = a-2 \ast \text{elem}(i_2, \text{a} \ast \text{all}(\text{pred}(i_1, a-1)) \ast \text{elem}(j, cp)) \]
\[ ri_4 = s \ast ri \ast a-1 \ast \text{elem}(j - 1, cp) \]

(6) \( \text{is-finished}(a-1, i, j, cp) = \)

\[ \text{ba}_2 = \text{ba}_3 \& \text{is-2a} \ast \text{a-1} \ast \text{elem}(j, cp) \& \text{a} \ast \text{in}(c_2) \& \text{goto-1} a \]

\[ \& (\text{ek})(i < k < j \& \text{ba}_3 = \text{ba}_1 \& \text{is-2a} \ast \text{a-1} \ast \text{elem}(k, cp)) \]

where:

\[ \text{ba}_1 = s \ast \text{tn-tri}_0 \ast \text{pa} \ast \text{elem}(i, cp) \]
\[ \text{ba}_2 = s \ast \text{tn-tri}_0 \ast \text{pa} \ast \text{elem}(j, cp) \]
\[ \text{ba}_3 = s \ast \text{tn-tri}_0 \ast \text{pa} \ast \text{elem}(k, cp) \]
\[ c_2 = s \ast \text{tn-tri}_0 \ast \text{pa} \ast \text{elem}(j, cp) \]

Ref.: goto-1 6-84 (121)
ABSTRACT SYNTAX AND INTERPRETATION OF PL/I

30 April 1969

(7) \( \text{is-common}(s, s-1, i, cp) = \)
\[ s-\text{infix-} \text{last}(cp) = s-\text{infix-}1 \text{elem}(i, cp) = \text{eval-entry-expr-1} \land \]
\[ \text{is-common-expr} (\text{expr}_0, \text{descr}_0, \text{last}(cp), \text{expr}_1, \text{descr}_1, \text{elem}(i, cp), \theta) \lor \]
\[ s-\text{prefix-} \text{last}(cp) = s-\text{prefix-}1 \text{elem}(i, cp) = \text{eval-exp-2} \land \]
\[ \text{is-common-expr} (\text{expr}_0, \theta, \text{last}(cp), \text{expr}_1, \theta, \text{elem}(i, cp), \theta) \]

where:
\[ \text{expr}_0 = \text{elem}(1, s-\text{all-} \text{last}(cp)) \]
\[ \text{descr}_0 = \text{elem}(2, s-\text{all-} \text{last}(cp)) \]
\[ \text{expr}_1 = \text{elem}(1, s-\text{all-}1 \text{elem}(i, cp)) \]
\[ \text{descr}_1 = \text{elem}(2, s-\text{all-}1 \text{elem}(i, cp)) \]

Ref.: \text{eval-entry-expr-1} 8-17(48)
\text{eval-exp-2} 8-19(50)

Note: The instruction \text{eval-entry-expr-1(expr, descr)} is the only one which is always invoked during expression evaluation in entry context, the instruction \text{eval-exp-2(expr)} is the only one which is always invoked during expression evaluation in non-entry context. Therefore, these instructions constitute the starting-points for proper commoning steps.

13.1.2 COMMON EXPRESSIONS

Although only scalar expressions may be composed in a proper commoning step, the notion of common expressions must be defined for the general case of aggregate expressions, since these may appear as arguments of common function references.

Two expressions are common, if they have the same structure, and if corresponding components are common references, the same constants or the same isubs.

(8) \( \text{is-common-expr}(\text{expr}, \text{descr}, \xi, \text{expr}-1, \text{descr}-1, \xi-1, \theta) = \)
\[ \text{is-infix-expr-1} (\text{expr}) \rightarrow \]
\[ \text{is-infix-expr-1} (\text{expr}-1) \land s-\text{opr} (\text{expr}) = s-\text{opr} (\text{expr}-1) \land \]
\[ \text{is-common-infix-expr} (\text{expr}, \text{descr}, \xi, \text{expr}-1, \text{descr}-1, \xi-1, \theta) \]
\[ \text{is-prefix-expr-1} (\text{expr}) \rightarrow \]
\[ \text{is-prefix-expr-1} (\text{expr}-1) \land s-\text{opr} (\text{expr}) = s-\text{opr} (\text{expr}-1) \land \]
\[ \text{is-common-expr} (s-\text{cp} (\text{expr}), \text{descr}, \xi, s-\text{op} (\text{expr}-1), \text{descr}-1, \xi-1, \theta) \]
\[ \text{is-paren-expr-1} (\text{expr}) \rightarrow \]
\[ \text{is-paren-expr-1} (\text{expr}-1) \land \]
\[ \text{is-common-expr} (s-\text{cp} (\text{expr}), \text{descr}, \xi, s-\text{op} (\text{expr}-1), \text{descr}-1, \xi-1, \theta) \]
\[ \text{is-ref-1} (\text{expr}) \rightarrow \]
\[ \text{is-ref-1} (\text{expr}-1) \land \text{is-common-ref} (\text{expr}, \text{descr}, \xi, \text{expr}-1, \text{descr}-1, \xi-1, \theta) \]
\[ \text{is-const} (\text{expr}) \lor \text{is-isub} (\text{expr}) \rightarrow \text{expr} = \text{expr}-1 \]

Ref.: \text{is-infix-expr-1} 8-16(41)
\text{is-prefix-expr-1} 8-16(42)
\text{is-paren-expr-1} 8-16(43)
\text{is-ref-1} 8-17(44)
Note: The argument opt is ⟨ in entry context, * in non-entry context (in relevant cases, i.e., in cases concerning entry references). The arguments descril, descril-1 are relevant only for generic and built-in references with empty argument part in entry context.

(9) \( \text{is-common-infix-expr}(\text{expr}, \text{descr}l, \epsilon, \text{expr}-1, \text{descr}l-1, \epsilon-1, \text{opt}) = \)
\[
\text{is-common-expr}(\text{expr}, \text{descr}l, \epsilon, \text{expr}-1, \text{descr}l-1, \epsilon-1, \text{opt}) \quad \& \quad
\text{is-commutative-infix-opr}(\text{expr}) \quad \& \quad
\text{is-common-area}(\text{agr}1, \text{agr}2, \epsilon, \text{agr}1-1, \text{agr}1-2, \epsilon-1)
\]

Where:
\[
\text{agr}1 = \text{agr-expr}(\text{expr}, \text{agr}1-1, \epsilon)
\]
\[
\text{agr}2 = \text{agr-expr}(\text{expr}, \text{agr}2-1, \epsilon)
\]
\[
\text{agr}1-1 = \text{agr-expr}(\text{expr}-1, \text{agr}1-2, \epsilon-1)
\]
\[
\text{agr}1-2 = \text{agr-expr}(\text{expr}-1, \text{agr}1-1, \epsilon-1)
\]

Ref.: \( \text{agr-expr} 6-23(62) \)

Note: The term is-common-area(...) assures correct handling in the case of pointer-offset comparison.

(10) \( \text{is-commutative-infix-opr}(\text{opr}) = \)
\[
\text{opr} \in \{\text{OR, AND, IC, NE, AND, MUL}\}
\]
(11) \( \text{is-common-area}(\text{aggr-1}, \text{aggr-2}, \ldots, \text{aggr-11}, \text{aggr-12}, \xi-1) \rightarrow \\
\text{is-array}(\text{aggr-1}) \& \text{is-array}(\text{aggr-2}) \rightarrow \\
\text{is-common-area}(\text{s-elem}(\text{aggr-1}), \text{s-elem}(\text{aggr-2}), \ldots, \text{s-elem}(\text{aggr-11}), \text{s-elem}(\text{aggr-12}), \xi-1) \\
\text{is-array}(\text{aggr-1}) \rightarrow \\
\text{is-common-area}(\text{s-elem}(\text{aggr-1}), \text{aggr-2}, \ldots, \text{s-elem}(\text{aggr-11}), \text{aggr-12}, \xi-1) \\
\text{is-array}(\text{aggr-2}) \rightarrow \\
\text{is-common-area}(\text{aggr-1}, \text{s-elem}(\text{aggr-2}), \ldots, \text{aggr-11}, \text{s-elem}(\text{aggr-12}), \xi-1) \\
\text{is-struct}(\text{aggr-1}) \& \text{is-struct}(\text{aggr-2}) \rightarrow \\
\text{length}(\text{aggr-1}) \\
\text{Et} \quad \text{is-common-area}(\text{s-aggr\_elem}(i, \text{aggr-1}), \text{s-aggr\_elem}(i, \text{aggr-2}), \\
\ldots, \text{s-aggr\_elem}(i, \text{aggr-11}), \text{s-aggr\_elem}(i, \text{aggr-12}), \xi-1) \\
\text{is-struct}(\text{aggr-1}) \rightarrow \\
\text{length}(\text{aggr-1}) \\
\text{Et} \quad \text{is-common-area}(\text{s-aggr\_elem}(i, \text{aggr-1}), \text{aggr-2}, \ldots, \text{s-aggr\_elem}(i, \text{aggr-11}), \text{aggr-12}, \xi-1) \\
\text{is-struct}(\text{aggr-2}) \rightarrow \\
\text{length}(\text{aggr-2}) \\
\text{Et} \quad \text{is-common-area}(\text{aggr-1}, \text{s-aggr\_elem}(i, \text{aggr-2}), \ldots, \text{s-aggr\_elem}(i, \text{aggr-11}), \\
\text{s-aggr\_elem}(i, \text{aggr-12}), \xi-1) \\
\text{is-PTE\_s\_da}(\text{aggr-1}) \& \text{is-offset\_s\_da}(\text{aggr-2}) \rightarrow \\
\text{is-common-ref}(\text{s-area\_s\_da}(\text{aggr-2}), \xi, \text{s-area\_s\_da}(\text{aggr-12}), \xi-1, \xi) \\
\text{is-offset\_s\_da}(\text{aggr-1}) \& \text{is-PTE\_s\_da}(\text{aggr-2}) \rightarrow \\
\text{is-common-ref}(\text{s-area\_s\_da}(\text{aggr-1}), \xi, \text{s-area\_s\_da}(\text{aggr-11}), \xi-1, \xi) \\
\xi \rightarrow 0 \\

13.1.3 COMMON REFERENCES

First, if one of the references is a generic reference, it is replaced by the reference built from the selected entry reference by concatenating its argument part with the argument part of the originally reference.

Then, two references are common, if they have the same evaluated list of name qualifiers, if their subscript lists are common (i.e., if corresponding subscript expressions are common), and if, according to the kind of the references, one of the following conditions holds:

1) Both are references to variables, these variables are common, and, for entry components, the corresponding entry reference in question is reducible and the argument parts are common.

2) Both are entry references to the same function, the reference in question is reducible, and the argument parts are common.

3) Both are references to the same reducible built-in function, and the argument parts are common.

6 13. OPTIMIZATION
4) Both are references to label, format or file constants with the same denotation.

Two variables are common in the following cases:

1) Both are proper variables with the same generation and the same value representation.

2) Both are defined variables with the same denotation, their bases are common references, and their positions are either 0 or common expressions.

3) Both are based variables with the same generation and the same value representation, their pointer qualifiers are common references, and their refer options have the same evaluated list of name qualifiers.

Note, that based variables can not be commoned if they occur in argument expressions of function references, since in this case their generations are unknown in the states in question.

Whether an entry reference is reducible or not depends on the context of the entry reference (entry or non-entry), and, in the case of entry context, on the length of the argument part. An entry reference in entry context is reducible, if either the argument part is empty, or the corresponding entry attribute is declared as reducible and the number of its reducible declared return types is at least equal to the length of the tail of the argument part. An entry reference in non-entry context is reducible, if the corresponding entry attribute is declared as reducible and all its return types of type entry are declared as reducible.

Note, that the explained notion of reducible entry references constitutes the definition of the REDUCIBLE attribute.

Two argument parts are common, if corresponding arguments are common expressions, their descriptor aggregate attributes (given explicitly or be default) are essentially equal, and, in special cases, some further conditions hold (cf. the note to (22)).

Abbreviations

$id_0 = \text{head}^{*}\text{-id-list}(\text{ref})$  

main identifier of a reference

$n_o = s^n(\text{ref})$  

unique name of referenced item

$\text{attr}_o = p^*s-at(\ell)$  

attributes of referenced item

$\text{aggr}_o = s^*aggr(\text{attr}_o)$  

aggregate attributes of referenced variable

$ecl_1 = \text{eval-ql}(\text{tail}\text{-id-list}(\text{ref}),\text{aggr}_o)$  

evaluated list of name qualifiers

$\text{aggr}_1 = s^*aggr(\text{aggr}_o, s^*sl(\text{ref}), ecl_1)$  

aggregate attributes of referenced part of variable

$id-1_0 = \text{head}^{*}\text{-id-list}(\text{ref}-1)$  

main identifier of a reference

$n-1_0 = s^r(\text{ref}-1)$  

unique name of referenced item

$\text{attr}-1_0 = n-1_0^*s-at(\ell-1)$  

attributes of referenced item

$\text{aggr}-1_0 = s^*aggr(\text{attr}-1_0)$  

aggregate attributes of referenced variable

$ecl-1_1 = \text{eval-ql}(\text{tail}\text{-id-list}(\text{ref}-1),\text{aggr}-1_0)$  

13. OPTIMIZATION
aggr-1 = sub-aggr(aggr-1, s-sl(ref-1), egl-1)

(12) is-common-ref(ref, descr1, r, ref-1, descr-1, r-1, opt) =
    is-generic(attr0) & is<>s-sl(ref) & is-gptr(ref) ---
    is-common-ref(ref, s, ref-1, descr-1, r-1, opt)
    is-generic(attr-1) --- is-common-ref(ref, descr1, r, ref-1, descr-1, r-1, opt)

T =
    egl-1 & is-common-sl(s-sl(ref), r, s-sl(ref-1), r-1) &
    is-common-ref-1(ref, descr-1, r-1, descr-1, r-1, opt)

where:
ref0 = m(ref-g0: s-ap(s-ap(ref-g0) s-ap(ref))
ref-g0 = generic-sel(attr0, descr1)
descr1 = (is-<>(opt) & is<>s-ap(ref) --- descr1, T --- attr-expr-list(head s-ap(ref), s-at(r)))
ref-1 = m(ref-g-1: s-ap(s-ap(ref-g-1) s-ap(ref))
ref-g-1 = generic-sel(attr-1, descr1-1)
descr-1 = (is<>(opt) & is<>s-ap(ref-1) --- descr-1, T --- attr-expr-list(head s-ap(ref-1), s-at(r)))

Ref.:
    generic-sel 6-31(77)
    attr-expr-list 6-35(96)

(13) is-common-sl(sl, r, sl-1, r-1) =
    is<>(sl) --- is<>(sl-1)
    is-*head(sl) --- is-*head(sl-1) & is-common-sl(tail(sl), r, tail(sl-1), r-1)

T =
    is-common-exr(head(sl), r, head(sl-1), r, r-1, *) &
    is-common-sl(tail(sl), r, tail(sl-1), r-1)
ABSTRACT SYNTAX AND INTERPRETATION OF PL/I

(14) is-common-ref-1(ref, descr, ε, ref-1, descr-1, ε-1, opt) =
    is-var(attr0) --
    is-var(attr-1) & is-common-var(ref, ε, ref-1, ε-1) &
    is-common-entry(aggr1, s-ap(ref), ε, aggr1, s-ap(ref-1), ε-1, opt)
    is-entry(attr0) & is-<>•s-sl(ref) & is-0•s-ptr(ref) --
    is-entry(attr-1) & n0•s-dn(ε) = n-1•s-dn(ε-1) &
    is-reducible(attr0, length•s-ap(ref), opt) &
    is-common-ap(s-ap(ref), attr0, ε, s-ap(ref-1), attr-1, ε-1)
    is-BUILTIN(attr0) & is-<>•s-sl(ref) & is-0•s-ptr(ref) --
    is-BUILTIN(attr-1) & id0 = id-1 & is-reducible-builtin(id0) &
    (is-<•c(opt) & is-<>•s-ap(ref) & descr = descr-1 v
    is-common-builtin-ap(s-ap(ref), ε, s-ap(ref-1), ε-1))
    (is-label-const v is-format-const v is-file-const)(attr0) & is-<>•s-sl(ref) &
    is-<>•s-ap(ref) & is-0•s-ptr(ref) --
    n0•s-dn(ε) = n-1•s-dn(ε-1)
T → ε

Ref.: is-var 8-9(22)

(15) is-common-var(ref, ε, ref-1, ε-1) =
    is-prop-var(attr0) -- is-prop-var(attr-1) & gen0 = gen-1 & vrf0 = vrf-1 &
    is-defined(attr0) --
    is-defined(attr-1) & n0•s-dn(ε) = n-1•s-dn(ε-1) &
    is-common-ref(s-base(attr0), ε, s-base(attr-1), ε, ε-1, ε) &
    (s-pos(attr0) = s-pos(attr-1) = s v
    is-common-expr(s-pos(attr0), ε, s-pos(attr-1), ε, ε-1, ε))
    is-based(attr0) & is-0•s-gen(ref) --
    is-based(attr-1) & s-gen(ref) = s-gen(ref-1) & vrp-f0 = vrp-f-1 &
    is-common-ref(ptr0, ε, ptr-1, ε, ε-1, ε) &
    is-common-refer-options(aggr0, aggr-1)
T → P

where:
    tn-prig0 = tri-sched(ε)
    gen0 = head•(n0•s-dn(ε))•s-ag•tn-prig0•s-pa(ε)
    vrp0 = vr-part(s-pg(gen0), s-s(ε))
    vrp-f0 = vr-part(s-pg(s-gen(ref)), s-s(ε))
    ptr0 = (is-0•s-ptr(ref) -- s-ptr(attr0),
               T → s-ptr(ref))
    gen-10 = head•(n-1•s-dn(ε-1))•s-ag•tn-prig0•s-pa(ε-1)
    vrp-10 = vr-part(s-pg(gen-10), s-s(ε-1))
    vrp-f-10 = vr-part(s-pg(s-gen(ref-1)), s-s(ε-1))
    ptr-10 = (is-0•s-ptr(ref-1) -- s-ptr(attr-1),
               T → s-ptr(ref-1))

Ref.: pri-sched 3-30(112)
Note: For the case of based variables see the remark in the text of this section.

(16) \( \text{vr-part}(pp, stg) = \)

\[ \text{is-} \text{ptr-val}(pp) \rightarrow pp(stg) \]

\[ \text{length}(pp) \]

\[ T \leftarrow \text{LIST} \]

\[ i=1 \]

vr-part(eles(i,pp),stg)

Ref.: \( \text{is-} \text{ptr-val} \ 3-15(36) \)

(17) \( \text{is-common-refer-options}(aggr, aggr-1) = \)

\( (\forall e)(\text{is-refer}\_e(aggr) \land \text{is-refer}\_e(aggr-1) \land \text{egl}\_r-e = \text{egl}\_r-1_e \lor \text{is-refer}\_e(aggr) \land \text{is-refer}\_e(aggr-1)) \)

where:

\( \text{egl}\_r-e = \text{eval-gl}(s-id-list\_e(aggr), aggr) \)

\( \text{egl}\_r-1_e = \text{eval-gl}(s-id-list\_e(aggr-1), aggr-1) \)

Ref.: \( \text{eval-gl} \ 8-30(94) \)

(18) \( \text{is-common-entry}(aggr, ap, e, aggr-1, ap-1, e-1, opt) = \)

is-array(aggr) \( \rightarrow \)

is-common-entry(s-elem(aggr), ap, e, s-elem(aggr-1), ap-1, e-1, opt)

is-struct(aggr) \( \rightarrow \)

length(aggr)

\[ \text{Et} \]

is-common-entry(s-aggr-elem(i, aggr), ap, e, s-aggr-elem(i, aggr-1), ap-1, e-1, opt)

is-entry-s-da(aggr) \( \rightarrow \)

is-reducible(s-s-da(aggr), length(ap), opt) \( \land \)

is-common-ap(ap, s-da(aggr), e, ap-1, s-da(aggr-1), e-1)

\[ T \rightarrow \text{is-}\rightarrow(s) \]

(19) \( \text{is-reducible}(da, i, opt) = \)

is-entry(da) \( \land \) \( (\text{is-}*(opt) \lor i > 0) \) \( \rightarrow \)

is-\*da-reducible(da) \( \land \) \( \text{is-reducible}(s-da\_s-ret-type(da), i - 1, opt) \)

\[ T \rightarrow T \]

for: \( \text{is-entry-da} \lor \text{is-entry-const}(da) \)

10 13. OPTIMIZATION
(20) \( \text{is-common-ap}(\text{ap}, \text{da}, \xi, \text{ap-1}, \text{da-1}, \xi-1) = \)
\[ \text{is-<>(ap)} \rightarrow \text{is-<>(ap-1)} \]
\[ T \rightarrow \]
\[ \text{is-common-alist}\left(\text{head(ap)}, \xi, \text{head(da)}, \xi, \text{head(da-1)}, \xi, \text{head(da-1)}, \xi-1\right) \& \]
\[ \text{is-common-ap}(\text{tail(ap)}, \xi, \text{tail(da)}, \xi, \text{tail(da-1)}, \xi, \text{tail(da-1)}, \xi-1) \]
for: \( \text{is-entry-da} \vee \text{is-entry-const}(\text{da}) \)

(21) \( \text{is-common-alist}\left(\text{al}, \text{descrl}, \xi, \text{al-1}, \text{descrl-1}, \xi-1\right) = \)
\[ \text{is-<>(al)} \rightarrow \text{is-<>(al-1)} \]
\[ T \rightarrow \]
\[ \text{is-common-arg}(\text{head(al)}, \xi, \text{head(descrl)}, \xi, \text{head(descrl-1)}, \xi, \text{head(descrl-1)}, \xi-1) \& \]
\[ \text{is-common-alist}(\text{tail(al)}, \xi, \text{tail(descrl)}, \xi, \text{tail(al-1)}, \xi, \text{tail(descrl-1)}, \xi-1) \]

(22) \( \text{is-common-arg}(\text{expr}, \d-aggr, \xi, \text{expr-1}, \d-aggr-1, \xi-1) = \)
\[ \text{is-ENTRY•s-da} \left(\text{aggr}_{x} \right) \vee \text{is-ENTRY•s-da}(\text{aggr}_{x-1}) \& \]
\[ \text{is-ENTRY•s-da}(\text{aggr}_{x}) \& \]
\[ \text{pure-aggr}(\d-aggr_{x}) = \text{pure-aggr}(\d-aggr_{x-1}) \& \]
\[ \text{is-common-generic-arg}(\text{expr}, \d-aggr_{x}, \xi, \text{expr-1}, \d-aggr_{x-1}, \xi-1) \]
\[ T \rightarrow \]
\[ \text{is-common-expr}(\text{expr}, 0, \xi, \text{expr-1}, 0, \xi-1) \& \]
\[ \text{pure-aggr}(\d-aggr_{x}) = \text{pure-aggr}(\d-aggr_{x-1}) \& \]
\[ \text{is-common-aggr}(\text{aggr}_{x}, \d-aggr_{x}, \xi, \text{aggr}_{x-1}, \d-aggr_{x-1}, \xi-1) \]

where:
\[ \text{aggr}_{x} = \text{aggr-expr-1}(\text{expr}, \xi, \text{at}(\xi)) \]
\[ \d-aggr_{x} = (\text{is}-\exists(\d-aggr) \rightarrow \text{aggr}_{x}, \text{T} \rightarrow \d-agqr) \]
\[ \text{aggr}_{x-1} = \text{aggr-expr-1}(\text{expr-1}, \xi, \text{at}(\xi-1)) \]
\[ \d-aggr_{x-1} = (\text{is}-\exists(\d-aggr-1) \rightarrow \text{aggr}_{x-1}, \text{T} \rightarrow \d-agqr-1) \]

Def.: \( \text{pure-aggr} \& 20(52) \)
\( \text{aggr-expr-1} \& 35(98) \)

Note: The first alternative assures correct handling of generic references in the argument expressions. The term \( \text{is-common-aggr(...)} \) in the second alternative assures correct handling of entry - non-entry and pointer - offset conversion caused by differences between the aggregate attributes of the argument expressions and their descriptor aggregate attributes.
is-common-generic-arg(expr, d-aggr, $\xi$, expr-1, d-aggr-1, $\xi$-1) =$
\begin{align*}
is-array(d-aggr) \quad \rightarrow \\
is-common-generic-arg(expr, s-elem(d-aggr), $\xi$, expr-1, s-elem(d-aggr-1), $\xi$-1) \\
is-struct(d-aggr) \quad \rightarrow \\
\text{length}(d-aggr) \\
\text{ Et } is-common-generic-arg(expr, s-elem(s-aggr\cdot elem(i, d-aggr)), \\
i=1 \\
\xi, expr-1, s-aggr\cdot elem(i, d-aggr-1), $\xi$-1) \\
T \rightarrow \\
is-common-expr(expr, s-descr-lists-da(d-aggr), $\xi$, expr-1, s-descr-lists-da(d-aggr-1), $\xi$-1, $\Omega$)
\end{align*}

(24) is-common-aggr(aggr, d-aggr, $\xi$, aggr-1, d-aggr-1, $\xi$-1) =$
\begin{align*}
is-array(aggr) \& is-array(d-aggr) \quad \rightarrow \\
is-common-aggr(s-elem(aggr), s-elem(d-aggr), $\xi$, s-elem(aggr-1), s-elem(d-aggr-1), $\xi$-1) \\
is-array(d-aggr) \quad \rightarrow \\
is-common-aggr(aggr, s-elem(d-aggr), $\xi$, aggr-1, s-elem(d-aggr-1), $\xi$-1) \\
is-struct(aggr) \& is-struct(d-aggr) \quad \rightarrow \\
\text{length}(aggr) \\
\text{ Et } is-common-aggr(s-aggr\cdot elem(i, aggr), s-aggr\cdot elem(i, d-aggr), \\
i=1 \\
\xi, s-aggr\cdot elem(i, aggr-1), s-aggr\cdot elem(i, d-aggr-1), $\xi$-1) \\
is-struct(d-aggr) \quad \rightarrow \\
\text{length}(d-aggr) \\
\text{ Et } is-common-aggr(aggr, s-aggr\cdot elem(i, d-aggr), $\xi$, aggr-1, \\
i=1 \\
s-aggr\cdot elem(i, d-aggr-1), $\xi$-1) \\
T \rightarrow is-common-da(s-da(aggr), s-da(d-aggr), $\xi$, s-da(aggr-1), s-da(d-aggr-1), $\xi$-1)
\end{align*}

(25) is-common-da(da, d-da, $\xi$, da-1, d-da-1, $\xi$-1) =$
\begin{align*}
is-entry(da) \& is-entry(d-da) \quad \rightarrow \text{is-reducible}(da, 0, *) \\
is-PTR(da) \& is-offset(d-da) \quad \rightarrow \\
is-common-ref(s-area(d-da), $\Omega$, $\xi$, s-area(d-da-1), $\Omega$, $\xi$-1, *) \\
is-offset(da) \& is-PTR(d-da) \quad \rightarrow \\
is-common-ref(s-area(da), $\Omega$, $\xi$, s-area(da-1), $\Omega$, $\xi$-1, *) \\
T \rightarrow T
\end{align*}
(26) \text{is-reducible-built-in}(id) =
\begin{align*}
\text{is-built-in}(id) \land \\
\{ & \text{mk-id} \text{ (STATEIELD)}, \text{mk-id} \text{ (ONCHAR)}, \text{mk-id} \text{ (ONCODE)}, \text{mk-id} \text{ (ONCOUNT)}, \text{mk-id} \text{ (ONLINE)}, \\
& \text{mk-id} \text{ (ONCOUNT)}, \text{mk-id} \text{ (ONLOC)}, \text{mk-id} \text{ (ONSOURCE)}, \\
& \text{mk-id} \text{ (PRIORITY)}, \text{mk-id} \text{ (STATUS)}, \text{mk-id} \text{ (COUNT)}, \text{mk-id} \text{ (ENABLED)}, \text{mk-id} \text{ (LINENO)}, \\
& \text{mk-id} \text{ (TIME)} \}
\end{align*}

Ref.: is-built-in 12-13(31)

(27) \text{is-common-built-in-ap}(ap, \xi, ap-1, \xi-1) =
\begin{align*}
\text{is-}<>\text{(ap)} \rightarrow \text{is-}<>\text{(ap-1)} \\
T \rightarrow \text{is-}<>\text{tail}(ap) \land \text{is-common-sl}(head(ap), \xi, head(ap-1), \xi-1)
\end{align*}

13.2 \textbf{DEFINITION OF THE REORDER ATTRIBUTE AND THE RECURSIVE ATTRIBUTE}

The \textit{REORDER} attribute and the \textit{RECURSIVE} attribute are defined in such a way, that two predicates \textit{is-reorder-erroneous} and \textit{is-recursive-erroneous} are defined which prove a member of the set of strict computations of a program with regard to the wrong use of these attributes in the interpreted program text. I.e., a computation \( cp \) is rejected from the set of strict computations, if one of the following conditions holds:

1) \( \text{is-reorder-erroneous}(cp) \)

2) \( \text{is-recursive-erroneous}(cp) \)

\textbf{Metavariabes}

- \( cp \): \text{is-state-list} \quad \text{strict computation}
- \( tn \): \text{is-n} \quad \text{task name}
- \( \xi, \xi-1, \xi-2, \xi-3 \): \text{is-state} \quad \text{states}
- \( a, a-1, a-2 \): \text{is-selector} \quad \text{selectors}
- \( i, i_1, i_2 \): \text{is-intg-val} \quad \text{integer values}
- \( j, j_1, j_2 \): \text{is-intg-val} \quad \text{integer values}
- \( k, k_1, k_2 \): \text{is-intg-val} \quad \text{integer values}
- \( p \): \text{is-ptr-val} \quad \text{pointer value}
- \( b \): \text{is-s} \quad \text{aggregate name}
- \( n \): \text{is-n} \quad \text{unique name}
- \( vr \): \text{is-vr} \quad \text{value representation}
13.2.1 THE REORDER ATTRIBUTE

A computation cp is erroneous because of the wrong use of the REORDER attribute in the interpreted program text, if the following conditions hold:

1) There exists a section \[ \langle \xi(i_1), \ldots, \xi(i_2) \rangle \] as part of the computation cp, called reorder-section, which constitutes (disregarding steps belonging to another task) the computation of a block or procedure body declared with the REORDER attribute.

2) There exists a section \[ \langle \xi(j_1), \ldots, \xi(j_2) \rangle \] as part of this reorder section, called on-section, which constitutes the computation of an on-unit.

3) There exists a pointer p which is contained in the allocation state of the storage between \( \xi(k_1) \) and \( \xi(k_2) \) (exclusively), and which does not belong to an automatic variable declared in a block or procedure whose body is declared with the ORDER attribute and computed inside the reorder-section.

4) There is a reference to this pointer p in a state \( \xi(k) \) between \( \xi(k_1) \) and \( \xi(k_2) \) which is not guaranteed in such circumstances.

A reference to the pointer p in the state \( \xi(k) \) is not guaranteed in the above circumstances, if one of the following alternatives holds:

1) \( \xi(k) \) lies in the on-section.
   p is not referenced by the use of on-built-in functions.
   p is allocated or freed or its contents is modified in the reorder-section (this means now outside the on-section), or p is allocated outside the on-section (possibly also outside the reorder-section) and belongs to a controlled variable which is allocated or freed in the reorder-section.

2) \( \xi(k) \) lies in or behind the reorder-section.
   p is allocated or freed or its contents is modified in the on-section without the use of on-pseudovariables, or p is allocated before the end of the reorder-section and belongs to a controlled variable which is allocated or freed or modified in the on-section without the use of on-pseudovariables.

3) \( \xi(k) \) lies in or behind the reorder-section.
   There is an abnormal return from the on-section.
   p is allocated or freed or its contents is modified in the reorder-section, or p is allocated before the end of the reorder-section and belongs to a controlled variable which is allocated or freed in the reorder-section.

\[(28) \text{is-reorder-erroneous}(cp) = \]
\[\langle i_1, i_2, j_1, j_2, p, k, k_1, k_2, t_n \rangle (i_1 < j_1 < j_2 < i_2) \]
\[\text{is-reorder-section}(i_1, i_2, t_n, cp) \& \text{is-on-section}(j_1, j_2, t_n, cp) \& k_1 < k < k_2 \& \]
\[\text{is-alloc}(p, k_1, k_2, j_1, t_n, cp) \& \]
\[\text{is-not-guaranteed-reference}(i_1, i_2, j_1, j_2, p, k, k_1, k_2, t_n, cp)\]

\[(29) \text{is-reorder-section}(i_1, i_2, t_n, cp) = \]
\[\text{is-section}(i_1, i_2, t_n, cp) \& \]
\[\text{(2e)} (s \& \text{term-node}(c_1)) \& \]
\[\text{(s-in-o}(c_1) = \text{int-block} \& \text{is-*o-reorder}(arg_1) \lor \text{s-in-o}(c_1) = \text{cell-proc} \& \text{is-*o-reorder-s-body}(arg_1)) \& \text{is-}^o(s(c_1))\]

where:
\[c_1 = s\cdot t_n\cdot s\cdot p\cdot a\cdot e\cdot l\cdot e\cdot m(i_1, cp)\]
\[arg_1 = \text{elem}(1, s\cdot a\cdot e\cdot m(c_1))\]
\[c_2 = s\cdot t_n\cdot s\cdot p\cdot a\cdot e\cdot l\cdot e\cdot m(i_2, cp)\]
Ref.: int-block 6-2(1)
call-proc 6-16(43)

(30) is-section(j1,j2,tn,cp) =

is-section(j1,j2,tn,cp) &
(3e) (e \in term-node(c_a) & s-in\sigma(e) = int-cond & is-\Omega(e))

where:
\[ c_a = s-ctns-paelem(j1,cp) \]
\[ c_a = s-ctns-paelem(j2,cp) \]

Ref.: int-cond 10-20(60)

(31) is-section(i1,i2,tn,cp) =

i1 < i2 & ba_1 = ba_2 & (\forall i)(i1 < i < i2 = ba_0 \neq ba_1)

where:
\[ ba_1 = s-ba*tn*pa*elem(i1,cp) \]
\[ ba_2 = s-ba*tn*pa*elem(i2,cp) \]
\[ ba_0 = s-ba*tn*pa*elem(i,cp) \]

(32) is-alloc(p,k1,k2,i1,tn,cp) =

(\forall k)(k1 < k < k2 \Rightarrow p \in allst_0) & p \notin allst_1 & p \notin allst_2 &
(\exists b, l, o)(p = s-pp*head*b(a_0) & il < i < k_1 & o \in term-node(c_0) &
(s-in\sigma(c_0) = int-block & is-\Omega*s-reorder(arg_0) v s-in\sigma(c_0) = call-proc &
is-\Omega*s-reorder*s-body(arg_0)) & is-\Omega*b(a_0))

where:
\[ allst_0 = alloc-state*paelem(k,cp) \]
\[ allst_1 = alloc-state*paelem(k1,cp) \]
\[ allst_2 = alloc-state*paelem(k2,cp) \]
\[ a_0 = s-ag*tn*pa*elem(k1+1,cp) \]
\[ c_0 = s-c*tn*pa*elem(i,cp) \]
\[ arg_0 = elem(1,s-al\sigma(c_0)) \]
\[ a_0 = s-ag*tn*pa*elem(i,cp) \]

Ref.: int-block 6-2(1)
call-proc 6-16(43)
alloc-state 3-16(53)

(33) is-not-guaranteed-reference(i1,i2,j1,j2,p,k,k1,k2,tn,cp) =

j1 < k < j2 & is-referenced-1(p,k,tn,cp) &
is-mcd-alloc-freed-1(k1,i2,j1,j2,p,k,k1,k2,tn,cp) v (i1 < k < j1 v j2 < k) &
is-referenced(p,k,tn,cp) &
is-mcd-alloc-freed-2(i2,j1,j2,p,k,k1,k2,tn,cp) v
is-abnormal-return(j2,tn,cp) &
is-mcd-alloc-freed-3(i1,i2,j1,j2,p,k,k1,k2,tn,cp))

(34) is-abnormal-return(j2,tn,cp) =

s-in\sigma*c*tn*pa*elem(j2,cp) = goto-1

Ref.: goto-1 6-44(121)

13. OPTIMIZATION 15
is-referenced\(p,k,tn,cp\) =
\[
\begin{align*}
tn & = \text{pri-sched}(\xi_0) \land (3e) (e \in \text{term-node}(c_0) \land \text{is-}e\text{-dependent}(p,\xi_0,tn,e_0)) \land \text{is-}e\text{-dependent}(p,\xi_0,tn,e_1)\\
\end{align*}
\]
where:
\[
\begin{align*}
\xi_0 & = \text{elem}(k,cp)\\
\xi_1 & = \text{elem}(k+1,cp)\\
c_0 & = s\cdot c\cdot tm\cdot s\cdot ps(\xi_0)\\
e_0 & = s\cdot s\cdot c\cdot tm\cdot s\cdot ps\\
\xi_1 & = \text{exec-sel}(\xi_0,\xi_1,tn)\\
\end{align*}
\]
Ref.: \(\text{pri-sched} 3-30(112)\)

Note: A pointer \(p\) is assumed to be referenced in a state \(\xi(k)\), if the result of executing the instruction executed in \(\xi(k)\) depends on the contents of \(p\). The existence of an instruction with the above property and ready for execution in \(\xi(k)\) must be required, since only in this case the executed instruction is uniquely determined by \(\xi(k+1)\).

is-\(e\)-dependent\(p,\xi,tn,e\) =
\[
(\exists vr) (\text{size}(vr) = \text{size-1}(p) \land vr \neq p\cdot e\cdot s(e) \land e_3 \neq e_4)
\]
where:
\[
\begin{align*}
\xi_1 & = \text{compute}(\mu(\xi;\langle s\cdot tn;tn\rangle),e)\\
\xi_3 & = \mu(\xi;\langle s\cdot s\cdot e\cdot ss(vr,p,e\cdot s(e))\rangle)\\
\xi_4 & = \text{compute}(\mu(\xi_3;\langle s\cdot tn;tn\rangle),e)\\
\xi_5 & = \mu(\xi_3;\langle s\cdot s\cdot e\cdot ss(\xi_1,p,e\cdot s(e))\rangle)
\end{align*}
\]
Ref.: \(\text{size} 3-15(33)\)
\(\text{size-1} 3-15(38)\)
\(\text{e\cdot ss} 3-16(48)\)

is-referenced-1\(p,k,tn,cp\) =
\[
is\text{-referenced}(p,k,tn,cp) \land \neg(\exists i,e-1,e-2) (e_0 = e-2\cdot e-1 \land \text{is-corresponding}(i,k,tn,e-1,cp) \land \text{in}_i \in \{\text{eval-on source, eval-on char, eval-on ident}\})
\]
where:
\[
\begin{align*}
e_0 & = \text{exec-sel}(\text{elem}(k,cp),\text{elem}(k+1,cp),tn)\\
in_1 & = s\cdot in\cdot e-1\cdot e\cdot ss(i,cp)
\end{align*}
\]
cont'd
ABSTRACT SYNTAX AND INTERPRETATION OF PL/I

Ref.:  
  eval-resource 12-61(148)  
eval-onchar 12-60(140)  
eval-ident 12-61(145)

(39)  
is-corrresponding(i₁,i₂,tn,ε,cp) =  

  i₁ < i₂ &  
  ba₁ = ba₂ & (∀i)(i₁ ≤ i ≤ i₂ & baₐ = baₐ)  

where:  
  ba₁ = s-ba*tns-paelem(i₁,cp)  
  ba₂ = s-ba*tns-paelem(i₂,cp)  
  baₐ = s-ba*tns-paelem(i,cp)

(40)  
is-mod-alloc-freeed-1(i₁,i₂,j₁,j₂,pₖ,k₁,k₂,tn,cp) =  

  (3i)(i₁ < i < j₁ v j₂ ≤ i ≤ i₂) &  
  (i = k₁ v i = k₂ v k₁ < i < k₂ & is-modified(p₁,i,cp) v k₁ < j₁ v j₂ < k₂) &  
  (3b)(is-ctl-alloc(b₁,pₖ,kₐ,tn,cp) & is-modified-gen(b₁,i,tn,cp)))

(41)  
is-mod-alloc-freeed-2(i₂,j₁,j₂,pₖ,k₁,k₂,tn,cp) =  

  (3j)(j₁ < j < j₂ &  
  (j = k₁ v j = k₂ v k₁ < j < k₂ & is-modified-1(p₁,j,tn,cp) v  
  (k₁ < i₂ v  
  (3b)(is-ctl-alloc(b₁,pₖ,kₐ,tn,cp) &  
  (is-modified-gen(b₁,j,tn,cp) v is-modified-1(p₁,j,tn,cp)))))

where:  
  p₁ = s-pp*head*b*ag*tns-paelem(j,cp)

(42)  
is-mod-alloc-freeed-3(i₁,i₂,j₁,j₂,pₖ,k₁,k₂,tn,cp) =  

  (3i)(i₁ < i < j₁ v j₂ ≤ i ≤ i₂) &  
  (i = k₁ v i = k₂ v k₁ < i < k₂ & is-modified-1(p₁,i,tn,cp) v  
  (k₁ < i₂ &  
  (3b)(is-ctl-alloc(b₁,pₖ,kₐ,tn,cp) & is-modified-gen(b₁,i,tn,cp)))

(43)  
is-modified(p₁,i,cp) =  

  pₛ*ₐ*aelem(i,cp) ≠ pₛ*ₐ*aelem(i + 1,cp)

(44)  
is-modified-1(p₁,j,tn,cp) =  

  is-modified(p₁,j,cp) &  
  (3i)(is-corrresponding(i,j,tn,ε,cp) & inₐ = pseudo-assign &  
  idₐ ∈ {sk-id(ONSOURCE),sk-id(ONCHAR),sk-id(GIDENT)})

where:  
  ε₀ = exec-select(elem(j,cp),elem(j + 1,cp),tn)  
  inₐ = ε₀*ₐ*aelem(i,cp)  
  idₐ = s-id*aelem(1,s*ₐ*aelem(i,cp))

Ref.:  
  pseudo-assign 12-68(172)

(45)  
is-ctl-alloc(b₁,pₖ,kₐ,tn,cp) =  

  p = s-pp*head*b*ag*tns-paelem(k,cp) &  
  (3b)(b = nₛ*ₐ*aelem(k,cp) & is-ctl-s-tg-cl*nₛ*ₐ*aelem(k,cp))
13.2.2 THE RECURSIVE ATTRIBUTE

A computation cp is erroneous because of the wrong use of the RECURSIVE attribute in the interpreted program text, if the following conditions hold:

1) There exists a section \( <\xi(i_1), \ldots, \xi(i_2)> \) as part of the computation cp which constitutes (disregarding steps belonging to another task) the computation of a procedure body.

2) There exists a section \( <\xi(j_1), \ldots, \xi(j_2)> \) as part of the above section which constitutes a computation of the same procedure body.

3) For both sections this procedure body belongs to the same procedure.

4) This procedure body is not declared with the RECURSIVE attribute.

\[
\text{is-recursive-erroneous}(cp) =
\begin{cases} 
(x_1, i_1, i_2, s-1, j_1, j_2, s) & \text{if } (i_1 < j_1 < j_2 < i_2) \text{ and } \text{is-proc-section}(s, i_1, i_2, s) \text{ and } \\
\text{is-rec-recursive-s-body}(arg_1) & \text{and } \end{cases}
\]

where:

\[
\begin{align*}
arg_1 &= \text{elem}(1, s-\text{a}*s-c*\text{tn}s-pa*elem(i_1, cp)) \\
arg_2 &= \text{elem}(1, s-\text{a}*s-c*\text{tn}s-pa*elem(j_1, cp))
\end{align*}
\]

\[
\text{is-rec-recursive-s-body}(arg_1) =
\begin{cases} 
\text{a} & \text{if } \text{term-node}(c_1) \text{ and } s-*s(c_1) = \text{call-proc} \text{ and } \\
\text{a} & \text{else }
\end{cases}
\]

where:

\[
\begin{align*}
c_1 &= s*\text{tn}s-pa*elem(i_1, cp) \\
c_2 &= s*\text{tn}s-pa*elem(i_2, cp)
\end{align*}
\]

Ref.: call-proc 6-16(43)
APPENDIX : CROSS-REFERENCE INDEX

This index lists all names used in the document, with the exception of names of selectors (i.e., names prefixed by s-), metavariables and abbreviations. Selectors to state components in basic instructions are referenced however. Formulas are referenced to by the form XX-YYY(ZZZ), where XX is the number of the main chapter, YYY is the page number within the main chapter and ZZZ is the number of the formula within the main chapter. The following conventions hold:

1) For names defined in chapter 1 only the defining formula is given.

2) For all other names all instances of use in a formula are given. The defining formula is indicated by an underlined reference.

3) Occurrences of names of the form phase-f (cf. 1.5) are listed under the entry f.
   Occurrences of names of the form is-pred-suffix (cf. 1.2.3.3), where suffix stands for list, list-1 or set, or one of these followed again by -suffix, are listed under the entry is-pred.
   Occurrences of names of the form is-OBJ (cf. 1.2.3.3) are listed under the entry OBJ.
   Occurrences of names defined by the predicate scheme is-e-[pred] (cf. 1.4.1) are listed under the entry is-e-[pred].
<table>
<thead>
<tr>
<th><strong>FUNCTION</strong></th>
<th><strong>REFERENCES</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>A-CHAR</td>
<td>2-9(76), 9-4(8), 9-59(235)</td>
</tr>
<tr>
<td>ABNL</td>
<td>5-7(22), 5-7(24), 5-8(26), 11-30(05)</td>
</tr>
<tr>
<td>abs(x)</td>
<td>1-13(271)</td>
</tr>
<tr>
<td>ACC</td>
<td>2-15(126), 3-18(67), 3-19(72), 10-5(10), 10-16(49), 10-17(51)</td>
</tr>
<tr>
<td>acc-infix-nun(v-1,v-2,opr)</td>
<td>9-15(71), 9-15(70)</td>
</tr>
<tr>
<td>acc-prefix-nun(v,opr)</td>
<td>9-27(118), 9-27(113)</td>
</tr>
<tr>
<td>ACC-1</td>
<td>3-18(67), 10-15(43), 10-15(44)</td>
</tr>
<tr>
<td>ACCESS</td>
<td>2-14(124)</td>
</tr>
<tr>
<td>ACOS</td>
<td>12-11(26)</td>
</tr>
<tr>
<td>ACOS-CPLX</td>
<td>12-52(112)</td>
</tr>
<tr>
<td>ACOS-REAL</td>
<td>12-52(112)</td>
</tr>
<tr>
<td>activate-tasks(pa)</td>
<td>5-12(42), 3-32(123), 5-7(25), 5-8(28), 6-9(23), 10-9(24), 11-42(115), 11-46(126)</td>
</tr>
<tr>
<td>active-descendants(ei,pa)</td>
<td>5-7(23), 5-6(18), 5-6(19), 5-7(20)</td>
</tr>
<tr>
<td>ADD</td>
<td>2-18(155), 3-18(149), 9-10(44), 9-15(71), 9-24(103), 9-24(104), 11-78(218), 12-58(131), 12-59(135), 13-5(10)</td>
</tr>
<tr>
<td>add-one(op)</td>
<td>11-78(218), 11-78(217), 11-80(223), 11-97(260)</td>
</tr>
<tr>
<td>adjust-elem(id,i,el,at)</td>
<td>12-10(25), 12-9(24)</td>
</tr>
<tr>
<td>adjust-string(base,n,v)</td>
<td>9-8(32), 9-8(30), 9-13(66), 9-17(77), 9-51(205), 12-37(49), 12-42(68)</td>
</tr>
<tr>
<td>adjust-substr-1(edav1-0,i,j,v1)</td>
<td>11-93(251), 11-62(161), 11-91(288)</td>
</tr>
<tr>
<td>adjust-substr-2(edav1-0,i,j,v1)</td>
<td>11-93(252), 11-93(251)</td>
</tr>
<tr>
<td>adjust-val(edav,v)</td>
<td>9-8(30), 9-7(27), 9-9(38)</td>
</tr>
<tr>
<td>AGG</td>
<td>4-9(24), 5-4(10), 5-8(27), 6-11(32), 6-14(36), 6-22(58), 6-9(23), 6-20(74), 7-2(2), 7-4(7), 7-5(8), 7-10(24), 7-11(25), 7-18(58), 8-11(27), 11-77(223), 12-12(30), 12-62(150), 12-66(161)</td>
</tr>
<tr>
<td>AGGR</td>
<td>12-7(19), 12-10(25)</td>
</tr>
<tr>
<td>agr-builtin(id,expr-list,at)</td>
<td>12-7(18), 6-25(71)</td>
</tr>
<tr>
<td>agr-builtin-1(id,list,at)</td>
<td>12-8(21), 12-8(20)</td>
</tr>
<tr>
<td>agr-builtin-2(id,list,at)</td>
<td>12-9(24), 12-8(20)</td>
</tr>
<tr>
<td>agr-c-exp(aggr,const)</td>
<td>8-23(63), 9-23(62), 6-23(63)</td>
</tr>
<tr>
<td>agr-entry(aggr,ap)</td>
<td>8-25(72), 8-25(71), 8-25(72)</td>
</tr>
<tr>
<td>agr-expr(expr,at)</td>
<td>8-23(62), 6-24(63), 6-35(99), 8-7(13), 8-23(62), 8-26(76), 9-28(91), 11-77(215), 12-4(6), 12-5(9), 12-7(19), 12-10(25), 13-5(9)</td>
</tr>
<tr>
<td>agr-expr-1(expr,at)</td>
<td>5-35(98), 6-23(62), 6-35(97), 6-35(98), 13-11(22)</td>
</tr>
<tr>
<td>agr-generic(ref,ap,at)</td>
<td>8-26(74), 8-25(71)</td>
</tr>
</tbody>
</table>
AGGR-REF(ref, at) .......................... 8-23 (142), 9-36 (142)
AGGR-STATE(vr) ............................ 9-36 (142)
AGGR-BASED(b, eva) .......................... 7-10 (24), 5-2 (3), 6-8 (22)
AGGR-BASED-BUFFER(eva, n, key) ................. 11-59 (155), 11-58 (154)
AGGR-BUFFER(ev) .............................. 11-58 (154)
AGGR-CTL(b, eva) ................................ 7-5 (8), 7-8 (7)
ALLOCATE .......................................... 2-13 (110)
ALLOCATE-AREA(gen, eva, area) .................. 7-3 (20), 7-4 (7), 7-8 (20)
ALLOCATE-AREA-BUFFER(gen, eva, area, n, key) .... 11-59 (155), 11-58 (154)
ALLOCATE-ASSIGN(b, op) .......................... 6-28 (74), 6-28 (73), 12-11 (29)
ALLOCATE-AUTO(b, eva) ........................... 7-10 (24), 5-2 (3), 6-8 (22)
ALLOCABLE(list, vr) ............................. 9-36 (142), 9-36 (141)
ALLOCATE(b, eva, type) .......................... 7-11 (25), 8-9 (24), 6-23 (62), 6-28 (74), 10-14 (41), 10-24 (70), 11-32 (92), 11-48 (132)
ALL .................................................. 2-16 (139), 7-51 (138), 11-75 (211), 11-87 (238)
ALLOC-SIZE(eva) .................................. 7-11 (26), 4-2 (3), 5-3 (5), 5-35 (98), 6-40 (114), 8-15 (37), 8-20 (55), 8-22 (60), 8-23 (62), 8-23 (63), 8-24 (66), 8-25 (69), 8-25 (71), 9-9 (36), 9-13 (64), 9-26 (109), 10-24 (70), 11-21 (56), 11-32 (92), 11-37 (103), 11-48 (132), 11-64 (164), 11-87 (238), 11-99 (264), 11-106 (283), 12-4 (6), 12-5 (11), 12-6 (14), 12-9 (24), 12-36 (48), 12-36 (47), 12-37 (52), 12-38 (54), 12-39 (62), 12-41 (66), 12-43 (73), 12-44 (84), 12-46 (90), 12-47 (92), 12-48 (98), 12-50 (106), 12-52 (111), 12-53 (115), 12-55 (130), 12-58 (133), 12-59 (135), 12-68 (172), 12-68 (173), 12-69 (174), 12-73 (181), 9-35 (140), 11-59 (155), 11-59 (156), 12-42 (72), 12-68 (173), 11-64 (165), 11-66 (166), 11-76 (213)
ALLOC-BASED-BUFFER(eva, n, key) ............... 11-59 (155), 11-58 (154)
ALLOC-BUFFER(et) ................................ 11-58 (154)
ALLOC-CTL(b, eva) ................................ 7-5 (8), 7-8 (7)
APPENDIX .......................................... 3
ABSTRACT SYNTAX AND INTERPRETATION OF PL/I

30 April 1969

array-assign(z,vr) ................................. 9-36(143), 9-35(140), 9-36(144)
array-conv(eva-vr,op) ........................................ 9-25(143), 9-29(120)
area-expr(expr,at) ....................................... 8-28(187), 6-24(63), 6-28(73), 6-47(132), 7-3(4), 7-2(7), 7-9(21), 7-16(52), 7-17(58), 8-15(38), 8-19(50)
area-test(et,add,gen,cond) ..................................... 11-66(166), 11-60(158), 11-65(166)
arg-check-list(expr-list,arg1) .............................. 6-15(49), 6-14(39), 6-15(40)
arg-class(id,i) ........................................... 12-14(33), 8-12(29), 8-28(79), 12-2(2), 12-4(7), 12-5(12), 12-6(17), 12-7(18), 12-8(20), 12-9(21), 12-10(25)
arg-edu(id,i) .............................................. 12-15(35), 12-4(6), 12-5(9), 12-10(25)
arg-list(ap) ............................................... 8-9(61), 6-6(18), 8-4(4), 8-9(20), 8-20(55), 8-21(56), 8-25(71), 8-26(76), 10-22(67), 12-3(3)
arg-test(id,i) ............................................... 12-16(36), 12-4(6), 12-5(9), 12-5(12)
arg-type(id,i) .............................................. 12-15(34), 12-4(5)
array .......................................................... 12-5(10), 12-7(19), 12-10(25)
array-elem(id,i,el) .......................................... 12-9(22), 12-8(20)
array-eva-builtin(id,expr-list,i) .......................... 8-27(79), 8-26(76), 8-27(78)
array-eva-builtin-arg(id,expr,i) .......................... 8-28(79), 8-27(78)
array-eva-expr(expr) .................................. 8-26(261), 6-24(63), 8-3(3), 8-26(76), 8-28(79), 11-77(215), 12-6(16)
array-eva-infix(eva-1,eva-2) ........................................ 8-27(77), 8-26(76), 8-27(77), 8-27(78)
ASIN .......................................................... 12-11(26)
ASIN-CPLX ................................................... 12-52(113)
ASIN-REAL ................................................... 12-52(113)
assign(gen,op) ............................................. 8-9(23), 4-4(9), 6-28(74), 7-16(52), 8-8(16), 10-15(42), 10-28(70), 11-32(92), 11-48(132), 11-91(248), 12-68(173), 12-71(181), 12-73(184)
assign .......................................................... 2-13(109)
assign-completion(gen,op) .................................... 12-73(184), 12-69(174)
assign-complex(gen-1,gen-2,op) .......................... 12-72(176), 12-68(172)
assign-idto(g,cl,chif,vr,pa) ............................. 11-62(160), 11-64(165), 11-124(334)
assign-imag(gen,op) ......................................... 12-70(179), 12-68(172)
assign-priority(gen,op) ...................................... 12-73(185), 12-69(174)
assign-priority-1(gen,op) ..................................... 12-73(186), 12-73(185)

APPENDIX
ABSTRACT SYNTAX AND INTERPRETATION OF PL/I
ABSTRACT SYNTAX AND INTERPRETATION OF PL/I

attr-expr(expr,at)  ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ......
**ABSTRACT SYNTAX AND INTERPRETATION OF PL/I**

**BLANK** | 2-9 (70), 2-9 (76), 9-3 (6), 9-9 (33), 9-31 (125), 9-31 (125), 9-46 (180), 9-47 (189), 9-50 (201), 9-56 (219), 9-57 (228), 9-57 (228), 9-59 (235), 11-69 (190), 11-69 (192), 11-70 (193), 11-70 (194), 11-72 (203), 11-87 (237), 11-89 (241), 11-96 (257), 11-101 (271), 11-103 (272), 11-103 (273), 11-104 (275), 11-104 (275), 11-105 (280), 11-107 (287), 11-108 (293), 11-110 (296), 11-117 (314), 11-117 (315), 11-117 (317), 11-118 (322), 12-56 (140)

**BLOCK** | 3-8 (16), 6-2 (1), 6-5 (11), 6-29 (76)

**BOOL** | 2-14 (121), 3-20 (75), 3-22 (94), 3-26 (96), 11-10 (26), 10-26 (77), 11-27 (78), 11-31 (89), 11-32 (90), 11-32 (90), 11-33 (94), 11-34 (95), 11-35 (97)

**BPIC** | 2-10 (84), 11-81 (225), 11-84 (232), 11-85 (235), 11-99 (265), 11-99 (266), 11-100 (267), 11-115 (312)

**BREAK** | 3-8 (16), 6-2 (1), 6-5 (11), 6-29 (76)

**BSPACE** | 2-10 (86), 11-32 (227), 11-62 (228), 11-83 (229), 11-94 (232)

**BST** | 2-3 (19), 3-26 (98), 4-9 (12), 11-4 (12), 11-9 (28), 11-28 (80), 11-28 (81), 11-28 (81), 11-29 (232), 11-29 (233), 11-34 (97)

**BU** | 2-3 (19), 3-26 (98), 4-9 (22), 11-4 (10), 11-22 (61), 11-25 (73), 11-25 (75), 11-25 (75), 11-26 (80), 11-35 (85), 11-61 (159), 11-62 (160)

**BUITER** | 11-59 (155), 11-61 (159), 11-62 (160)

**BUFFER-TRANSMISSION** | 11-55 (150), 11-49 (156), 11-54 (141)

**BUILDIN** | 2-2 (12), 4-1 (11), 4-4 (6), 4-6 (13), 4-7 (14), 4-9 (23), 5-6 (15), 6-8 (22), 6-35 (58), 7-17 (58), 8-4 (4), 8-6 (10), 8-9 (20), 8-11 (27), 8-17 (68), 8-26 (71), 8-26 (76), 10-22 (67), 11-30 (104), 13-8 (74)

**builtin-excons(id,id,expr)** | 12-9 (51), 12-3 (9)

**builtin-excons(id,attr-list)** | 12-9 (51), 12-3 (9)

**C-CHAR** | 6-2 (1), 6-5 (11), 6-16 (43), 6-17 (46), 6-39 (108), 10-20 (69)

**C-EXP** | 9-10 (43), 9-10 (44), 9-13 (64), 9-13 (65), 9-13 (66), 9-14 (68), 9-15 (70), 9-15 (71), 9-21 (89), 9-25 (103)

**C-EXPTARGET-1(eda-1,eda-2,v-2)** | 2-25 (102), 9-13 (68), 9-25 (106)

**C-EXIT** | 8-19 (31)

**CALL** | 2-12 (98)

**CALL-ATTN-COND(cond)** | 10-74 (41), 10-14 (40), 10-17 (50), 10-17 (52), 10-18 (53)
ABSTRACT SYNTAX AND INTERPRETATION OF PL/I

CHECK ................................................................. 2-14 (115)
CHECK-ARG-LIST (expr-list, arg1, n, pa-opt) ...................... 6-14 (39), 6-14 (37)
CHECK-ARG-VAL ......................................................... 11-51 (138), 11-87 (238), 11-123 (331)
CHECK-ENTRY (ref) ...................................................... 5-14 (39), 6-14 (37)
CHECK-ENTRY (list) ..................................................... 11-52 (119), 11-40 (109)
CHI (id) ........................................................................... 11-89 (249), 11-89 (243), 12-61 (144), 12-61 (147)
CI .................................................................................... 6-2 (1), 6-5 (12), 6-5 (12), 6-16 (43), 6-39 (108), 6-39 (109), 6-39 (111), 6-39 (112), 6-45 (123), 6-45 (125), 6-45 (126), 10-20 (60), 11-75 (210), 11-83 (229), 11-88 (230)
CI-INDL (ci) ................................................................. 6-45 (124)
CIPHER (mp, ids) ............................................................ 11-5 (17), 11-6 (18), 11-12 (42), 11-13 (44), 11-14 (45), 11-14 (46), 11-14 (47), 11-27 (79), 11-30 (87), 11-31 (89), 11-83 (240)
CLOSE ................................................................. 5-8 (26), 11-29 (84)
CLOSE-LOCAPP ......................................................... 11-30 (85), 11-43 (117)
close-1 (u, f) ........................................................................ 6-45 (124)
close-2 (u, f) ........................................................................ 6-45 (123), 5-45 (124)
COD ................................................................. 11-29 (84), 5-8 (26), 11-21 (56)
COMMA ................................................................. 11-39 (87), 11-29 (84)
COMMA ................................................................. 11-30 (85), 11-43 (117)
COMMA ................................................................. 11-39 (87), 11-29 (84)
COMMA ................................................................. 11-27 (80), 10-15 (45), 10-18 (54), 10-20 (62), 10-22 (65), 10-24 (71), 10-24 (72), 10-27 (81), 11-27 (78), 11-31 (89), 11-33 (94)
COL ................................................................. 11-29 (84), 5-8 (26), 11-21 (56)
COLLECT-ID (id) ........................................................... 6-24 (71), 6-3 (5), 6-3 (7)
COLLECT-IDL (idl, aggr) ................................................. 6-24 (71), 6-3 (5), 6-3 (7)
COLON .............................................................................. 9-3 (6)
COMMA ................................................................. 9-4 (8)
COMMA ................................................................. 2-9 (70), 2-9 (76), 9-3 (6), 9-46 (180), 9-47 (185), 9-50 (201), 11-70 (194), 11-88 (285), 11-103 (272), 11-103 (273), 11-103 (273), 11-104 (275), 11-104 (275), 11-104 (276), 11-105 (280), 11-106 (287), 11-107 (288), 11-109 (295), 11-116 (296), 11-110 (296), 11-110 (297), 11-110 (298)
COMMENT (cond, cbf) ...................................................... 10-10 (27), 10-10 (27)
COMMENT-F (cond, cbf, f) ............................................... 10-10 (27), 10-10 (27)
COMPL-OP (op) ........................................................... 6-49 (147), 6-48 (133)
COMPL-EVENT (en) ........................................................ 5-13 (43), 5-12 (39)
COMPL-VAL (eda, sf, v) ................................................ 9-54 (211)
COMPL-WAIT (gen, count, w, i) ...................................... 5-12 (39), 5-11 (35)
COMPL-ENTER (gen) .................................................... 5-12 (39), 5-12 (38)
COMPLETE-AR (eda-tg, eda) ........................................... 9-39 (155), 9-37 (149)
COMPLETE-AREA (eda-tg, eda) ....................................... 9-39 (155), 9-37 (149)
COMPLETE-EVA (eva-tg, eva) ........................................... 9-37 (155), 9-29 (119)
ABSTRACT SYNTAX AND INTERPRETATION OF PL/I

complete-m-b-s(eda-tg, eda) ........................................... 9-38 (151), 9-38 (150)
complete-prec(eda-tg, eda) ............................................. 9-38 (152), 9-38 (150)
complete-str (eda-tg, eda) ............................................. 9-38 (154), 9-37 (149)

complete-tg(eda-tg, eda) ............................................. 9-37 (149), 9-20 (87), 9-25 (106), 9-25 (107), 9-28 (117), 9-31 (126), 9-37 (145), 9-40 (159), 12-5 (11), 12-10 (25), 12-44 (75), 12-44 (76), 12-45 (79), 12-45 (82), 12-46 (88), 12-47 (91), 12-47 (93), 12-49 (100), 12-49 (103), 12-51 (108), 12-52 (111), 12-58 (132), 12-58 (134), 12-59 (136)

COMPLETION ............................................................... 12-67 (170), 12-69 (174), 12-74 (189)

CONPLEX ........................................................................ 8-4 (4), 12-67 (170), 12-69 (172), 12-74 (189)

compute(ε, ε) .................................................................. 1-24 (49)

cond(a, b) ........................................................................... 9-52 (207), 9-52 (206), 11-104 (275), 11-104 (276), 11-108 (289), 11-108 (290), 11-109 (298), 11-110 (297), 11-110 (298)

cond-event-list (genl) ......................................................... 5-10 (32), 5-10 (31)

COND ................................................................................. 2-2 (12), 2-14 (122), 4-4 (6), 4-6 (13), 4-9 (24), 6-8 (22), 10-11 (31), 10-12 (36), 10-20 (62)

cond-ct(ct) ........................................................................... 11-32 (90), 11-31 (89), 11-33 (93)

cond-infixedf (eda-1, eda-2, v-1, v-2, opr) ........................... 9-14 (69), 9-14 (67)

cond-io-sel(iocond) ......................................................... 3-20 (75), 3-19 (74)

cond-set(co-set, f, key) ..................................................... 11-55 (142), 11-55 (142), 11-55 (145), 11-57 (151), 11-60 (157), 11-61 (159), 11-64 (164), 11-65 (167), 11-66 (168)

conjg(z) .............................................................................. 1-14 (361), 12-46 (66)

connected-match(ref, descr, at) ........................................ 6-21 (56), 6-19 (48)

connected-match-1 (gen, attr) .......................................... 6-27 (72), 6-27 (71)

const ................................................................................. 12-5 (12), 12-7 (19), 12-10 (25)

construct-mi(mi-1, mi-2, eql) ............................................ 8-14 (34), 8-13 (32), 8-14 (34)

continue ............................................................................ 6-39 (109), 6-39 (108), 6-39 (111), 6-45 (126)

control-format(tr, efo) ...................................................... 11-32 (226), 11-51 (140), 11-81 (225)

conv ...................................................................................... 2-14 (118), 3-19 (74), 10-10 (26), 10-18 (55), 10-24 (70), 10-24 (71), 10-25 (75), 10-26 (78)

conv-char-string-1 (tr, op) .................................................... 11-87 (238), 11-87 (237), 11-88 (240)

conv-scale-f(v, n) .............................................................. 9-31 (127), 9-31 (126)

convert (eva, op, area-1, area-2) .................................... 8-8 (17), 8-8 (18), 6-24 (63), 6-28 (73), 8-8 (16), 10-11 (32), 12-4 (6)

convert-assign (gen, op, area-1, area-2) ..................... 9-8 (16), 6-26 (69), 6-47 (132), 6-49 (140), 7-9 (21), 7-9 (22), 7-16 (51), 7-16 (52), 8-7 (14), 11-80 (223), 12-69 (174), 12-70 (176), 12-70 (177), 12-70 (178), 12-72 (183)

convert-assign-list (genl, op, dal, da) ......................... 8-7 (14), 8-7 (13), 8-7 (14), 12-72 (183)

convert-pri (eva, op) ............................................................. 5-3 (9), 5-3 (7), 5-3 (8), 12-64 (159)
ABSTRACT SYNTAX AND INTERPRETATION OF PL/I

12-37 (51)
da-bool(da-1, da-2) ............................................................. 12-16 (371, 12-9 (24))
da-builtin(id, list) ...................................................................... 12-46 (89)
da-dec(da, p, g) ............................................................................ 12-47 (91)
da-fixed(da, p, g) .......................................................................... 12-46 (90), 12-47 (91)
da-float(da, p) .............................................................................. 12-47 (93)
da-infix(da-1, da-2, opr) ............................................................. 8-24 (67), 8-24 (66)
da-infix-1(da-1, da-2, p, g, opr) ...................................................... 12-46 (76)
da-mod(da-1, da-2) ........................................................................ 12-46 (106), 12-46 (98)
da-poly(da-1, da-2) ....................................................................... 12-58 (132)
da-prec(da, p, g) ........................................................................... 12-46 (99)
da-prefix(da, opr) .......................................................................... 8-25 (79), 8-25 (69)
da-prod(da) .................................................................................. 12-58 (139), 12-58 (133)
da-round(da) ................................................................................ 12-55 (105), 12-50 (106), 12-51 (108)
da-string(aggr) ............................................................................. 12-39 (61)
da-sum(da) .................................................................................. 12-59 (136), 12-59 (135)
da-trunc(da) ................................................................................ 12-52 (111), 12-45 (83), 12-46 (94), 12-51 (110)

DATAs ........................................................................................... 2-16 (139), 11-51 (138), 11-75 (211), 11-87 (238)
data-lg(edfo) .................................................................................. 11-85 (235), 11-85 (235), 11-86 (236), 11-98 (263), 11-115 (311), 11-115 (312)
data-string(idl, intgl, eda, cl) ........................................................ 11-89 (242), 11-89 (241)

DATAF-LENGTH ............................................................................. 11-29 (83), 12-60 (138)

D -E -F- L E N G T H ........................................................................ 13-12 (26)

decipher(mp, ds) ........................................................................... 11-5 (15), 11-5 (16), 11-6 (18), 11-6 (19), 11-6 (20), 11-7 (22), 11-7 (23), 11-8 (24), 11-8 (26), 11-8 (27), 11-9 (28), 11-10 (36), 11-10 (38), 11-11 (40), 11-27 (79), 11-45 (123), 11-60 (157)

ded-attr(t, alt) ............................................................................. 11-25 (75), 11-24 (67)
def-copy(u) .................................................................................. 11-121 (328), 11-121 (327)
def-DA-1 ....................................................................................... 5-8 (26), 11-22 (60)
def-DA-2 ....................................................................................... 11-22 (60), 11-26 (76)
def-LSZ-1 ...................................................................................... 11-28 (81), 11-29 (83)
def-LSZ-2 ...................................................................................... 11-28 (81), 11-29 (83)
def-LSZ-3 ...................................................................................... 11-28 (81), 11-29 (83)
def-PREC-BIN ............................................................................. 9-11 (50), 12-47 (91)
def-PREC-DEC ............................................................................ 9-11 (49), 12-47 (91)

12 APPENDIX
ABSTRACT SYNTAX AND INTERPRETATION OF PL/I
ABSTRACT SYNTAX AND INTERPRETATION OF PL/I

30 April 1969

DUMMY...

dummy-assign(gen, dop) 6-25(69), 6-23(62), 6-26(69)
dummy-da(dal) 6-25(67), 6-25(66)
dummy-eva(dop) 6-25(65), 6-23(62)
dummy-eva-1(dopl) 6-25(66), 6-25(65), 6-25(66)
Z-CHAR 2-3(69), 9-4(6), 9-60(239), 11-68(162), 11-101(270)
edu-c-exp(edu-1, const) 9-25(106), 8-23(63)
edu-f(m, b, s, p, q) 12-36(45), 12-36(44), 12-44(76), 12-45(79), 12-45(82), 12-46(88), 12-87(91), 12-47(93), 12-49(100), 12-49(103), 12-52(111), 12-58(134), 12-59(136)
edu-infix(edu-1, edu-2, opr) 9-20(87), 8-24(67), 12-44(75)
edu-prefix(edu, opr) 9-28(117), 8-25(70)
edu-type(edu) 9-30(122), 8-29(121)
EDIT...

el-alloc(p, vr, type) 3-17(55), 3-17(58), 3-17(59), 3-17(60), 7-5(8), 7-8(20), 7-9(23), 7-10(24), 10-20(60)
el-ass(vr-1, p, vr) 3-16(48), 3-16(49), 3-16(50), 3-16(51), 3-16(52), 3-17(61), 5-4(10), 5-8(28), 7-8(20), 7-19(60), 8-9(33), 10-7(19), 11-41(113), 11-56(147), 11-59(156), 11-62(160), 11-64(165)
el-free(ps, vr) 3-17(62), 3-17(63), 3-17(64), 3-17(65), 5-8(27), 6-11(32), 7-18(58), 7-18(59), 7-19(60), 11-41(113)
elem(i) 3-7(12)
elem-bool(bit-1, bit-2, bit-list) 12-37(591), 12-37(49)
elem-trans(char, repl, pos) 12-42(69), 12-42(68)
EMPTY-OF

EN...

enab-wait

enable(enable, cond) 10-8(19), 10-4(8), 10-8(20)
ENABLE 2-15(125)
enabled

APPENDIX
ABSTRACT SYNTAX AND INTERPRETATION OF PL/I

END .... 11-2(1), 11-9(30), 11-9(32), 11-10(33), 11-12(42), 11-12(42), 11-13(44), 11-13(44), 11-14(45), 11-14(46), 11-15(47), 11-15(47), 11-34(19), 11-45(123), 11-45(125), 11-55(142), 11-56(144), 11-56(145), 11-56(146), 11-61(159), 11-64(164), 11-65(167), 11-95(255)

end-index(cond-list) ........................................ 11-25(125), 11-42(115), 11-43(118), 11-44(120)

END ... 2-14(121), 3-20(75), 3-26(96), 10-26(77), 11-27(78), 11-34(96), 11-35(37), 11-41(112), 11-45(124), 11-50(134), 11-50(134), 11-56(146), 11-95(256), 17-96(256)

ENDM-SCAN ... 11-96(256), 11-104(275), 11-104(276), 11-108(289), 11-108(290), 71-110(296), 11-110(297), 11-170(298), 11-112(303), 11-112(305)

ENDP ... 2-14(121), 3-20(75), 10-10(26), 10-26(77), 11-97(259), 11-97(260), 12-97(261)

ENTRY ... 2-7(58), 2-20(173), 2-21(184), 4-6(13), 4-9(24), 6-10(26), 6-33(35), 6-35(98), 6-37(103), 7-15(50), 7-16(51), 7-16(52), 8-7(13), 8-10(25), 8-21(57), 9-2(1), 9-4(13), 9-37(146), 13-11(22)

entry-val-list(opl) ........................................... 6-37(103), 6-37(104)

EOV ... 2-14(121), 3-20(75), 3-22(34), 3-26(96), 10-10(26), 10-26(77), 11-29(84), 11-30(86), 11-32(90), 11-33(94), 11-34(95)

EOV-EOV ... 11-27(78), 11-34(95), 11-48(120), 11-95(255)

ecv-bov-cond(u,gen,ct) ...................................... 11-33(93), 11-32(92)

epilogue ... 6-11(31), 5-7(22), 6-2(3), 6-29(76), 6-44(121), 10-14(40), 10-29(60)

EQ ... 2-18(155), 9-3(6), 9-10(45), 9-17(77), 9-17(77), 9-18(80), 9-18(81), 11-89(242), 11-107(288), 11-108(289), 11-108(290), 13-5(10)

equal-intg(expr-1,expr-2) .................................. 8-36(113), 8-35(112)

erf(x) .............................................................. 12-59(121)

dep ... ......................................................... 12-11(26)

error ... .......................................................... 12-11(26)

error ... .......................................................... 1-30(52)

error ... .......................................................... 1-30(51)

dir ... 2-14(117), 3-19(74), 5-4(10), 7-18(53), 7-18(59), 7-19(60), 8-9(23), 9-14(68), 10-7(19), 10-10(26), 10-10(26), 10-18(54), 10-18(55), 10-25(73), 11-24(69), 11-41(113), 11-91(248), 11-108(291), 11-112(305)

ERROR-ENDF .................................................... 11-96(256), 11-103(272), 11-107(287), 11-111(300), 11-112(305)

error-exec ........................................................ 10-20(59), 10-18(59)

ERS ... 11-27(79), 11-30(87), 11-31(89), 11-33(94), 11-55(142), 11-57(151), 11-60(157), 11-61(159), 11-64(164), 11-65(167), 11-66(168), 11-94(254)

ET ... 5-8(28), 5-10(30), 5-11(35), 5-12(40), 5-12(41), 8-9(23)

EV ... 8-9(24), 10-4(8), 10-11(31), 10-12(36), 10-16(46), 10-20(62), 12-65(165)

eva-f(m,b,s,p,g) ................................................ 12-36(44), 12-45(81), 12-46(87), 12-49(102)

eval-abs(op) ........................................................ 12-44(77)
<table>
<thead>
<tr>
<th>Function</th>
<th>References</th>
</tr>
</thead>
<tbody>
<tr>
<td>eval-acos (op)</td>
<td>12-52(112)</td>
</tr>
<tr>
<td>eval-add (op-1, op-2, p, q)</td>
<td>12-25(60)</td>
</tr>
<tr>
<td>eval-add (ref)</td>
<td>12-62(150)</td>
</tr>
<tr>
<td>eval-aggr (aggr)</td>
<td>6-9(23), 6-8(22), 6-9(23), 7-4(7), 7-7(18)</td>
</tr>
<tr>
<td>eval-aggr-1 (aggr)</td>
<td>6-9(23), 6-9(25), 6-24(63), 6-27(71), 6-28(73), 6-13(32)</td>
</tr>
<tr>
<td>eval-all (op1)</td>
<td>12-57(124)</td>
</tr>
<tr>
<td>eval-alloc-aggr (aggr)</td>
<td>7-7(18), 7-7(18), 7-7(18), 11-58(154)</td>
</tr>
<tr>
<td>eval-alloc-extent (extent)</td>
<td>7-7(19), 7-7(18)</td>
</tr>
<tr>
<td>eval-allocation (t)</td>
<td>12-64(161)</td>
</tr>
<tr>
<td>eval-ary (op1)</td>
<td>12-57(129)</td>
</tr>
<tr>
<td>eval-arg (expr, descr)</td>
<td>6-19(48), 6-18(47)</td>
</tr>
<tr>
<td>eval-arg-list (expr-list, descr)</td>
<td>6-16(47), 6-14(37), 6-18(47)</td>
</tr>
<tr>
<td>eval-arg-1 (p, type)</td>
<td>6-19(49), 6-19(48), 6-22(59), 6-23(62)</td>
</tr>
<tr>
<td>eval-asin (op)</td>
<td>12-52(113)</td>
</tr>
<tr>
<td>eval-atan (op, op-1)</td>
<td>12-52(114)</td>
</tr>
<tr>
<td>eval-atanh (op, op-1)</td>
<td>12-53(115)</td>
</tr>
<tr>
<td>eval-atanh (op)</td>
<td>12-53(116)</td>
</tr>
<tr>
<td>eval-attn(idr, dn, ev)</td>
<td>10-7(16), 10-7(15), 10-20(62), 12-65(165)</td>
</tr>
<tr>
<td>eval-attn-list(idrl, dn, ev)</td>
<td>10-7(15), 10-4(3), 10-7(15), 10-8(22), 10-11(31), 10-12(36), 10-16(46)</td>
</tr>
<tr>
<td>eval-based-aggr (aggr, p, eql)</td>
<td>8-13(30), 8-11(27)</td>
</tr>
<tr>
<td>eval-based-aggr-1 (aggr, p, eql)</td>
<td>8-13(32), 8-13(30), 8-13(32)</td>
</tr>
<tr>
<td>eval-bin (op, p, g)</td>
<td>12-45(81)</td>
</tr>
<tr>
<td>eval-bit (op, i)</td>
<td>12-33(87)</td>
</tr>
<tr>
<td>eval-bool (op-1, op-2, op-3)</td>
<td>12-37(88)</td>
</tr>
<tr>
<td>eval-builtin (ref)</td>
<td>12-3(31), 8-20(55)</td>
</tr>
<tr>
<td>eval-builtin-1 (id, bal)</td>
<td>12-17(38), 12-3(3), 12-11(29), 12-74(169)</td>
</tr>
<tr>
<td>eval-by-to(by, to)</td>
<td>6-48(133), 6-47(132)</td>
</tr>
<tr>
<td>eval-coil (op)</td>
<td>12-45(83)</td>
</tr>
<tr>
<td>eval-ceil (op)</td>
<td>12-45(83)</td>
</tr>
<tr>
<td>eval-ceil (op, i)</td>
<td>12-37(82)</td>
</tr>
<tr>
<td>eval-com (gen, by-to-ops)</td>
<td>6-48(133), 6-48(133)</td>
</tr>
<tr>
<td>eval-completion (op)</td>
<td>12-64(158)</td>
</tr>
<tr>
<td>eval-const (op)</td>
<td>12-46(86)</td>
</tr>
<tr>
<td>eval-const (const)</td>
<td>8-20(52), 8-19(50)</td>
</tr>
</tbody>
</table>
eval-count (op) ........................................... 12-48(84), 11-105 (282), 11-115 (312), 12-50 (106)

eval-cplx (op-1, op-2) ...................................... 12-46(85), 12-46(84)

eval-cplx-abs (op) ........................................... 12-44(78), 12-44(77)

eval-cplx-1 (op-1, op-2) ...................................... 12-46(85), 12-46(84)

eval-da (da) .................................................. 6-9(24), 6-7(16), 6-9(23)

eval-da-1 (da) .................................................. 6-10(25), 6-9(24), 6-9(25), 6-20(53), 6-21(55), 6-34(91), 8-14(36), 9-37(49)

eval-datafield .................................................. 12-60(138)

eval-date ........................................................ 12-55(163)

eval-def-gen (ref) ............................................. 8-32(192), 8-29(88)

eval-den (id, attr, body) ...................................... 6-8(22), 6-5(13)

eval-dir (ev, i) .................................................. 12-57(127), 12-57(127)

eval-divide (op-1, op-2, p, q) ................................... 12-46(89)

eval-dummy-arg (expr, agr) .................................... 6-23(62), 6-19(48)

eval-dummy-expr (expr, agr) ................................... 6-24(63), 6-23(62), 6-24(63), 6-24(64), 6-37(101), 12-5(9)

eval-empty ...................................................... 12-52(151), 7-16(52)

eval-enabled (ref) ............................................. 12-65(165)

eval-entry (n, da, ap, ref) .................................... 9-18(49), 8-17(48), 8-18(49)

eval-entry-expr-refer (ref, descr1) ............................. 8-17(49), 6-13(34), 8-17(48)

eval-entry-expr-1 (ref, descr1) ............................... 8-17(49), 8-10(25), 8-17(47), 8-17(48), 13-3(7)

eval-entry-ref (n, da, ap, ref) ................................ 8-21(56), 8-20(55), 8-21(57)

eval-entry-ref-1 (op, da, ap) ................................... 9-21(57), 8-21(56)

eval-env-attr (env-attr) ...................................... 11-21(57), 4-6(13), 11-21(56)

eval-ext (expr, da) ............................................. 8-10(24), 4-8(18), 5-3(4), 6-28(73), 6-40(114), 6-47(132), 6-48(133), 7-16(52), 8-22(60), 8-22(61), 11-21(56), 11-37(103), 11-48(38), 12-4(7)

eval-ext-1 (expr, da) .......................................... 8-10(25), 6-24(63), 8-7(13), 8-10(24)

eval-ext-2 (expr) .............................................. 8-10(50), 8-10(25), 8-19(50), 11-79(221), 12-4(7), 12-6(14), 12-39(62), 12-50(105), 13-3(7)

eval-extref (extent, gen, agr) ................................ 8-15(37), 8-14(36)

eval-fixed (op, p, q) .......................................... 12-46(90)

eval-float (op, p) ............................................. 12-47(92)

eval-floor (op) ............................................... 12-57(98)

eval-format (tr) ................................................ 11-93(229), 11-81(225)

eval-function (n, expr-list, entry, ref) ........................ 6-13(35), 8-15(49), 8-21(56)

eval-gen (ref) .................................................. 8-23(88), 8-29(87)

APPENDIX 17
eval-gen-arg(ref,aggr)                       6-21(57), 6-19(46)
eval-gen-arg-1(b,aggr)                      6-22(59), 6-21(57)
eval-bound(eval)                           -12-57(128), 12-57(128)
eval-high(k)                               -12-38(53)
eval-imag(op)                              12-47(95), 12-50(106)
eval-index(op-1,op-2)                      -12-38(54), 12-38(54)
eval-infix-conversion(op-x,op-y,p,g,op)    12-48(75), 12-45(80), 12-46(89), 12-49(101)
eval-infix-expr(op-1,op-2,op)              9-13(60), 5-3(8), 6-48(135), 6-49(140), 8-19(50), 11-78(218), 12-58(131), 12-60(159)
eval-int-expr(expr)                       .8-22(60), 4-7(16), 4-8(18), 5-10(31), 5-15(48), 6-9(23), 6-9(24), 7-9(19), 7-16(52), 8-30(91), 8-32(100), 11-21(56), 11-37(103), 11-83(229), 12-5(12)
eval-int-expr-1(expr)                        12-6(14), 12-5(12), 12-6(170)
eval-int-gen(ref,rl)                         8-31(95), 8-29(87)
eval-lbound(eval)                           -12-57(129), 12-57(129)
eval-length(op)                            -12-38(56)
eval-linxor(op)                             12-56(166)
eval-log(op)                                12-53(117)
eval-log-10(op)                              12-54(118)
eval-log-2(op)                              12-58(139)
eval-low(k)                                 -12-38(57)
eval-lp(ref)                                 8-9(29), 6-47(130), 8-9(19), 11-37(103), 11-48(132), 11-77(215), 12-39(62)
eval-lp-list(lp)                             8-9(29), 6-7(13), 8-9(19)
eval-max(op1)                                12-48(96)
eval-min(op1)                                12-48(97)
eval-mod(op-1,op-2)                          12-98(98)
eval-multiply(op-1,op-2,p,q)               12-49(101)
eval-null                                    12-62(153)
eval-nullo                                  12-63(154)
eval-offset(op,gen)                        12-63(156), 8-8(18)
eval-opattr                                 12-60(139)
eval-onchar                                 12-60(140), 13-16(38)
eval-oncode                                 12-60(142)
eval-oncount                               12-50(143)
eval-onfile                                 12-51(184)
ABSTRACT SYNTAX AND INTERPRETATION OF PL/I

- 12-61(145), 13-76(38)
- 12-61(146)
- 12-61(147)
- 12-61(148)
- 12-62(149), 10-24(70), 12-61(148)
- 8-22(611), 6-48(133)
- 12-57(130)
- 12-59(131), 12-57(130), 12-58(131)
- 12-49(102)
- 10-3(4), 10-2(1), 10-3(4)
- 9-26(109), 8-19(50)
- 5-3(7), 5-2(1)
- 5-3(8), 5-3(7), 12-73(185)
- 12-54(159)
- 12-58(133)
- 12-67(170), 8-9(20)
- 12-63(157), 8-8(18), 8-16(39)
- 8-15(38), 8-11(27)
- 8-19(94), 7-9(22), 8-15(37), 6-25(71), 8-26(75), 8-30(94), 8-34(109), 8-35(111), 10-22(67), 11-79(221), 13-10(17)
- 12-49(109), 12-50(106)
- 8-20(54), 4-4(8), 6-43(119), 8-15(38), 8-20(55), 10-11(32), 11-21(56), 11-23(64), 11-83(229)
- 8-28(82), 5-2(3), 5-10(31), 6-21(57), 7-4(7), 7-8(20), 7-9(21), 7-17(56), 8-5(18), 8-16(39), 8-31(95), 9-32(100), 10-4(8), 11-37(103), 11-58(154), 11-59(156), 11-60(158), 11-65(166), 11-109(235), 12-5(17), 12-62(150)
- 8-28(83), 8-9(20), 8-22(58), 8-28(82), 12-6(17), 12-67(170)
- 8-20(55), 8-19(50), 8-20(54)
- 8-14(36), 8-13(32), 8-14(36)
- 12-39(59)
- 12-39(59), 12-39(58), 12-39(59)
- 12-50(105)
- 12-50(106), 12-50(105), 12-50(106)
- 12-51(109)
- 11-80(222), 11-79(221), 11-80(222)

APPENDIX 19
ABSTRACT SYNTAX AND INTERPRETATION OF PL/I

30 April 1969

**eval-sort**(op) ........................................ 12-54(120)
**eval-static-aggr**(aggr,k) ......................... 4-7(16), 4-6(13), 4-7(16)
**eval-static-init**(init,eva,k) ..................... 4-8(18), 4-7(16), 4-8(18)
**eval-status**(op) ....................................... 12-64(160)
**eval-string**(op1) ..................................... 12-39(69)
**eval-sub-gen**(ref,rl) .................................. 8-29(87), 8-28(83), 11-79(221)
**eval-substr**(t,i,j) ..................................... 12-80(63), 12-39(62), 12-62(149), 12-74(189)
**eval-sun**(op1) ......................................... 12-59(135)
**eval-te-gen**(ref,eda) ................................. 5-2(3), 5-2(2)
**eval-translate**(op-1,op-2,op-3) .................... 12-41(66), 12-41(56)
**eval-trunc**(op) ......................................... 12-51(110)
**eval-truth**(expr) ....................................... 6-40(113), 6-40(113), 6-40(128), 6-49(138)
**eval-unspec**(gen) ..................................... 12-82(70), 12-74(189)
**eval-var-ref**(ref) ..................................... 8-22(58), 8-17(48), 8-20(55)
**eval-verify**(op-1,op-2) ............................... 12-43(73), 12-43(73)
**eval-while**(trut,while) .............................. 6-49(138), 6-48(134)

**EVENT** 2-5(34), 2-6(50), 2-7(58), 2-20(173), 2-21(180), 4-2(2), 5-2(2), 5-2(3), 5-10(32), 8-9(23), 9-2(1), 9-4(13), 9-37(146), 11-37(103), 12-73(184), 12-73(186), 12-73(187), 12-73(187)

**event-transmission**(et,n) ............................ 11-42(119), 11-41(113)

**EXC** 2-3(19), 3-26(98), 4-9(22), 11-4(8), 11-4(10), 11-4(62), 11-24(67), 11-25(74), 11-25(75), 11-47(127), 11-47(128), 11-67(129), 11-47(131)

**exec-sel**(et,et,tn) .................................... 13-16(36), 13-16(35), 13-16(38), 13-17(44)

**EXIT** .................................................. 2-13(107)

**EXP** .................................................. 2-18(155), 9-10(44), 9-14(68), 9-15(70), 9-15(71), 9-21(88), 9-21(89), 9-23(100), 9-25(105), 9-25(107), 9-26(108), 12-11(26)

**exp-sep**(pv) .......................................... 9-58(231), 9-58(230)

**EXP-SIZE** ............................................. 9-46(159), 11-29(63), 11-101(270)

**expand-exp**(tr,expr) .................................. 11-77(215), 11-76(214), 11-77(216), 11-123(331)

**expand-idl**(idl,aggr) ................................ 10-23(68), 10-22(67), 10-23(68)

**expand-item-list**(tr,ttl) ............................ 11-76(214), 6-49(139), 11-74(209), 11-76(218)

**expand-ref**(ref,at) .................................. 10-22(67), 10-22(66), 10-22(67)

**expand-rl**(rl,at) ..................................... 10-22(66), 10-3(4), 10-11(31), 10-12(36), 10-20(62), 10-21(64), 10-22(56), 10-22(67)

**expand-spec**(tr,spec) ................................ 11-74(209), 11-48(132), 11-50(134), 11-74(209)

20 APPENDIX
EXPND  .  8-12(29), 8-29(79), 12-2 (2), 12-4 (7), 12-5 (12), 12-6 (17), 12-7 (19), 12-7 (19), 12-8 (20), 12-8 (21), 12-9 (22), 12-9 (23), 12-10 (25)

EXT       .  2-3 (13), 2-3 (75), 2-3 (18), 3-19 (74), 4-4 (6), 4-4 (7), 6-5 (22), 11-26 (77)

f(e-1, ..., e-n)       .                          1-31 (55)

F-CHAR       .                          9-4 (8), 9-31 (127)

f-e-ed(a, const, d, p)   .                          11-117 (312), 11-115 (312)

f-e-val (ft, const, d, p) .                          11-119 (320), 11-115 (312)

f-in(x-1, x-2, ..., x-n, x1)    .                          1-125 (50)

FD       .  4-9 (24), 5-4 (10), 5-8 (26), 10-20 (62), 10-24 (72), 11-21 (56), 11-23 (66), 11-24 (67), 11-24 (68), 11-26 (77), 11-27 (78), 11-27 (79), 11-29 (84), 11-30 (87), 11-78 (219), 11-94 (254), 11-123 (332), 12-64 (162), 12-66 (166)

fetch (entry-list) .                          6-37 (105), 6-37 (104)

FETCH      .                          2-12 (103), 6-36 (99), 6-37 (104)

fetch-release (ref-list, type) .                          6-37 (101), 6-36 (99), 6-37 (100)

fetch-release-1 (entry-list, type) .                          6-37 (101), 6-36 (99), 6-37 (100)

FILE .  2-5 (34), 2-6 (50), 2-7 (58), 2-20 (173), 2-21 (170), 8-20 (55), 8-25 (71), 9-2 (1), 9-4 (13), 9-17 (77), 9-37 (146), 10-11 (32), 11-21 (56), 11-23 (66)

file-bif(s, u) .                          12-66 (167), 12-64 (162), 12-66 (166)

fill-char (base) .                          9-9 (33), 9-8 (32), 11-93 (252)

final-ret-type (aggr) .                          9-28 (60), 6-33 (85), 7-17 (56), 8-7 (13), 8-28 (80), 8-28 (81), 12-9 (6)

FINISH .  2-14 (117), 3-19 (74), 5-6 (14), 5-6 (16), 6-29 (75), 10-10 (26), 10-18 (54), 10-19 (55)

first (list) .                          1-9 (25)

FIX .  2-5 (35), 2-7 (59), 2-10 (81), 9-5 (17), 9-9 (23), 9-11 (47), 9-11 (49), 9-15 (70), 9-75 (72), 9-16 (73), 9-18 (79), 9-22 (96), 9-22 (96), 9-23 (100), 9-25 (107), 9-26 (108), 9-27 (113), 9-31 (126), 9-37 (147), 9-38 (152), 9-40 (159), 9-40 (161), 9-63 (248), 11-81 (225), 11-86 (232), 11-85 (235), 11-36 (236), 11-99 (266), 11-99 (266), 11-100 (267), 11-101 (271), 11-107 (286), 11-115 (312), 11-116 (313), 11-117 (320), 12-34 (39), 12-34 (41), 12-36 (42), 12-36 (44), 12-36 (45), 12-45 (79), 12-47 (91), 12-51 (107), 12-51 (109), 12-52 (111), 12-58 (134), 12-58 (134), 12-59 (136)

floor(x) .                          1-13 (31)

FLT .  2-5 (35), 2-7 (59), 2-10 (81), 9-5 (17), 9-9 (23), 9-11 (47), 9-11 (49), 9-15 (70), 9-75 (72), 9-16 (73), 9-18 (79), 9-22 (96), 9-22 (96), 9-23 (100), 9-25 (107), 9-26 (108), 9-27 (113), 9-31 (126), 9-37 (147), 9-38 (152), 9-40 (159), 9-40 (161), 9-63 (248), 11-81 (225), 11-86 (232), 11-85 (235), 11-36 (236), 11-99 (266), 11-99 (266), 11-100 (267), 11-101 (271), 11-107 (286), 11-115 (312), 11-116 (313), 11-117 (320), 12-34 (39), 12-34 (41), 12-36 (42), 12-36 (44), 12-36 (45), 12-44 (75), 12-44 (76), 12-45 (81), 12-46 (87), 12-47 (93), 12-49 (102), 12-49 (103), 12-52 (111), 12-54 (123), 12-58 (134)

FTL-EDA .                          2-11 (54), 12-58 (132)

F2FL .                          2-14 (118), 3-19 (74), 9-15 (72), 10-10 (26), 10-18 (55)

free (b) .                          7-18 (58), 6-14 (36), 6-28 (74), 7-17 (56), 10-14 (41), 10-24 (70), 11-32 (92), 11-48 (132)

FREE .                          2-13 (112)

free-area (gen-1, gen-2) .                          7-19 (60), 7-17 (56)
ABSTRACTSYNTAXANDINTERPRETATIONOFPL/I

30April1969

free-based(gen)..................7-18(59),7-17(56)
free-blockend..................6-11(32),6-11(31)
free-buffer-vr(p,o,vr,pa)........11-56(147),11-30(85),11-55(142),11-62(160),11-64(165)
free-taskend..................5-8(27),5-7(23)

G CHAR..................5-7(25),11-24(69),11-27(79),11-29(84),11-30(85),11-30(87),11-31(89),11-33(94),11-34(96),11-61(113),11-46(126),11-47(129),11-50(134),11-50(135),11-51(139),11-51(140),11-55(142),11-59(155),11-59(156),11-62(160),11-64(164),11-122(329),11-123(332),12-68(172),12-68(173),12-70(177),12-70(178),12-72(183),12-73(184),12-73(186),12-73(187)

G CHAR ..................5-7(25),6-14(135),6-22(59),6-22(60),6-27(71)

G CHAR ..................5-11(33),5-10(32),5-11(33),7-15(50),12-67(170)

G CHAR ..................5-11(33),6-49(136),6-49(135),6-49(140),11-112(304),12-39(62),12-40(63)
gen-aggr-match(eva,aggr)........6-22(60),6-14(135),6-22(59),6-22(60),6-27(71)
gen-aggr-match-1(aggr-1,aggr-2,sg-cl)........6-21(54),6-19(48),6-21(54)
gen-base(g)..................11-85(239),11-84(233)
gen-da(gen)..................7-17(53),7-8(20),7-16(52),7-19(60),8-9(23),8-33(104),11-59(156),12-42(70),12-63(155),12-68(172),12-68(173),12-70(177),12-70(178),12-72(183),12-73(184),12-73(186),12-73(187)
gen-da-match(da-1,da-2)........6-22(61),6-22(60)
gen-da-match-1(da-1,da-2,sg-cl)........6-21(55),6-21(54)
gen-list(gen)..................5-11(33),5-10(32),5-11(33),7-15(50),12-67(170)
gen-op(gen)..................8-22(59),5-3(7),6-14(36),6-24(122),6-49(135),6-49(136),8-15(37),8-22(58),10-24(70),11-32(92),11-79(221),12-12(30),12-61(145),12-61(149),12-62(149),12-62(149),12-64(159),12-64(160),12-70(175),12-70(177),12-70(178),12-73(184),12-73(187),12-74(189)
gen-op-1(gen)..................6-49(136),6-49(135),6-49(140),11-112(304),12-39(62),12-40(63)
generic-aggr-match(aggr,g-aggr)........6-33(89),6-33(88),6-33(89)
generic-aggr-match-1(aggr,g-aggr)........6-33(90),6-33(89),6-34(90)
generic-arithm-match(da,g-da)........6-34(92),6-34(91)
generic-atrr-match(attr,descr)........6-33(88),6-33(87)
generic-da-match(da,g-da)........6-34(92),6-34(91)
generic-dens-match(aggr,g-dens)........6-35(95),6-33(89),6-34(90),6-35(95)
generic-match(attr-list,descr)........6-33(87),6-33(86)
generic-pic-match(da,g-da)........6-35(94),6-34(91)
generic-sel(gml,attr-list)........6-31(72),6-6(18),6-7(19),6-13(34),8-17(48),8-20(55),8-25(71),13-8(12)
gen-select-1(gml,attr-list)........6-33(86),6-31(77),6-33(86)
ABSTRACT SYNTAX AND INTERPRETATION OF PL/I

generic-string-match(da,g-da) ....... 5-34 (93), 6-34 (91)
GET .... 2-15 (133), 2-16 (134), 11-50 (134), 11-51 (137), 11-74 (209), 11-76 (214), 11-77 (215),
11-82 (226), 11-83 (229), 11-95 (255), 11-96 (256), 11-121 (327)
get-column(tr,el) ......... 11-114 (310), 11-82 (227), 11-114 (310)
get-control-format(tr,efo) ....... 11-82 (227), 11-82 (226)
get-data-field(tr) ............. 11-80 (228), 11-77 (215)
get-data-lp(tr,ref-list) ....... 11-107 (287), 11-74 (209), 11-107 (288), 11-108 (291)
get-data-sp(tr,id1,n,const1) ........ 11-109 (295), 11-108 (291)
get-edit(tr,efo) ............... 11-115 (311), 11-80 (224)
get-file(tr,arg) ............... 11-113 (306), 11-112 (304), 11-119 (309), 11-114 (310)
get-list(tr) ................. 11-103 (272), 11-80 (224)
get-skip(tr,el) ............... 11-114 (309), 11-82 (227), 11-114 (309), 11-116 (310)
get-space(tr,v) ................ 11-119 (308), 11-82 (227)
get-string(tr,v1,arg) .......... 11-112 (305), 11-112 (304)
get-val-list(tr,arg) .......... 11-111 (299), 11-103 (272), 11-103 (273), 11-104 (275), 11-109 (276),
11-107 (287), 11-107 (288), 11-108 (289), 11-108 (290), 11-103 (295), 11-110 (286),
11-110 (297), 11-110 (298), 11-114 (308), 11-115 (311)
get-1(tr,v1,el,arg) ............ 11-112 (302), 11-111 (299), 11-112 (302)
get-2(tr,arg) .................. 11-112 (302), 11-112 (302)
GOTO .................. 2-12 (97)
goto-1(den,block-act,cbif) ...... 6-44 (171), 6-43 (170), 6-44 (121), 13-3 (6), 13-15 (38)
goto-2(ind1) .................. 6-45 (123), 6-44 (121), 6-45 (123)
goto-3(ind1) .................. 5-45 (125), 6-17 (66), 6-45 (123), 6-45 (125)
goto-4(index) .................. 6-45 (126), 6-45 (125)
G2 .................. 2-18 (155), 9-3 (6), 9-10 (45), 9-17 (77), 9-18 (80), 9-18 (81)
n-CHAR .................. 9-9 (8)
head(list) .................. 1-8 (22)
i-CHAR ........ 2-9 (70), 9-4 (8), 9-30 (124), 9-46 (182), 9-46 (184), 9-56 (219), 11-17 (171), 11-67 (172),
11-72 (206), 11-106 (284)
id-length ................. 11-29 (83), 12-61 (174), 12-61 (174)
id-ref(idr) ................. 6-40 (171), 6-40 (171)
id1-string(id1) ............ 11-39 (263), 11-39 (263)
IF .......................... 2-12 (36)
ignore(mp,dsp,n) .......... 11-13 (64), 11-45 (123), 11-65 (167)
ignore-transmission(et,add) .... 11-55 (167), 11-54 (141)
imag(z) .................. 1-14 (28)
ABSTRACT SYNTAX AND INTERPRETATION OF PL/I

IMAG .................................................. 12-67(170), 12-68(172), 12-74(189)
IMPL .................................................. 11-24(67), 11-26(76), 11-27(78)
impl-fas(fas) ...................................... 9-9(21), 4-6(13)
impl-fas-1(fa) .................................... 4-9(22), 4-9(21)
incorporate(text) ................................. 6-50(142), 6-56(144)
INCORPORATE ....................................... 2-12(101)
indx(str,conf) ....................................... 12-38(55), 12-38(54), 12-42(69), 12-43(74)
infix-bit(v-1,v-2,opr) ...................... 9-19(85), 9-13(66)
infix-num(eda-1,eda-2,v-1,v-2,opr) ...... 9-15(69), 9-14(67), 9-15(70)
infix-op(op-1,op-2,opr) .................. 9-13(65), 9-13(64), 12-39(59), 12-44(75), 12-58(131), 12-59(137)
infix-opr(expr) .................................... 8-19(51), 8-19(50)
infix-val(eda-1,eda-2,v-1,v-2,opr) .... 9-13(66), 9-13(65)
init-control(tr,type,v) ................. 11-51(190), 11-50(134)
init-format-ci(fol) ......................... 11-75(210), 11-74(209)
init-string(tr,g) ................................. 11-51(139), 11-48(132)
initial-assign(genl,init,area) .......... 7-16(52), 7-15(50), 7-16(52)
initial-state(stg,es,time,date,ty,ev,t,call,gen) .......... 4-1(1)
initialize(gen,aggr) ....................... 7-15(49), 4-9(24), 6-8(22), 7-4(7), 7-15(49), 11-58(154)
initialize-refer(gen,eva,aggr) ......... 7-9(22), 7-9(21)
initialize-1(gen,init,area) ............. 7-15(50), 7-15(49)
INP .............................................. 2-3(19), 4-9(22), 11-4(8), 11-8(25), 11-22(61), 11-25(72), 11-25(74), 11-25(75), 11-31(89), 11-33(93), 11-33(94), 11-44(120)
insert(list, const) ......................... 8-6(12), 8-6(10), 8-6(12)
insert-eds(eda,aggr) ................. 12-54(111), 12-5(9), 12-5(11), 12-10(25)
insert-isubs(ref,list) ......................... 8-31(97), 8-31(95)
insert-list(list,cond,opt) .............. 10-6(12), 10-6(11), 10-9(24), 10-9(25)
insert-1(list-1,list-2,cond,opt) ...... 10-7(13), 10-7(13)
install-arg(b,type,n,attr) .............. 6-27(71), 6-26(70), 6-27(71)
install-arg-list(argl,para-list,dp) .... 6-26(70), 6-16(44)
install-builtin-arg(arg,da) .......... 12-12(30), 12-11(29)
install-gen(b,gen) ......................... 6-22(58), 6-21(57), 6-27(71)
install-info(gen,cond) ............... 10-15(42), 10-14(41)
INT .............................................. 2-3(13), 2-3(16), 2-3(18), 4-6(10), 6-8(22)
int-acc-1(cdt,t) ......................... 10-16(47), 10-16(46)

29 APPENDIX
ABSTRACT SYNTAX AND INTERPRETATION OF PL/I

int-acc-2(t) .................................................. -10-16(9), 10-16(47)
int-acc-3(cdl,t) ............................................... -10-17(59), 10-16(47), 10-16(49)
int-access-st(t) ............................................... -10-16(61), 6-41(118)
int-allocate(t,i) ............................................... 7-2(5), 7-2(2)
int-allocate-list(t) .......................................... 7-2(21), 7-2(1), 7-3(5)
int-allocate-st(t) ............................................ 7-2(11), 6-41(118)
int-allocate-1(t) ............................................... 7-2(17), 7-3(5)
int-assign(lp,expr,bname) .................................. 8-3(3), 8-3(1), 8-5(7)
int-assign-st(t) ............................................... 8-3(31), 6-41(118)
intrin(cond) ..................................................... -10-14(39), 10-14(39)
int-block(t) ..................................................... 6-2(11), 6-41(118), 13-14(29), 13-15(32)
int-block-1(t,e) ................................................ 6-2(3), 6-2(1)
int-body(t,id,argl,e) ......................................... 5-16(49), 6-16(43)
int-call(n,expr-list,descr,b,ref,pa-opt) .................. 6-14(37), 6-13(34), 6-13(35)
int-call-st(t) ................................................... 6-13(34), 3-9(8), 6-41(118), 7-15(50)
int-call?-1(den,argl,b,pa-opt) ............................... 6-15(49), 6-14(37)
int-cond(ca,cbl) ............................................... -10-20(80), 10-19(56), 13-15(30)
int-cont-do(t,tr) ................................................ 6-47(139), 6-46(127), 11-76(214)
int-delay-st(t) .................................................. 5-15(49), 5-41(118)
int-disable-st(t) ............................................... 10-8(22), 6-41(118)
int-disable-1(cdl) ............................................. 10-8(23), 5-7(23), 10-8(22), 10-8(23)
int-display-st(t) ............................................... 11-40(108), 6-41(118)
int-do-list(truth,list,tr) .................................... 6-49(139), 6-46(128), 5-46(134)
int-do-spec(init,by,lo,while,ger,ref,list,tr) ............ 6-47(132), 6-47(131)
int-ent-read-st(et,add) ...................................... 11-60(159), 11-54(141)
int-enable-st(t) ................................................. 10-9(7), 6-41(118)
int-enable-1(enable-list) .................................... 10-9(8), 10-4(7), 10-4(8)
int-enable-2(tr,enable,cdl) ................................ 10-5(9), 10-4(8), 10-5(9)
int-exit-st ....................................................... 5-6(16), 6-41(118)
int-fetch-st(t) .................................................. 6-36(29), 6-41(118)
int-file-option(op) ............................................ 11-23(66), 11-23(64)
int-free(t,i) ...................................................... 7-17(58), 7-17(55)
int-free-st(t) .................................................... 7-17(55), 6-41(118)
int-get-put-file(et) .......................................... 11-50(134), 11-48(132)
ABSTRACT SYNTAX AND INTERPRETATION OF PL/I

30 April 1969

int-get-pat-st(t) .................................................. 11-48(132), 6-41(110)
int-goto-st(t) ...................................................... 6-43(110), 6-47(118)
int-group(t) .......................................................... 6-46(127), 6-41(118)
int-if-st(t) ............................................................ 6-40(113), 6-41(118)
int-impl-open(t) ...................................................... 11-23(64), 11-37(103)
int-impl-open-1(t,n) ................................................. 11-24(67), 11-23(64)
int-incorporate-st(t) ................................................. 6-50(141), 6-41(118)
int-ic-st-1(t,n) ....................................................... 11-49(109), 11-40(108)
int-ic-st-2(ref-list,et,n) ......................................... 11-41(112), 11-30(85), 11-40(109)
int-extend-st ........................................................... 6-39(111), 5-12(38), 6-39(109), 6-45(125), 7-3(5), 11-96(256)
int-on-st(t) ............................................................. 10-19(31), 6-41(118)
int-open-close-st(t) ............................................... 11-18(49), 6-41(118)
int-open-close-1(pt) ................................................ 11-19(49), 11-18(48), 11-20(55)
int-open-close-2(pt,et) .............................................. 11-20(55), 11-19(49)
int-open-close-3(pt,et) ............................................. 11-21(56), 11-20(55)
int-options-1(t,pt) .................................................. 11-37(101), 11-37(102), 11-40(109), 11-48(132)
int-options-2(t,pt,et) ................................................ 11-37(102), 11-37(101)
int-options-3(t,pt,et) ................................................ 11-37(103), 11-37(102)
int-proc-st-list(st-list,ind1) .................................. 6-17(46), 6-16(44)
int-program(t,call,gen) ............................................ 4-3(4), 4-1(1)
int-proc-st(t) .......................................................... 6-41(118), 6-40(116)
int-rec-lo-st(t) ....................................................... 11-49(106), 6-41(118)
int-release-st(t) ..................................................... 6-37(109), 6-41(118)
int-return-st(t) ........................................................ 6-28(73), 6-16(44), 6-41(118)
int-revert-st(t) ........................................................ 10-12(36), 6-41(118)
int-scalar-assign(lp,expr) ....................................... 8-7(13), 8-3(3)
int-signal-st(t) ........................................................ 19-20(62), 6-41(118)
int-snap(cond) .......................................................... 10-19(57), 10-19(56)
int-st(st) ............................................................... 6-40(116), 6-39(109), 10-20(60)
int-st-list(t) .......................................................... 6-38(107), 6-2(3), 5-41(118), 6-49(139)
int-st-pat-open ........................................................ 11-26(76), 11-121(327), 11-123(331)
int-stop-st ............................................................. 5-6(19), 6-41(118)
int-task(den,argl) ................................................... 5-5(13), 5-4(10)
int-unlock-1(st) ...................................................... 11-49(126), 5-14(46), 11-41(112), 11-47(131)
INT-SIT-ET (t) ............................................. 5-10 (29), 6-41 (118)
INT-SIT-ET-1 (ref-list, expr, w) ......................... 6-48 (129), 6-60 (129)
INT-SE (t) .................................................. 6-48 (129), 6-60 (129)
INTG .......................................................... 12-4 (5), 12-6 (13), 12-6 (15), 12-4 (5), 12-6 (15)
INTG-EDA (n) .............................................. 9-11 (47), 9-11 (49), 9-11 (51), 9-39 (156), 11-17 (319)
INTG-EDA ..................................................... 9-11 (47), 9-11 (49), 9-39 (156)
INTG-EDA ..................................................... 9-11 (47), 9-11 (49), 9-39 (156)
INTG-EDA ..................................................... 9-11 (47), 9-11 (49), 9-39 (156)
INTG-EDA ..................................................... 9-11 (47), 9-11 (49), 9-39 (156)
INTG-op (i) .................................................. 6-10 (123), 6-10 (127)
INTG-val (expr) ............................................. 6-10 (123), 6-10 (127)
INTG-val-1 (expr) ......................................... 6-10 (123), 6-10 (127)
INTG-string (intgl) ........................................ 12-8 (245), 12-10 (245)
INTS-Set-transmission (et, add, gen) .................... 12-8 (119), 12-10 (119), 12-10 (119)
INvert (I) .................................................... 12-8 (119), 12-10 (119)
IO-cond-ltrate (en) ........................................ 5-13 (44), 5-13 (44)
IO-cond-ltrate-1 (cond-list, en) ........................... 5-13 (44), 5-13 (44), 5-13 (44)
IO-cond-ltrate-2 (cond-list) ................................ 11-43 (119), 11-43 (119), 11-43 (119)
IO-cond-return (cond-list, et) ............................. 11-44 (120), 11-44 (120), 11-44 (120)
IO-data-op (t) .............................................. 12-105 (282), 12-105 (282), 12-105 (282)
IO-idl (io-ref) .............................................. 12-109 (292), 12-109 (292), 12-109 (292)
IO-on-actions (blck-act, cbif) .............................. 6-44 (122), 6-44 (122)
IO-sl (io-ref) ............................................... 11-103 (293), 11-103 (293)
IO-transmission (et, add) .................................. 11-43 (116), 11-43 (116), 11-43 (116)
IS-BEL ....................................................... 12-5 (12), 12-7 (19), 12-10 (25)
is-0 (x) ..................................................... 1-6 (3)
is-<> (list) ................................................... 1-8 (19)
is-abn-ret .................................................... 3-22 (84), 3-22 (84), 3-22 (84)
is-abn-return (j2, tn, cp) ................................ 13-15 (34), 13-15 (34)
is-access-st ............................................... 2-10 (24), 2-11 (89), 2-10 (24), 2-10 (24), 2-10 (24), 2-10 (24)
is-active (gen, pa) ......................................... 5-5 (12), 5-5 (12), 5-5 (12), 5-5 (12), 5-5 (12), 5-5 (12)
is-active-ba (ba-1, ba, d) ................................ 6-15 (42), 6-15 (42), 6-15 (42)
is-active-event (gen, pa) .................................. 5-5 (12), 5-5 (12), 5-5 (12), 5-5 (12), 5-5 (12), 5-5 (12)
is-active-pa (tn, e) ........................................ 3-30 (114), 3-30 (114), 3-30 (114)
is-ag ......................................................... 3-14 (28), 3-3 (3)

APPENDIX 27
ABSTRACT SYNTAX AND INTERPRETATION OF PL/I

is-attention
is-attr
is-attr-cond
is-attr-ev
is-attr-occ
is-attr-te
is-b-data
is-ba
is-based
is-bin-pi
is-bin-spec
is-bintg
is-bintg-op
is-bit-aggr
is-bit-opr
is-bit-val
is-block
is-body
is-body-part
is-buf
is-builtin
is-builtin-den
is-c
is-c-exponent
is-call-st
is-chif
is-char-aggr
is-char-ed-spec
is-char-num-data

APPENDIX 29
is-common-refer-opti~ns (aggr,aggr-1)  .......... 13-10(17), 13-9(15)
is-common-cl(sl,ref,ref-1,ref-2)  .......... 13-8(13), 13-8(12), 13-8(13), 13-13(27)
is-common-var (ref,ref,ref-1,ref-2)  .......... 13-9(15), 13-9(14)
is-commoning-step (cp,ref)  .......... 13-2(41)
is-commutative-infix-opr (opr)  .......... 13-5(10), 13-5(9)
is-comp-opr  .......... 2-10(45), 9-13(66), 9-17(77), 9-21(90), 9-23(100)
is-comp-step (f-1,f-2)  .......... 2-30(111), 3-30(110), 13-2(2)
is-compl(i,gen1,gen2,et,te)  .......... 5-11(36), 5-11(35), 5-12(40), 5-12(41)
is-compl-correct (cl,pred)  .......... 11-71(193), 11-71(200), 11-71(202), 11-104(278), 11-108(291), 11-116(313), 11-118(321)
is-cond  .......... 2-14(117), 2-14(114), 2-14(115), 2-14(116), 3-21(76)
is-cond-act  .......... 3-21(82), 3-21(81)
is-cond-act-part  .......... 3-21(83), 3-21(79)
is-cond-bif-part  .......... 3-21(84), 3-21(79)
is-cond-part  .......... 2-2(9), 2-2(5), 2-4(23), 2-11(88), 2-11(91)
is-cond-1 (aggr,gen1,eq1)  .......... 8-35(110), 8-34(109), 8-35(110)
is-cond-2 (ref,at)  .......... 8-35(111), 8-34(109)
is-cond-3 (aggr)  .......... 8-36(114), 8-34(109), 8-36(114)
is-connected (ref,at)  .......... 8-34(109), 8-33(103), 8-35(111), 11-40(106), 12-62(150)
is-cond  .......... 2-12(161), 2-16(153), 5-5(8), 8-16(40), 8-19(50), 8-23(52), 11-109(295), 13-4(8)
is-contr-group  .......... 2-17(94), 2-11(92), 6-46(127)
is-contr-item  .......... 2-16(142), 2-16(141), 11-76(214)
is-control-format  .......... 2-10(83), 2-10(79)
is-control-format-type  .......... 2-10(86), 2-10(85)
is-corr-on-unit (st)  .......... 10-20(81), 10-20(60)
is-correct-drift (eda,sf)  .......... 9-5(1203), 9-49(200)
is-correct-edu (eda)  .......... 9-5(111), 7-5(9), 8-20(52)
is-correct-env-attr (et)  .......... 9-5(129), 4-6(13)
is-correct-eval (eva)  .......... 7-5(9), 7-5(8), 7-5(9), 7-5(10), 7-8(20), 7-9(23), 7-10(24), 7-11(25), 11-59(155), 11-59(156)
is-correct-pic (eda)  .......... 9-48(193), 9-43(165), 9-45(172), 9-61(240), 9-63(246), 9-64(250), 11-85(235)
is-correct-pic-1 (eda)  .......... 9-48(193), 9-45(172), 9-48(193)
is-correct-sh-pc (eda,sf,m,n)  .......... 9-50(202), 9-50(207)
**Abstract Syntax and Interpretation of PL/I**

*30 April 1969*

**is-correct-static(s,t)**

- 4-7(14), 4-6(13)

**is-correct-sterling-subf(eda,sf)**

- 9-50(201), 9-49(200)

**is-correct-subf(eda,sf)**

- 9-49(200), 9-48(195)

**is-corresp(aggr-1,aggr-2)**

- 6-20(51), 6-19(48), 6-27(71), 8-31(95), 8-32(100), 8-35(111)

**is-corresp-conn(aggr-1,aggr-2,s1,eql)**

- 8-35(112), 8-35(111), 8-35(112)

**is-corresp-1(aggr-1,aggr-2)**

- 8-4(5), 8-4(4)

**is-corresponding(i1,i2,tn,s,cp)**

- 13-17(39), 13-16(38), 13-17(44)

**is-cplx-forma**

- 11-4(41), 3-25(95), 3-26(98), 11-22(62), 11-27(79)

**is-cs**

- 1-23(42)

**is-ctl-alloc(b,p,k,tn,cp)**

- 13-17(45), 13-17(40), 13-17(41), 13-17(42)

**is-ctl-ref(ref,at)**

- 6-19(50), 6-19(48), 6-35(97), 7-17(56), 12-64(161)

**is-d**

- 3-7(14), 3-3(3), 3-7(14)

**is-data-dir-ref(ref,at)**

- 2-10(92), 2-10(89)

**is-data-format**

- 3-21(79), 3-3(3), 3-7(14)

**is-data-spec**

- 11-4(41), 3-25(95), 3-26(98), 11-22(62), 11-27(79)

**is-data-dir**

- 11-51(138), 11-51(137), 11-79(221)

**is-data-ref(ref,at)**

- 11-76(212), 11-75(211), 11-76(213)

**is-data-format**

- 2-10(92), 2-10(79)

**is-dec-dig-spec(eda,sf,i)**

- 2-9(69), 2-9(68), 9-45(173), 9-45(175), 9-48(195), 11-85(235), 11-87(238)

**is-dec-pic**

- 2-8(69), 2-9(68), 9-43(165), 9-45(173), 9-45(175), 9-48(195), 11-85(235), 11-87(238)

**is-dec-spec**

- 2-9(70), 2-9(72)

**is-decipherable(mp,ds)**

- 11-5(14), 11-5(16), 11-6(18), 11-6(19), 11-6(20), 11-7(22), 11-7(23), 11-8(24), 11-8(26), 11-9(27), 11-9(28), 11-10(36), 11-10(36), 11-11(40), 11-11(41), 11-12(43)

**is-decipherable-0(mp,ds)**

- 11-6(20), 11-8(26), 11-9(27), 11-27(79)

**is-decipherable-1(mp,ds)**

- 11-13(43), 11-12(42), 11-13(44), 11-14(45), 11-14(46), 11-15(47), 11-30(87), 11-31(89), 11-33(94), 11-85(123)

**is-deciphered(mp,ids)**

- 11-6(19), 11-6(20), 11-7(21), 11-9(30), 11-9(31), 11-9(32), 11-9(33), 11-10(36), 11-10(36), 11-11(40), 11-11(41), 11-31(45)

**is-decl**

- 2-2(12), 2-2(11), 3-11(21), 8-4(5), 6-7(15)

**is-decl-part**

- 2-2(11), 2-2(3), 2-2(5), 2-11(91)

**is-defined**

- 2-3(14), 2-2(12), 6-6(15), 5-6(18), 6-8(22), 8-9(22), 8-25(71), 8-26(76), 8-29(88), 8-29(90), 8-31(95), 13-9(15)
ABSTRACT SYNTAX AND INTERPRETATION OF PL/I
ABSTRACT SYNTAX AND INTERPRETATION OF PL/I

30 April 1969

is-e-open-close

is-ea(ea)

is-eatn-conv

is-ed-spec

is-ed

is-edit-dir

is-editdir-exponent

is-ei

is-el

is-en

is-enable

is-enable-st

is-end

is-entry

is-entry-const

is-entry-da

is-entry-den

is-entry-part

is-entry-point

is-env-attr

is-env-step

is-eq-struct

is-equal-ptr

is-es

is-et

is-et-entry

is-ev

is-eva

is-event-val

APPENDIX
ABSTRACT SYNTAX AND INTERPRETATION OF PL/I

is-exit-st

is-expr

is-ext-decl

is-extest

is-extra-lingual-char

is-f-cond

is-f-const

is-f-data

is-f-end-cond

is-fd

is-fd-status

is-fetch-st

is-file-attr

is-file-const

is-file-get-st

is-file-put-st

is-finished

is-fix-match

is-fix-pic

is-fixed-string

is-float-generic-built-in

is-ft-match

is-ft-pic

is-fn-cond

is-format

is-format-const

is-format-den

APPENDIX 35
ABSTRACT SYNTAX AND INTERPRETATION OF PL/I

is-format-item

is-format-iter

is-format-match(fu-el,type)

is-format-1

is-free

is-free-sel-1(pt,v)

is-free-sel

is-free-space(p,vr,type)

is-free-st

is-fu

is-fu-el

is-fu-file(fu-el)

is-fu-file-1

is-fu-file-2(fu-el,csa)

is-fu-status

is-fu-string

is-g-gen

is-garbage

is-generic

is-generic-aggr

is-generic-arithmetic

is-generic-array

is-generic-data

is-generic-dependency(ref-1,ref-2,ref-3,at)

is-generic-description

is-generic-member

is-generic-scalar

is-generic-string

is-generic-structure

is-generic-successor

APPENDIX

36
ABSTRACT SYNTAX AND INTERPRETATION OF PL/I

is-get-st

2-15(132), 2-11(89), 6-41(118)

is-goto-st

2-12(97), 2-11(89), 6-41(118)

is-group

2-11(92), 2-11(89), 6-41(118), 10-20(61)

is-id

1-6(43)

is-id-ref

2-24(93), 2-2(7), 2-5(39), 2-11(88), 2-14(123), 6-40(117), 10-7(15), 10-7(16)

is-ident

3-18(69), 3-18(66), 3-18(68), 3-19(72), 10-7(18)

is-idm

11-21(11), 11-5(16)

is-if-st

2-12(96), 2-11(89), 6-39(110), 6-40(113), 6-41(118), 10-20(61)

is-ignore-read-st

2-17(143), 11-25(74)

is-inc-arths

9-37(147), 9-37(146), 9-37(149)

is-inc-edc

9-37(148), 9-29(119)

is-inc-string

9-37(148), 9-37(146), 9-37(149), 9-35(154)

is-incorporate-st

2-12(101), 2-11(89), 6-41(118), 6-50(144)

is-incorporate-text

2-12(102), 2-12(101)

is-indep(0-1, p-2)

3-16(43), 3-16(44), 3-16(45), 3-16(46), 3-16(47), 3-16(51), 3-17(57),
5-4(10), 5-5(12), 7-12(30), 7-12(35), 7-13(43), 9-19(84), 11-56(147)

is-index

2-4(22), 2-2(7), 2-3(21), 3-9(17), 3-12(25), 6-45(123)

is-infix-expr

2-18(152), 2-18(153), 8-11(26), 9-25(105)

is-infix-expr-1

8-16(41), 8-5(8), 8-16(40), 8-19(50), 8-19(51), 8-23(62), 8-26(76), 13-4(6)

is-infix-ops

2-18(155), 2-18(154), 8-16(41), 3-10(43), 12-44(75), 12-44(76), 12-59(137)

is-info

3-18(79), 3-15(67), 3-16(68)

is-init

2-5(41), 2-5(33), 2-21(179)

is-init-elem

2-6(42), 2-5(41), 2-6(43), 2-8(65), 4-8(18), 7-15(50)

is-init-iter

2-6(43), 2-6(42), 4-8(18), 7-16(52)

is-intg(eda)

9-11(51), 6-10(30), 9-10(41), 9-14(68), 9-21(89), 9-25(105)

is-intg-const

6-10(40), 6-10(28), 6-10(29), 6-21(55), 12-6(13)

is-intg-op(x)

9-10(41)

is-intg-ral

9-3(5), 1-7(11), 1-7(13), 1-7(14), 1-13(29), 1-13(30), 1-13(31), 1-23(42), 1-23(43), 2-9(22), 2-5(35), 2-6(44), 2-6(46), 2-6(51), 2-7(52), 2-7(59), 2-8(63), 2-9(71), 2-9(73), 2-19(183), 2-19(168), 2-20(163), 2-21(176), 3-5(11), 3-5(11), 3-10(19), 3-22(63), 3-22(85), 3-25(94), 3-26(98), 3-26(99), 3-31(118), 4-2(2), 4-8(17), 5-11(34), 6-10(30), 6-21(55), 6-22(61), 6-24(63), 6-25(66), 6-26(68), 6-31(78), 6-37(102), 6-39(110), 6-39(111), 7-5(9), 7-7(16), 8-5(8), 8-6(10), 8-17(46), 9-2(1), 9-4(12), 9-4(13), 9-5(17), 9-24(104), 9-29(102), 9-25(105), 9-25(107), 9-26(108), 9-32(123), 9-32(129), 9-37(147), 9-37(148), 9-43(165), 10-24(76), 10-24(71), 10-24(72), 10-27(81), 11-2(1), 11-4(11), 11-9(30), 11-13(44), 11-20(53), 11-22(59), 11-28(81), 11-28(82), 11-29(83), 11-35(98), 11-36(99), 11-48(120), 11-49(125), 11-50(134), 11-55(144), 11-56(145), 11-57(149), 11-77(216), 11-80(222), 11-82(226), 11-86(236), 11-90(246), 11-94(254), 11-96(256), 11-98(263), 11-111(279), 11-120(283), 11-122(329), 11-122(330), 12-6(15), 12-10(25), 12-14(33), 12-15(34), 12-15(35), 12-16(36), 12-16(37), 12-17(38), 12-36(47), 12-37(52), 12-62(72), 12-71(180)

APPENDIX 37
ABSTRACT

SYNTAX

INTERPRETATION

PL/I

30 April 1969

is-into-read-st

is-io-arithm-const

is-io-arithm-data

is-io-basic-ref

is-io-binary-fixed

is-io-binary-fixed-part

is-io-binary-float

is-io-binary-integer

is-io-bit-char

is-io-bit-string

is-io-char-string

is-io-cond

is-io-cplx-expr

is-io-data

is-io-decimal-fixed

is-io-decimal-float

is-io-del-subs

is-io-ev

is-io-exponent

is-io-identifier

is-io-integer

is-io-prop-arithm-data(dt)

is-io-prop-data(dt)

is-io-prop-rp

is-io-real-const

is-io-sign

is-io-string-apost

is-io-string-char(x)

is-io-subs(x)

is-io-subs-part

is-io-te

APPENDIX
is-isub: 2-19 (155), 2-18 (153), 8-5 (8), 8-16 (40), 8-19 (50), 8-23 (62), 8-29 (90), 8-31 (97), 13-4 (8)
is-isub-def (attr): 8-29 (90), 8-29 (87), 8-35 (111)
is-isub-val: 8-17 (46), 8-16 (40)
is-item: 2-16 (141), 2-16 (139), 2-16 (140), 2-16 (142), 6-49 (139), 11-76 (212)
is-krec: 11-3 (5), 11-3 (4), 11-5 (12), 11-14 (40)
is-label-const: 2-3 (21), 2-2 (12), 6-8 (22), 7-16 (51), 8-6 (10), 8-20 (55), 8-25 (71), 10-2 (3),
10-22 (67), 11-79 (221), 13-8 (14)
is-label-da: 2-5 (39), 2-5 (34), 2-21 (186), 6-10 (26)
is-label-den: 2-12 (25), 3-12 (23), 6-43 (120)
is-last-vol(mp,ds,volno): 11-25 (98), 11-34 (96)
is-left-correct(cl,pred): 11-71 (203), 11-71 (201)
is-letter: 9-3 (8), 9-3 (7), 9-59 (235), 11-59 (191)
is-level-1(ref): 7-18 (57), 7-17 (56)
is-list(list): 1-9 (18)
is-list-data-dir: 2-16 (139), 2-16 (138)
is-list-1(list): 1-2 (29)
is-locate-st: 2-17 (139), 2-11 (89), 6-41 (118), 11-23 (65), 11-25 (74), 11-25 (75)
is-locked-foreign(fu-el,tn,ident): 11-47 (128), 11-47 (129), 11-47 (130)
is-locked-own(fu-el,tn,ident): 11-47 (127), 11-46 (126)
is-m: 3-28 (197), 3-11 (1)
is-mod-alloc-freed-1(i1,i2,j1,j2,p,k,k1,k2,tn,cp): 13-17 (49), 13-15 (33)
is-mod-alloc-freed-2(i2,j1,j2,p,k,k1,k2,tn,cp): 13-17 (41), 13-15 (33)
is-mod-alloc-freed-3(i1,i2,j1,j2,p,k,k1,k2,tn,cp): 13-17 (42), 10-15 (33)
is-modified(p,i,cp): 13-17 (43), 13-17 (40), 13-17 (42), 13-17 (44)
is-modified-gen(b,i,tn,cp): 13-18 (46), 13-17 (40), 13-17 (41), 13-17 (42)
is-modified-1(p,j,tn,cp): 13-17 (41), 13-17 (41)
is-mp(obj): 11-4 (31), 11-35 (98), 11-45 (123)
is-n: 1-6 (51)
is-named-ic-cond: 2-14 (129), 2-14 (117), 10-11 (31), 10-11 (32), 10-12 (33), 10-12 (36), 10-20 (62)
is-named-message: 3-28 (108), 3-28 (107), 3-31 (119)
is-ncl: 11-70 (198), 11-70 (196), 11-70 (196)
is-non-varying(aggr): 9-29 (39), 9-29 (88), 9-31 (95)
is-norm-subf(eda,sf,m): 9-63 (244), 9-62 (241)
is-not-guaranteed-reference(i1,i2,j1,j2,p,k,k1,k2,tn,cp): 13-15 (33), 13-14 (28)
is-prop-var :: 2-3(13), 2-2(12), 8-6(11), 8-9(22), 8-11(27), 8-25(71), 8-26(76), 8-29(88), 8-35(111), 10-2(3), 10-22(67), 11-76(212), 13-9(15), 13-9(15)
is-proper-commoning-step(cp, t-1) :: 13-3(3), 13-2(2)
is-proper-program(t) :: 2-1(1), 4-2(2)
is-proper-step(cp, t-1) :: 13-2(2), 13-2(1)
is-ps-gen :: 12-67(169), 3-22(83), 3-22(85), 3-26(99), 6-49(136), 8-8(15), 8-8(16), 10-24(70), 10-24(71), 11-76(212), 11-80(223), 11-85(234), 12-62(149)
is-put-st :: 11-3(6), 11-3(4), 11-5(12), 11-14(46)
is-rec :: 11-3(4), 11-3(2)
is-rec-file(t, csa) :: 1-14(40), 1-14(41), 9-4(13), 9-8(28), 9-16(73), 9-16(74), 9-31(126), 9-33(132), 9-52(208), 12-57(124), 12-65(164)
is-rec-file-1(t, csa) :: 11-25(73), 11-25(73)
is-rec-io-st :: 11-25(73)
is-recursive-error(cp) :: 13-18(47)
is-reducible(da, i, opt) :: 13-10(19), 13-8(14), 13-10(18), 13-10(19), 13-12(25)
is-reducible-built-in(i) :: 13-12(26), 13-8(14)
is-ref :: 2-18(159), 2-3(14), 2-4(25), 2-5(40), 2-10(87), 2-11(94), 2-12(97), 2-12(98), 2-12(99), 2-12(103), 2-12(104), 2-12(105), 2-13(109), 2-13(113), 2-14(119), 2-14(120), 2-15(129), 2-15(131), 2-15(133), 2-16(136), 2-16(137), 2-16(142), 2-17(144), 2-17(145), 2-17(146), 2-17(147), 2-17(148), 2-17(149), 2-17(150), 2-18(151), 2-18(153), 2-19(167), 3-5(9), 3-21(77), 3-22(3), 4-4(8), 6-7(14), 6-35(98), 7-17(56), 8-4(4), 8-5(8), 8-9(21), 8-11(26), 8-17(48), 8-28(83), 10-2(3), 10-21(64), 10-22(65), 10-22(65), 10-22(67), 11-20(54), 11-25(74), 11-38(104), 11-40(108), 11-40(110), 11-59(156), 11-76(212), 11-76(214), 11-123(331), 12-65(165)
is-ref-1 :: 8-17(49), 8-16(40), 8-19(50), 8-23(62), 13-4(8)
is-refer :: 2-4(30), 2-4(29), 2-5(36), 2-5(37), 2-21(176), 7-8(19), 7-9(22), 8-15(37), 8-22(60), 13-10(17)
is-referenced(p, k, tn, cp) :: 13-16(35), 13-15(33), 13-16(38)
is-referenced-1(p, k, tn, cp) :: 13-16(38)
is-release-st :: 2-12(109), 2-11(89), 6-41(118)
is-remote-format

is-reorder-erroneous(cm)

is-reorder-section(l1, l2, tn, cp)

is-return-st(reply-1, reply-2)

is-revert-st(eda, i)

is-size(da, aggr)

is-scalar da(da, aggr)

is-sign-mag-pic(eda)

is-sign-pas(eda, sf, a)

is-sigti-spec(eda, sf, i)

is-sign-intg(eda, sf)

is-size(z)

is-size-cond(eda, v)

is-size-violation(pp, ds, el)

is-skipped-format(edfo)

is-spec-init-elem

ABSTRACT SYNTAX AND INTERPRETATION OF PL/I
is-st. 2-11(56), 2-2(5), 2-11(89), 2-11(91), 2-11(93), 2-11(94), 2-12(96), 2-14(114), 2-14(120), 3-21(82), 6-36(107), 6-39(110), 6-41(119), 6-49(139), 10-19(56), 10-20(61)

is-standard-fd-el

is-state

is-step(ε-1,ε-2)

is-sterling-dig-match(spec,n)

is-sterling-pic 2-9(71), 2-8(66), 9-43(155), 9-40(173), 9-45(176), 9-48(195), 11-87(238)

is-sterling-spec 2-9(72), 2-9(71), 9-46(179)

is-stop-inf 11-112(303), 11-95(255), 11-112(302), 11-113(307)

is-stop-st 2-13(108), 2-11(89), 6-41(118)

is-str-el 11-33(1), 11-3(2), 11-5(12), 11-94(253), 11-95(255)

is-str-file(t,csa) 11-25(72), 11-24(69)

is-str-size-cond(eda,v)

is-string 2-21(182), 2-21(180), 5-9(24), 6-10(26), 6-20(53), 6-21(55), 6-22(61), 6-25(67), 6-34(91), 6-47(132), 7-7(18), 8-14(36), 8-20(52), 8-24(65), 8-25(68), 8-32(102), 8-33(106), 8-34(107), 8-35(112), 8-35(114), 9-2(1), 9-7(27), 9-8(29), 9-8(30), 9-8(31), 9-17(77), 9-21(92), 9-33(134), 9-39(154), 9-39(157), 9-40(162), 9-41(163), 9-41(164), 9-39(105), 12-34(40), 12-42(76), 12-71(180)

is-string-const 2-19(163), 2-19(161)

is-string-format

is-string-get-st

is-string-put-st

is-string-type 2-16(139), 2-15(132), 11-48(132)

is-string-type 2-16(137), 2-16(135), 11-48(132)

is-str-type 9-21(92), 9-21(90), 9-30(122), 9-39(156), 9-40(160), 11-87(238), 11-88(239), 12-50(106), 12-51(108), 12-68(173), 12-74(189)

is-struct 2-21(172), 2-20(174), 4-7(16), 6-3(7), 6-7(21), 6-9(23), 6-9(25), 6-20(52), 6-21(54), 6-22(60), 6-24(63), 6-26(69), 6-32(63), 6-32(84), 6-34(90), 6-35(95), 7-5(9), 7-6(14), 7-7(18), 7-12(34), 7-12(35), 7-13(37), 7-13(38), 7-15(49), 8-3(3), 8-4(51), 8-5(7), 8-6(10), 8-13(31), 8-13(32), 8-14(33), 8-14(34), 8-14(36), 8-23(63), 8-24(66), 8-25(69), 8-25(72), 8-26(75), 8-29(84), 8-30(91), 8-30(92), 8-30(94), 8-32(102), 8-33(106), 8-34(107), 8-34(108), 8-35(110), 8-35(112), 8-36(114), 8-37(117), 8-37(118), 10-83(66), 11-77(215), 11-60(222), 12-5(11), 12-8(21), 12-9(23), 13-5(11), 13-10(12), 13-11(23), 13-12(24)

is-struct-eva 2-20(170), 2-20(168)


is-subscr-expr 2-19(169), 2-18(159), 8-17(44)

is-succ 2-21(178), 2-21(177)

is-succ-eva 2-20(171), 2-20(170)

is-task 3-3(3), 3-3(2)

is-task-te 3-5(8), 3-3(3), 3-5(7)

is-tl 3-5(11), 3-1(1)
is-time-step(time-1, time-2)

is-tr-var

is-trail-sign(eda, sf, i)

is-transient-input(et, mp, ds)

is-true-ace-num-comp(v-1, v-2, opr)

is-true-comp(eda-1, eda-2, v-1, v-2, opr, da)

is-true-num-comp(eda-1, eda-2, v-1, v-2, opr)

is-type-match(eda, v)

is-uniqu-key(data, key)

is-unlock-st

is-valid-date(d)

is-value

is-var

is-vr(vr)

is-wait-st

is-while-group

is-write-st

is-xdec-pic(eda)

is-xnum-pic

is-x'str(eda)

is-x'struning-pic(eda)

is-x'string-pic(base, v-1, v-2, opr)

is-x'str-Comp(eda-1, eda-2, v-1, v-2, opr, dn)

is-x'str-Comp(eda, v)

is-x'str-Comp(base, v-1, v-2, opr)

is-x'str-Comp(eda-1, eda-2, v-1, v-2, opr)

is-x'str-Comp(eda, v)

is-x'str-Comp(base, v-1, v-2, opr)

is-x'str-Comp(eda-1, eda-2, v-1, v-2, opr)

is-x'str-Comp(eda, v)

is-x'str-Comp(base, v-1, v-2, opr)

is-x'str-Comp(eda-1, eda-2, v-1, v-2, opr)

is-x'str-Comp(eda, v)

is-x'str-Comp(base, v-1, v-2, opr)

is-x'str-Comp(eda-1, eda-2, v-1, v-2, opr)

is-x'str-Comp(eda, v)

is-x'str-Comp(base, v-1, v-2, opr)

is-x'str-Comp(eda-1, eda-2, v-1, v-2, opr)

is-x'str-Comp(eda, v)

is-x'str-Comp(base, v-1, v-2, opr)

is-x'str-Comp(eda-1, eda-2, v-1, v-2, opr)

is-x'str-Comp(eda, v)

is-x'str-Comp(base, v-1, v-2, opr)
is-1-compl-pic(eda)  9-49(199), 9-48(195), 9-54(214), 9-63(245)
is-2-compl-pic(eda)  8-49(198), 9-48(195), 9-53(210), 9-54(214), 9-63(245)
isub-list(r1,k)  8-32(29), 8-31(95)
isub-op(isub)  8-20(33), 8-19(50)
isub-val(i,list)  8-31(39), 8-31(97)
itrate-assign(lp,expr,bname,eva,i)  8-5(7), 8-3(3), 8-5(7)
itrate-call-cond(r1,type)  10-22(65), 10-20(62), 10-21(64), 10-22(65)
itrate-control-format(tr,efo)  11-81(225), 11-79(221), 11-80(224), 11-81(225)
itrate-do(gen,by-to-ops,while,ref,list,tr)  6-48(134), 6-47(132), 6-49(140)
itrate-dummy-exp(expr,aggr,i)  6-54(54), 6-24(63), 6-24(64)
itrate-expansion(tr,expr,aggr,i)  11-77(216), 11-77(215), 11-77(216)
itrate-infix-op(op,opl,expr)  12-59(137), 12-39(60), 12-57(124), 12-57(126), 12-58(133), 12-59(135), 12-59(137)
itrate-spec-list(spec-list,gen,ref,list,tr)  6-47(131), 6-47(130), 6-47(131)
J-CHAR  9-4(8)
K-CHAR  9-4(8)
KEY  2-3(19), 2-14(121), 3-20(75), 3-26(98), 4-9(22), 11-4(8), 11-4(11), 11-5(12), 11-9(28), 11-9(30), 11-11(40), 11-12(42), 11-14(45), 11-14(46), 11-15(47), 11-22(62), 11-25(73), 11-25(74), 11-26(77), 11-29(84), 11-34(96), 11-45(123), 11-56(144), 11-56(145), 11-56(146), 11-61(159), 11-62(160), 11-64(164), 11-64(165)
L-CHAR  9-4(8)
lab sel(ct)  11-32(81), 11-31(89), 11-33(94)
LABEL  2-5(39), 2-6(50), 2-7(58), 2-20(173), 6-60(26), 6-43(120), 7-15(50), 7-16(51), 8-20(55), 8-25(71), 9-2(1), 9-4(13), 9-17(77), 9-37(146)
last(list)  1-9(26)
lbd(eva)  7-13(37), 7-12(34), 7-12(35), 8-37(118)
LDCL  11-3(3), 11-5(12), 11-97(260), 11-112(302), 11-113(307), 11-114(309), 11-114(310)
LE  2-18(155), 6-49(137), 9-10(45), 9-17(77)
left-aggr(aggr,eqn)  8-14(33), 7-15(48), 8-13(32), 8-14(33)
LST-PAR  9-3(5), 11-70(193), 11-88(245)
length(list)  1-8(21)
length-sum(lgth-1,lgth-2)  2-24(102), 9-23(100)
LIN  12-5(10)
LIN-ARR  12-5(10)
lin-fix(eda,pv)  9-58(227), 9-52(206), 9-62(242)
lin-f1t(eda,pv-1,pv-2)  9-58(227), 9-52(208), 9-62(243)
ABSTRACT SYNTAX AND INTERPRETATION OF PL/I

lin-flt-1 (pv)  2-10(86), 11-50(134), 11-51(140), 11-82(228), 11-94(232)
LIST  11-16(139), 11-51(137), 11-74(209), 11-79(221), 11-80(224), 11-87(238), 11-104(277)
list-dop (dop)  6-37(102), 6-37(101), 6-37(102), 12-5(9)
list-elem (arg-cl, expr, at)  11-86(239), 11-87(237), 11-89(242)
list-string (alt, eda, cl)  11-94(256)

lin-flt-1 (pv)  9-58(230), 9-58(228), 9-64(250)
LIST  11-16(139), 11-50(134), 11-51(140), 11-82(228), 11-94(232)

APPENDIX 47
IDM LAB VIENNA
ABSTRACT SYNTAX AND INTERPRETATION OF PL/I
30 April 1969

MESS-LENGTH
min(x, y)
MIN
min-flt(ida)
min-set(n-set)
MINUS
mix(spec-1, spec-2)
mix-init(init-1, init-2)
mix-spec(aggr, al, eva)
mix-spec-scalar(aggr, al, eva)
mix-spec-1(aggr, al, eva)
mix-spec-2(aggr, al, eva)
mk-arithm(ida)
mk-arithm-pic(ida)
mk-base-1(ida)
mk-bit-string(ida)
mk-chif(cond, chif, cs)
mk-chif-0(cond, chif, cs)
mk-char-string(ida)
mk-cond(io-cond, f, key)
mk-conv-chif(cond, chif, cs)
mk-cbif(cond, cbif, cs)
mk-cbif-0(cond, cbif, cs)
mk-dep(dp)
mk-eattn-chif(cond, chif, cs)
mk-edit-op(tr, vl, edfo)
mk-ext-e(t)
mk-f-cond(op, io-cond)
mk-f-field(sgn, d, dl)
mk-gen(eva, p)
mk-get-op(edn, v)
mk-id(cl)
mk-ident(id, ea)

min-12(29), 6-12(29), 8-27(76), 12-2(2), 12-3(4), 12-7(18), 12-8(20), 12-9(24), 12-45(75), 12-72(183)
min-flt(edn)
min-set(n-set)

mix-spec-1, spec-2

mix-spec-1, spec-2

mix-spec-1, spec-2

mix-spec-1, spec-2

mix-spec-1, spec-2

mix-spec-1, spec-2

mix-spec-1, spec-2

mix-spec-1, spec-2

mix-spec-1, spec-2

mix-spec-1, spec-2

mix-spec-1, spec-2

mix-spec-1, spec-2

mix-spec-1, spec-2

mix-spec-1, spec-2

mix-spec-1, spec-2

mix-spec-1, spec-2

mix-spec-1, spec-2

mix-spec-1, spec-2

mix-spec-1, spec-2

mix-spec-1, spec-2

mix-spec-1, spec-2

mix-spec-1, spec-2

mix-spec-1, spec-2

mix-spec-1, spec-2

mix-spec-1, spec-2

mix-spec-1, spec-2

mix-spec-1, spec-2

mix-spec-1, spec-2

mix-spec-1, spec-2

mix-spec-1, spec-2

mix-spec-1, spec-2

mix-spec-1, spec-2

mix-spec-1, spec-2

mix-spec-1, spec-2

mix-spec-1, spec-2

mix-spec-1, spec-2

mix-spec-1, spec-2

mix-spec-1, spec-2

mix-spec-1, spec-2

mix-spec-1, spec-2

mix-spec-1, spec-2
mak-lo-chif(cond,chif,eq)

mak-list(x,list)

mak-list-1(list,x)

mak-m-field(sgn,d,s,dl)

mak-nm-list(eda, sf, v)

mak-num-list-1(n,v)

mak-op(evav,vr)

mak-lg-gen(id,list)

mak-ref-list(n,idl-list)

mak-rl-1(evl,eva)

mak-ref-list-1(evl,eva)

mak-rl-1(evl,eva)

motrop(tr,cl)

mak-sterl-subl(eda, sf, n,v)

mk-str-1(eda)

mk-string(eda)

mk-substr-gen(g,lo,lg)

mk-str(par-opt)

mk-x-field(sgn,dl)

mod(expr, eva, i)

mod-built-in(ref, eva, i)

mod-built-in-explist(id, expr-list, eva, i)

mod-ref(ref, eva, i)

modify-entry(attr,aggr1)

modify-entry-list(attr-list, gnl)

modify-entry-1(aggr,aggr1)

modify-entry-2(aggr,aggr1)

modify-entry-3(aggr,aggr1)

modulo(x,y)

modulo-1(op-1,op-2)

mpic(eda)

mnk-1(eda)

msubf(eda,n)

MULT .2-18(155), .9-10(44), .9-15(71), .9-24(103), .9-24(104), .9-12(101), .9-12(101), .9-58(131), .9-58(133), .13-5(10)

Appendix 49
ABSTRACT SYNTAX AND INTERPRETATION OF PL/I

**N-CHAR**

**n-of-d(eda,sf,m,n)**

**n-of-d-subf(eda,sf)**

**NAME**

**E2**

**nest-descr-list(i,da)**

**new(x,da)**

**next-tab(col)**

**make-of-comp(eva,k)**

**no-area-cond(z,allst)**

**NO-COND**

**NO-STRG**

**node(c)**

**NORMAL**

**NOT**

**note-delay(tine)**

**NPTE**

**null**

**NUM**

**num-bit-comp(eda,v)**

**num-bit-list(n,v)**

**num-char-comp(eda,v)**

**num-char-list(n,v)**

**num-eda(dt)**

**num-pic-char-comp(eda,v)**

**num-pic-val(eda,v)**

**NUMBE-R-SIGN**

**O-CHAR**

**OFFSET**

**OCL**

---

50 APPENDIX
ABSTRACT SYNTAX AND INTERPRETATION OF PL/I

ON .................................................. 2-14 (114), 3-8 (16), 6-29 (76), 6-44 (122), 10-20 (60)
on-establish (cond, t) .................................. 10-12 (35), 10-11 (31), 10-12 (34)
on-establish-1 (cal, t) ................................ 10-12 (34), 10-11 (31)
onattn .................................................. 13-12 (26)
onchar ................................................. 11-38 (104), 11-92 (249), 12-67 (170), 12-69 (174), 12-74 (189), 13-12 (26), 13-17 (44)
oncode ................................................ 13-12 (26)
oncount ............................................... 13-12 (26)
onf (cond, fd) ...................................... 10-21 (63), 10-20 (62)
onil .................................................... 13-12 (26)
onil-length .......................................... 11-29 (83), 11-32 (92)
onident .............................................. 11-38 (104), 11-92 (249), 12-67 (170), 12-69 (174), 12-74 (189), 13-12 (26), 13-17 (44)
onkey ................................................. 13-12 (26)
onkey-length ........................................ 11-29 (83), 12-61 (146)
onloc .................................................. 13-12 (26)
on-length ............................................ 10-24 (70), 11-29 (83)
onshif (i, e) ...................................... 10-24 (71), 10-24 (70)
onsource ............................................ 11-38 (104), 11-92 (249), 12-67 (170), 12-69 (174), 12-74 (189), 13-12 (26), 13-17 (44)
op .................................................... 12-4 (5)
op-da (op) ........................................ 9-9 (37), 8-8 (18), 8-16 (39), 8-21 (57), 12-4 (8), 12-37 (48), 12-38 (54), 12-39 (60), 12-40 (63), 12-41 (66), 12-43 (73), 12-44 (76), 12-44 (78), 12-45 (81), 12-45 (83), 12-46 (84), 12-46 (85), 12-46 (86), 12-46 (87), 12-46 (90), 12-47 (92), 12-47 (94), 12-47 (95), 12-48 (96), 12-48 (97), 12-49 (98), 12-49 (102), 12-49 (104), 12-50 (106), 12-51 (110), 12-52 (112), 12-52 (113), 12-52 (114), 12-53 (119), 12-53 (116), 12-53 (117), 12-54 (120), 12-54 (123), 12-57 (130), 12-58 (133), 12-59 (135), 12-60 (141), 12-62 (152), 12-63 (156), 12-63 (157), 12-70 (176)
op-val (op) ........................................ 9-9 (37), 5-3 (9), 6-37 (103), 6-40 (115), 6-43 (120), 8-9 (23), 8-11 (27), 8-15 (37), 6-17 (48), 8-18 (49), 8-20 (55), 8-21 (57), 8-22 (60), 9-29 (120), 10-12 (33), 10-24 (70), 11-21 (56), 11-21 (56), 11-23 (66), 11-33 (94), 11-37 (103), 11-83 (229), 11-87 (238), 11-99 (266), 11-112 (304), 12-6 (14), 12-77 (48), 12-38 (54), 12-38 (56), 12-40 (63), 12-41 (66), 12-43 (73), 12-44 (77), 12-45 (83), 12-46 (88), 12-46 (88), 12-46 (94), 12-47 (95), 12-48 (96), 12-48 (97), 12-49 (98), 12-50 (106), 12-51 (109), 12-51 (110), 12-52 (112), 12-52 (113), 12-53 (115), 12-53 (116), 12-53 (117), 12-54 (119), 12-54 (120), 12-54 (123), 12-60 (141), 12-63 (156), 12-63 (157), 12-64 (168), 12-64 (169), 12-64 (170), 12-64 (171), 12-66 (177), 12-70 (177), 12-70 (178), 12-71 (181), 12-73 (184), 12-73 (184), 12-73 (187)
open (u, f, st-prt, csa, title, ea, vol, op-opt) ............. 11-27 (79), 11-26 (77)
open ................................................ 2-15 (128)
open-fu-el (csa, ea, op-opt) ........................ 11-28 (80), 11-27 (79)
open-1 (n, csa, title, ea, vol, op-opt) ............... 11-26 (77), 11-21 (56), 11-24 (67), 11-26 (76)
open-2 (u, f, vol) ................................ 11-27 (78), 11-26 (77)
open-3 (t, f, cond) ................................ 11-29 (68), 11-24 (67), 11-26 (76)

APPENDIX 51
ABSTRACT SYNTAX AND INTERPRETATION OF PL/I

30 April 1969

ORDER

order-set(set,type)

order-test-data-list(itl,type,at)

out-fix-pic(m,n)

out-ill-pic(a)

overlay-gen(gen,eva,i)

overpunch(sg,n)

p-mp(p)

PA

PA

PAGE

PARAM

parsing(cl,pred)

page(e)

PDIL

PEND

P2HC

PIC

pic-length(eda)

pic-prec(eda)

pic-val(eda,v)

PLUS

POINT

POINTER

posn(i)

position(mp,ids,key)

APPENDIX

52
<table>
<thead>
<tr>
<th>Function</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>poic(x)</td>
<td>9-44(166), 9-43(165), 9-44(171), 9-45(175), 9-45(178), 9-50(227), 9-58(228)</td>
</tr>
<tr>
<td>pre-eval(expr)</td>
<td>8-11(261), 6-23(62), 6-37(101), 6-47(130), 8-3(1), 8-3(2), 8-10(24), 8-11(26), 8-12(29), 8-17(47), 8-20(54), 9-28(82), 11-37(103), 11-48(132), 11-76(214), 12-5(9), 12-6(16)</td>
</tr>
<tr>
<td>pre-eval-ap(ap, id)</td>
<td>8-12(28)</td>
</tr>
<tr>
<td>pre-eval-arq(expr, id, 1)</td>
<td>8-12(28)</td>
</tr>
<tr>
<td>pre-eval-list(tp)</td>
<td>8-3(2), 8-3(1), 8-3(2)</td>
</tr>
<tr>
<td>pre-eval-ref(ref)</td>
<td>8-11(27), 8-11(26)</td>
</tr>
<tr>
<td>prec-fact(eda-tg, eda)</td>
<td>9-38(153), 9-38(152), 9-40(161)</td>
</tr>
<tr>
<td>pred(i, e)</td>
<td>1-24(47)</td>
</tr>
<tr>
<td>pred-1(e)</td>
<td>1-23(46)</td>
</tr>
<tr>
<td>pref(eda-1, eda-2)</td>
<td>12-34(40), 12-35(43), 12-37(48), 12-46(84), 12-49(100), 12-52(114), 12-53(115), 12-54(123), 12-57(130), 12-58(132)</td>
</tr>
<tr>
<td>pref-ar-eda(eda-1, eda-2)</td>
<td>9-22(99), 9-22(99)</td>
</tr>
<tr>
<td>pref-base(eda-1, eda-2)</td>
<td>9-22(95), 9-21(89), 9-22(93)</td>
</tr>
<tr>
<td>pref-mode(eda-1, eda-2)</td>
<td>9-22(99), 9-22(99)</td>
</tr>
<tr>
<td>pref-prec(eda-1, eda-2)</td>
<td>12-35(41), 12-39(49)</td>
</tr>
<tr>
<td>pref-scale(eda-1, eda-2)</td>
<td>9-22(96), 9-22(93)</td>
</tr>
<tr>
<td>pref-scale-f(eda-1, eda-2)</td>
<td>12-35(42), 12-34(40)</td>
</tr>
<tr>
<td>pref-str-base(eda-1, eda-2)</td>
<td>9-22(97), 9-22(97)</td>
</tr>
<tr>
<td>pref-str-eda(eda-1, eda-2)</td>
<td>9-22(97), 9-21(90), 12-34(40)</td>
</tr>
<tr>
<td>pref-varying(eda-1, eda-2)</td>
<td>9-22(97), 9-22(97), 12-37(51)</td>
</tr>
<tr>
<td>pref-1(eda, eda-list)</td>
<td>12-39(43), 12-35(43), 12-48(96), 12-48(97)</td>
</tr>
<tr>
<td>prefix-not(v)</td>
<td>9-28(119), 9-27(111)</td>
</tr>
<tr>
<td>prefix-nml(eda, v, opr)</td>
<td>9-27(112), 9-27(111), 9-27(113)</td>
</tr>
<tr>
<td>prefix-op(op, opr)</td>
<td>9-26(119), 9-26(109)</td>
</tr>
<tr>
<td>prefix-target(eda, opr)</td>
<td>9-28(119), 9-26(109), 9-28(117)</td>
</tr>
<tr>
<td>prefix-val(eda, v, opr)</td>
<td>9-27(111), 9-26(110)</td>
</tr>
<tr>
<td>prep-activate(ε-1, ε-2)</td>
<td>3-32(121), 3-32(121)</td>
</tr>
<tr>
<td>prep-attn(ε-1, ε-2)</td>
<td>3-32(122), 3-32(121)</td>
</tr>
<tr>
<td>prep-attr-1(task, cond)</td>
<td>10-14(191), 3-32(122), 10-5(10), 10-9(24), 10-15(44)</td>
</tr>
<tr>
<td>prep-decl(s, t)</td>
<td>4-6(131), 4-6(12)</td>
</tr>
<tr>
<td>prep-decl-1(b, evdecl)</td>
<td>8-9(24), 8-9(23)</td>
</tr>
<tr>
<td>prep-decls(t, t-den)</td>
<td>4-6(12), 4-6(10)</td>
</tr>
<tr>
<td>prep-decls-1(t-evdecl, t-den)</td>
<td>9-9(23), 4-6(12)</td>
</tr>
</tbody>
</table>
ABSTRACT SYNTAX AND INTERPRETATION OF PL/I

30 April 1969

IBM LAB VIENNA

PREP-FILE (fas, ea, id) .................................................. 3-34 (121), 3-30 (110), 13-2 (1), 13-16 (36)

PREP-INTERCEPT (e-1, e-2) .............................................. 3-34 (121), 3-30 (110), 13-2 (1), 13-16 (36)

PREP-TEXT (t, e) .............................................................. 5-6 (110), 4-3 (4)

PREP-TEXT-1 (t, t-den) .................................................... 9-6 (111), 4-6 (10)

PRI-EVA ........................................................................... 5-3 (5), 5-3 (4), 5-3 (8), 12-64 (159), 12-69 (174)

PRI-PREC ........................................................................... 5-3 (5), 5-3 (5), 5-9 (16)

PRI-sched (e) ...................................................................... 3-30 (112), 3-30 (111), 3-30 (113), 13-1 (3), 13-9 (15), 13-16 (35)

PRIORITY ........................................................................... 12-67 (170), 12-69 (174), 12-74 (189), 13-12 (26)

PROC .................................................................................. 3-8 (16), 6-16 (43), 6-29 (76)

PROGRAM-PRELUDE (t, e) .................................................... 4-4 (71), 4-3 (8)

PROGRAM-STOP .................................................................. 5-6 (15), 5-6 (14)

PROLOGUE (dp, bp, cp, e) ..................................................... 5-6 (15), 5-6 (14)

PRT .................................................................................... 2-3 (19), 3-26 (98), 4-9 (22), 11-4 (8), 11-5 (12), 11-9 (28), 11-22 (62), 11-25 (72), 11-26 (76), 11-26 (77), 11-28 (80), 11-28 (81), 11-84 (232), 11-87 (237), 11-97 (260), 11-122 (329), 11-123 (332)

PS-gen-op (id, list) .............................................................. 12-74 (189), 6-49 (136)

Pseudo-assign (ps-gen, op) ................................................. 12-68 (172), 8-8 (16), 13-17 (44)

Pseudo-assign-contd (ps-gen, op) ........................................ 12-68 (173), 12-68 (173)

Pseudo-assign-contd-9 (ps-gen, op) ..................................... 12-68 (173), 12-68 (173)

PTR ..................................................................................... 2-3 (34), 2-6 (50), 2-7 (58), 2-20 (173), 2-21 (180), 6-24 (63), 6-28 (73), 7-4 (7), 7-9 (21), 7-17 (56), 7-17 (56), 8-8 (17), 8-8 (18), 8-16 (39), 8-19 (50), 9-2 (1), 9-4 (13), 9-17 (77), 9-37 (146), 11-58 (154), 11-60 (158), 11-61 (159), 12-42 (70), 12-60 (139), 12-62 (150), 12-62 (153), 12-63 (156), 12-68 (173), 13-5 (11), 13-12 (25)

PTR-CONV (op, area) .......................................................... 8-16 (39), 8-15 (38), 8-19 (50)

PTR-OFFSET-CONVERT (eva, op, area-1, area-2) ............... 8-8 (16), 8-8 (17)

pure-aggr (aggr) ............................................................... 6-20 (52), 4-6 (13), 6-26 (51), 6-20 (52), 6-23 (62), 13-11 (22)

pure-da (da) ...................................................................... 6-20 (53), 6-20 (52), 6-21 (55), 6-22 (61)

PUT ..................................................................................... 2-16 (136), 2-16 (137), 11-48 (132), 11-50 (134), 11-51 (137), 11-74 (209), 11-77 (215), 11-82 (226), 11-94 (254), 11-95 (255), 11-96 (256), 11-121 (327), 11-122 (329), 11-123 (331), 11-123 (332)

PUT-COLUMN (tr, w) .......................................................... 11-87 (258), 11-82 (228), 11-97 (258)

PUT-CONTROL-FORMAT (tr, efo) ........................................ 11-82 (228), 11-82 (228)

PUT-DATA (tr, id, intgl, op) .............................................. 11-88 (240), 11-79 (221)

PUT-DATA-FIELD (tr, expr) ............................................... 11-79 (221), 11-77 (215)

PUT-EDIT (tr, op, edfo) .................................................... 11-98 (263), 11-79 (221)

PUT-EDIT-1 (tr, op, edfo, mode) ....................................... 11-98 (264), 11-98 (263)

PUT-FILE (tr, col, el) ...................................................... 11-94 (253), 11-82 (228), 11-91 (247), 11-97 (260), 11-98 (262), 11-122 (329)

54 APPENDIX
IBV LAB VIENNA

30 April 1969

ABSTRACT SYNTAX AND INTERPRETATION OF PL/I

put-line(tr,w,endp) ........................................ 11-27(261), 11-82(228), 11-97(261)
put-list(tr,op) .................................................. 11-87(231), 11-79(221)
put-page(tr) ...................................................... 11-93(262), 11-51(140), 11-82(228), 11-97(261), 11-123(332)
put-skip(tr,w,endp) ........................................ 11-87(259), 11-82(228), 11-91(247), 11-97(258), 11-97(259), 11-122(329)
put-space(tr,w) ................................................... 11-26(257), 11-82(228), 11-97(258)
put-sticky(tr,v) ................................................ 11-21(248), 11-91(247)
put-val-list(tr,vl) ........................................... 11-21(247), 11-74(209), 11-87(237), 11-89(241), 11-91(247), 11-96(257), 11-99(264), 11-121(327), 11-123(331)
Q-CHAR .................................................................... 9-4(8)
caller-name(t,e) ....................................................... 6-3(50), 6-3(64), 6-2(41), 6-16(43)
caller-name-1(t,u) ................................................... 6-3(60), 6-3(5)
QUEST ..................................................................... 9-3(6)
R-CHAR ................................................................... 2-9(70), 9-4(6), 9-46(183), 9-46(185), 9-56(219)
r-data-label(a,ct) ................................................. 11-31(88), 11-31(88)
read(md,ds,ad) ....................................................... 11-12(42), 11-45(123), 11-62(160), 11-64(165), 11-94(254)
READ ....................................................................... 2-17(144), 2-17(145), 2-17(146), 11-40(110)
real(z) ..................................................................... 1-14(37)
real-app-eda(dt) ...................................................... 11-107(289), 11-106(285), 11-107(286)
real-char-conv(eda,v) .............................................. 2-31(126), 9-30(123), 9-31(126)
real-eda(dt) ............................................................. 11-106(285), 11-106(284), 11-117(319)
real-val(dt) .............................................................. 11-72(297), 11-72(206), 11-117(320)
real-val-1(dt,base) .................................................. 11-72(208), 11-72(207), 11-72(208)
REC ....................................................................... 2-3(19), 2-14(121), 3-20(75), 3-26(98), 4-9(22), 11-4(8), 11-4(11), 11-5(12), 11-11(40), 11-11(41), 11-14(46), 11-15(47), 11-24(69), 11-25(75), 11-28(80), 11-55(142), 11-56(148), 11-56(146), 11-57(151), 11-60(157)
rec-code(z-1,z-2) .................................................... 11-56(148), 11-55(142), 11-57(151), 11-57(149), 11-57(151), 11-60(157), 11-62(160)
rec-loc-transmission(et,add) ................................... 11-54(141), 11-43(116)
reduce-aggr1(aggr,aggr1) ....................................... 6-32(62), 6-32(61), 6-32(62)
reduce-gml(gml,lg) .................................................. 6-31(78), 6-31(77), 6-31(78)
ref-dim(aggr,cl) ....................................................... 8-29(84), 8-28(83), 8-29(84)
release(entry-list) ................................................... 6-37(106), 6-37(104)

APPENDIX 55
ABSTRACT SYNTAX AND INTERPRETATION OF PL/I

30 April 1969

IBM LAB VIENNA

RELEASE

RENAME

remove(list,i)

rep(eva,v)

rep-bin-sign(eda,sf,sgn)

rep-conc(i,list-1,list-2)

rep-digit(spec,sg,n)

rep-num-list(eda,sf,sg,ml)

rep-nul-pic-1(eda,v)

rep-nul(eda,v)

rep-real-pic-1(eda,v)

rep-sign(eda,sf,sg,pv)

rep-steri-dig(spec,n)

rep-subf(eda,sf,v)

rep-valid-subf(eda,sf,v)

res-c-exp-eda(eda-1,eda-2,v-2)

res-eda(eda-1,eda-2,opr)

res-prec(eda-1,eda-2,opr)

res-scale-f(eda-1,eda-2,opr)

reset-eql(eql)

result-length(eda,v)

ret-val(a-1,j,cp)

retry-on-transmission(cond,et)

retry-stream-transmission(tr,cond,arg)

RETURN

RETURN-free(b,aggr)

return-nod(expr)

REVERT

revert-on(cond)

revert-on-1(cdl)

56 APPENDIX
ABSTRACT SYNTAX AND INTERPRETATION OF PL/I

rewrite(MP,Os,el) rewrite-transmission(ch) RIGHT-PAR rp-field(tr,cl)
ABSTRACT SYNTAX AND INTERPRETATION OF PL/I

30 April 1969

58 APPENDIX
IBM LAB VIENNA

30 April 1969

ABSTRACT SYNTAX AND INTERPRETATION OF PL/I

single-bit-char(v) ......................................................... 2-25(139), 9-35(138), 11-121(327)
single-bit-num(v) ......................................................... 2-34(137), 9-18(81), 9-34(136)
single-char-bit(v) ......................................................... 2-34(135), 9-33(134), 11-105(282), 11-116(323)
single-infix-bit(v-1,v-2,opr) ........................................ 2-12(86), 9-19(85)
single-num-bit(v) ......................................................... 9-33(133), 9-33(132)
single-num-char(v) ....................................................... 2-32(129), 9-32(128), 9-56(219), 9-56(222), 11-72(208), 11-90(246)
single-prefix-not(v) ...................................................... 2-28(116), 9-28(115)
SINH ................................................................. 12-11(26)
size{vr} ............................................................... 12-42(70), 12-42(72), 13-16(37)
size-el(rop,ds,el) ...................................................... 11-55(142), 11-57(151), 11-60(157), 11-62(160), 11-63(163), 11-64(165), 12-42(70), 12-42(72), 13-16(37)
SIZE ............................................................... 2-14(118), 3-19(74), 9-7(27), 9-53(209), 10-10(26), 10-18(55), 11-101(271), 11-109(294)
size-el(mp,ds,el) ...................................................... 11-10(35), 11-10(36), 11-10(38), 11-11(40), 11-14(46), 11-15(47), 11-55(142), 11-57(151)
size-vr(vr-1,vr-2) ...................................................... 11-10(35), 11-10(36), 11-10(38), 11-11(40), 11-14(46), 11-15(47), 11-55(142), 11-57(151)
skip ................................................................. 2-10(86), 11-50(134), 11-51(140), 11-55(142), 11-57(150), 11-58(153), 11-82(226), 11-82(227), 11-82(228), 11-84(232)
SLASH ................................................................. 2-9(70), 2-9(76), 9-3(6), 9-46(180), 9-47(189), 9-50(201)
SQAED-ACTION(cond) .................................................. 10-19(84), 10-19(57)
SPACE ............................................................... 2-10(86), 11-82(227), 11-82(228), 11-83(229), 11-84(232)
special-initialize(gen,init) ............................................ 7-16(51), 7-16(50), 7-16(51)
SQT ................................................................. 12-11(26)
SQT-CPLX ............................................................ 12-54(120)
SQT-REAL ........................................................... 12-54(120)
ST-PRT ........................................................... 3-24(87), 5-8(26), 11-24(70), 11-27(79), 11-30(87), 11-79(220)
stack(ba,ei,c,ds,el) .................................................. 6-24(62), 6-2(1), 6-16(43), 10-14(39), 10-29(60)
stack-cd(index,t) ...................................................... 6-39(168), 6-38(107), 6-40(113), 10-17(50)
start-line(tl) ........................................................... 11-97(269), 11-97(259), 11-97(261)
STATIC ........................................................ 2-3(13), 4-6(10), 4-6(13), 4-7(14), 4-9(24), 4-9(24), 6-8(23), 7-11(25)
STATUS .......................................................... 12-67(179), 12-69(174), 12-74(189), 13-12(26)
stc-overflow ......................................................... 7-5(11), 7-5(8), 7-10(24), 7-11(25), 11-59(155), 11-61(159)
STOP ................................................................. 2-13(108)

APPENDIX 59
ABSTRACT SYNTAX AND INTERPRETATION OF PL/I

str-eval(base, j) .... 7-18(45), 7-13(40), 7-13(41), 5-7(24), 9-7(25), 11-92(249), 12-42(70), 12-42(72), 12-68(173), 12-72(183)
str-length(eda) .... 9-31(164), 7-14(44), 8-34(108), 9-4(10), 9-7(25), 9-33(134), 9-40(162), 11-68(239), 11-93(251), 12-60(141)
str-part(base, i, j) .... 7-14(39), 7-13(40), 7-13(41), 7-13(42), 7-13(43), 7-14(44), 8-33(104), 9-7(24)
stream-base(fu-el) .... 11-94(233), 11-74(209), 11-82(226), 11-84(232), 11-97(237), 11-95(257), 11-97(260), 11-122(329), 11-123(332)
stream-transmission(tr, arg) .... 11-94(234), 11-96(253), 11-96(256), 11-113(306)
STREAM .... 2-14(119), 3-19(74), 10-10(26), 10-18(55), 12-41(65)
STRING .... 11-38(104), 11-85(234), 11-92(249), 12-57(170), 12-68(173), 12-74(189)
STRING-JSA .... 2-11(55), 9-11(56), 9-11(57), 9-12(60), 11-48(132), 12-39(62)
string-extent(eva) .... 8-38(109), 7-14(44), 8-33(104), 8-34(108)
struct-elem(id, i, j, el) .... 12-9(23), 12-8(24)
STRING .... 2-14(118), 3-19(74), 9-7(27), 10-10(26), 10-18(55)
sub-aggr(aggr, sl, eql) .... 8-26(175), 8-13(32), 2-25(71), 8-26(75), 8-26(76), 10-22(67)
sub-eva(eva, rl) .... 8-37(118), 8-36(115), 8-37(118)
sub-gen(gen, rl) .... 8-36(115), 5-11(33), 5-26(69), 7-9(22), 7-15(49), 7-16(51), 7-18(59), 7-19(60), 8-15(37), 8-29(87), 8-31(35)
sub-pp(pp, rl, mi) .... 8-38(116), 8-38(116), 8-36(115), 8-36(116)
sub-pp-1(b, rl, mi) .... 8-37(117), 8-36(116), 8-37(117)
SUBSTR .... 2-14(119), 3-19(74), 9-7(27), 10-10(26), 10-18(55)
substr(str, i, j) .... 12-40(63), 12-40(63), 12-72(183)
SUBSTR .... 2-14(119), 9-30(92), 10-10(26), 10-18(55)
str-eval(eva, i, g) .... 12-71(189), 12-60(141), 12-71(179), 12-72(182)
str-eval(g, lo, lg, cbif) .... 11-22(249), 11-62(161), 11-91(248), 11-92(249), 11-105(279)
SUBSTR .... 2-19(155), 9-10(44), 9-15(71), 9-24(103), 9-24(104), 12-64(159)
ST .... 3-26(96), 8-11(33), 8-13(32), 8-15(37), 8-24(104), 8-31(35)
SW-SOV .... 3-26(96), 11-27(78), 11-28(80), 11-31(89), 11-38(96)
SW-SOV .... 10-24(72), 11-26(76), 11-26(77)
syst-check-exe(ref) .... 11-123(331), 10-10(26)
syst-cond-exe(cond, cbif) .... 10-10(26), 10-19(56)
syst-engage-exe(t) .... 11-123(332), 10-10(26)
syst-error-exe .... 10-11(30), 10-10(26)
syst-exe-exe .... 2-14(114), 3-21(32)
ABSTRACT SYNTAX AND INTERPRETATION OF PL/I

T-CHAR

tab(tr,eda,ci)

TAB-SLT

TABL

tail(list)

tail-1(list)

take-at(index,t)

TAN

TANH

target(eda-1,eda-2,opr)

target-1(eda-1,eda-2,opr)

target-2(eda-1,eda-2,opr)

TASK

task-epilogue(status)

task-exit

task-fd-close

TE

term-events

term-noise(c)

term-tasks

term-tasks-wait

term-tasks-1

terminate-transmission(list,et)

test-activate(1,1,2)

test-and-allocate(et,skip)

test-and-prepare(et,skip)

test-assign(gen,op)

test-attn(cdl,an,er,tn)

test-built-in-op(op,test)

test-charset(eda,v)

APPENDIX 61
ABSTRACT SYNTAX AND INTERPRETATION OF PL/I

11-78(213), 11-74(209), 11-75(211)
11-107(288), 11-107(287)
11-108(289), 11-107(288), 11-108(290)
11-108(290), 11-109(289)
11-110(296), 11-109(295)
11-110(297), 11-110(296), 11-110(298)
11-110(297), 11-110(296), 11-110(297)
5-13(43), 5-13(43)
12-3(13), 12-3(12)
5-28(7), 6-28(73)
9-15(72), 9-13(66), 9-15(72), 9-27(111), 12-50(106)
11-113(307), 11-95(255)
11-95(255), 11-95(255), 11-113(306)
2-14(67), 9-13(66)
12-6(13), 12-5(12)
12-6(13), 12-5(12)
11-99(249), 11-98(240)
9-29(120), 9-32(130), 11-106(283), 12-36(46)
8-32(102), 8-32(102)
8-30(131), 8-30(131)
11-78(219), 11-78(217), 11-80(223)
9-14(66), 5-13(43)
5-12(38), 5-12(40)
12-9(5)
13-12(26)
2-14(121), 3-20(75), 3-27(102), 3-31(117), 11-24(68), 11-31(89), 11-31(89), 11-32(92), 11-33(94), 11-33(94), 11-34(95), 11-56(144), 11-56(145), 11-56(146), 11-78(219), 11-94(254)
ABSTRACT SYNTAX AND INTERPRETATION OF PL/I

tmt-sel(f,fd) ........................................ 11-79(229), 11-78(219), 11-94(254)

TN ......................................................... 5-6(16), 5-7(23), 5-7(25), 10-5(10),
10-6(11), 10-8(23), 10-9(25), 10-15(44),
10-16(47), 10-17(50), 10-17(52), 10-18(53),
10-13(55), 11-41(113)

tr-par(t) ................................................... 11-51(156), 11-48(132), 11-50(138)

TR-STR ...................................................... 12-41(66), 12-41(67)

TRA ......................................................... 2-3(19), 3-26(98), 4-9(22), 11-4(8),
11-13(44), 11-25(73), 11-28(80), 11-31(85),
11-44(120)

trans-wait-s(t) ........................................... 5-16(301), 5-10(29)

trans-last(s, a-1, j, cp) .................................. 12-44(68), 12-42(68)

translate(str, repl, pos) .................................. 1-13(32)

trunc(x) ....................................................... 6-40(115), 6-40(114), 6-48(135),
6-49(137)

truth-to-bit(x) ............................................. 9-4(8)

truth-val(op) ............................................... 9-4(8), 10-4(8), 10-14(41),
10-24(70), 11-26(77), 11-31(89), 11-40(108),
11-41(112), 11-48(132)

ubd(eva) ..................................................... 7-13(38), 6-24(64), 7-12(34),
7-12(35), 3-5(7), 8-37(118), 11-77(216)

UN ............................................................. 2-14(118), 3-19(74), 9-15(72),
10-10(26), 10-18(55)

UN ............................................................. 3-10(20)

UN-namc ...................................................... 3-10(20), 4-3(5), 4-6(10),
5-2(1), 5-2(3), 5-10(29), 6-3(4), 6-4(8),
6-8(22), 6-13(35), 6-21(57), 6-23(62),
6-27(71), 8-17(49), 10-4(8), 10-14(41),
10-24(70), 11-26(77), 11-31(89), 11-40(108),
11-41(112), 11-48(132)

UNAL ......................................................... 2-5(33), 2-6(49), 2-7(57),
2-20(172), 2-21(179), 7-14(45), 8-24(65),
8-33(106), 8-39(107), 9-7(25)

UNB ............................................................. 2-3(19), 4-9(22), 11-4(6),
11-4(10)

UNDC ......................................................... 2-14(121), 3-20(75), 10-26(77),
11-24(68), 11-27(78)

unite(set-1, set-2) ........................................ 11-57(152), 11-54(141)

unlock(et) .................................................... 11-47(131), 11-41(112)

UNLOCK ...................................................... 2-18(151)

unlock-taskend ............................................. 5-7(25), 5-7(23)

unnest-cl(scl) ............................................. 11-71(198), 11-71(197),
11-71(198), 11-71(199), 11-72(209), 11-107(286),
11-109(292)

UNSPEC ....................................................... 11-38(104), 11-85(234),
11-92(249), 12-67(170), 12-68(173), 12-74(189)

unspec-length(z) ........................................... 12-42(77), 11-92(249),
12-42(77), 12-42(72), 12-68(173)

unstack ....................................................... 5-11(33), 6-11(31)

UPC ......................................................... 2-3(19), 4-9(22), 11-4(8),
11-8(26), 11-8(27), 11-9(30), 11-10(36),
11-11(40), 11-14(45), 11-14(46), 11-15(47),
11-24(67), 11-25(74), 11-29(75), 11-33(94)

urde-at(spe) ................................................ 5-4(10), 4-4(7), 6-4(8)

urde-ba(ba) .................................................. 5-4(5), 6-4(8)
ABSTRACT SYNTAX AND INTERPRETATION OF PL/I

30 April 1969

upd-cs(onl,nol) ........................................ 10-24(11), 6-4(8)
upd-decl-dn(id,e,bp) .................................... 6-24(12), 6-5(12)
upd-decl-dn-1(s,dn) ...................................... 6-5(12), 6-5(13), 6-27(71)
up-dn(e,bp) ................................................ 6-5(11), 4-4(7), 6-4(7)
up-dn-1(e,bp) .............................................. 6-5(12), 6-5(11), 6-5(13)
upd-ci(argr) ................................................. 6-17(95), 6-16(44)
upd-format-ci(fol) ......................................... 11-84(239), 11-83(229)
upd-fuellem(u,s,e1) ........................................ 11-50(135), 11-50(134), 11-78(217), 11-80(223), 11-91(248), 11-94(253), 11-97(260), 11-98(262), 11-103(274), 11-112(305), 11-113(307), 11-123(331)
upd-index .................................................... 6-39(112), 6-39(111)
upd-loc-string(tr) ........................................ 11-103(274), 11-103(273), 11-109(295), 11-110(296), 11-115(311)
upd-st-cs(onl,nol) ........................................ 10-3(6), 6-40(116)
update-chif(cond,cbif) ..................................... 10-25(73), 10-20(60)
update-chif-1(cond,cbif) .................................. 10-25(73), 10-25(73)
v-char .......................................................... 9-4(8)
v-set(eda) .................................................... 9-5(161), 9-5(15), 9-6(20), 9-6(21)
v-set(eda) .................................................... 9-5(17), 9-5(15), 9-5(17), 9-6(21), 9-15(70), 9-16(79), 9-27(113), 12-51(107)
val(eva,vr) .................................................. 9-6(19), 4-2(2), 5-8(28), 5-11(36), 9-5(15), 9-6(21), 9-6(23), 9-7(24), 9-7(25), 9-7(26), 11-62(161), 11-87(238), 11-91(248), 11-95(266), 12-42(72)
val-op(eda,v) ............................................... 9-9(38), 7-9(21), 7-9(22), 7-16(51), 8-10(25), 8-20(52), 8-20(53), 8-20(55), 9-10(39), 9-10(40), 9-13(65), 9-25(110), 10-24(70), 11-32(92), 11-80(223), 12-37(48), 12-38(53), 12-38(57), 12-40(63), 12-40(63), 12-41(66), 12-44(79), 12-48(85), 12-46(86), 12-47(95), 12-49(104), 12-50(106), 12-54(123), 12-60(139), 12-60(139), 12-60(140), 12-61(144), 12-61(145), 12-61(147), 12-61(148), 12-62(150), 12-62(150), 12-62(153), 12-63(154), 12-63(156), 12-63(157), 12-64(158), 12-65(163), 12-65(165), 12-66(168), 12-70(176), 12-70(177), 12-70(178), 12-72(183), 12-72(183), 12-73(184), 12-73(187)
val-op-1(eda,v) .......................................... 12-36(46), 12-44(77), 12-45(83), 12-47(94), 12-48(96), 12-48(97), 12-48(97), 12-51(110), 12-57(130)
valid-initial-state(stg,es,time,date,tv,ev,t,call,gen) ........................................ 9-24(2), 4-1(1)
value(eva,vr) .............................................. 9-7(126), 9-9(36), 9-33(131)
value-conv(eda-tg,eda,v) ................................ 9-29(121), 9-29(120)
vary ......................................................... 2-5(36), 2-6(51), 2-7(63), 9-4(14), 9-12(60), 9-22(99), 9-37(148), 11-93(251), 11-93(252), 12-39(61), 12-40(63)
varying ..................................................... 12-34(39)
verify(arg,str) ........................................... 12-43(78), 12-43(73)
verify-b(tr,cl) ............................................ 11-118(321), 11-115(312), 11-118(321)
verify-e(tr,ft,cl) .......................................... 11-116(313), 11-115(312), 11-116(313)
verify-impl-open(t,f) ..................................... 11-24(69), 11-24(68)
ABSTRACT SYNTAX AND INTERPRETATION OF PL/I

verify-p(tr,pic,cl)  11-118(324), 11-115(372), 11-118(324)
verify-remote-format(s,fol)  11-84(231), 11-83(229)
verify-width(v1,edfo)  11-101(271), 11-99(269)
vr-part(pp,stg)  13-19(16), 13-9(15), 13-10(16)
vr-set(eva)  9-6(20), 9-6(21), 9-6(22), 9-6(23)
w-CHAR  9-4(8)
wait(genl,count,w)  5-12(49), 5-11(35), 5-12(90)
wait  2-12(105)
wait-acc(cdr)  10-17(52), 10-17(50), 10-17(52)
wait-count(genl,count)  5-11(34), 5-10(31)
wait-delay(time)  5-15(50), 5-15(48), 5-15(50)
wait-for-input(et)  11-44(121), 11-44(159), 11-44(121)
wait-for-reply(s,n)  11-124(334), 11-43(116), 11-124(334)
wait-for-unlock(et)  11-47(129), 11-47(129), 11-54(141)
while-continue(truth,t)  6-46(129), 6-46(128)
write-label(u,ct,op)  11-33(24), 6-44(122), 11-32(92)
write(rnp,ds,el)  11-15(97), 11-55(142), 11-57(151), 11-94(254)
write  2-17(147)
write-transmission(et)  11-57(151), 11-57(150)
wrong-pos(cl,pred)  11-71(201), 11-71(202), 11-104(278), 11-116(213), 11-118(321)
x-CHAR  2-9(76), 9-4(8), 9-59(235)
xout-fix-pic(b,n)  9-60(238), 9-60(237)
xpic(eda)  9-45(172), 9-51(205), 9-67(240), 9-63(246), 9-63(248)
y-CHAR  2-9(70), 9-4(8), 9-44(171), 9-46(182), 9-50(202), 9-51(203), 9-56(219)
z-CHAR  2-9(70), 9-4(8), 9-47(187), 9-49(200)
zdiv  2-14(118), 3-19(74), 9-14(68), 10-16(25), 10-18(55)
0-BIT  4-2(2), 5-4(10), 9-4(11), 9-9(33), 9-17(76), 9-19(86), 9-28(116), 9-28(116), 9-33(133), 9-34(135), 9-34(137), 9-35(139), 9-54(213), 10-7(19), 11-41(113), 11-96(257), 12-37(50), 12-65(165)
1-BIT  5-8(28), 5-11(36), 6-40(115), 8-9(23), 9-4(11), 9-17(76), 9-19(86), 9-19(86), 9-28(116), 9-28(116), 9-33(133), 9-34(135), 9-34(137), 9-35(139), 9-54(213), 12-37(50), 12-65(165)
1-CHAR  2-9(74), 9-4(9), 9-32(129), 9-33(134), 9-34(135), 9-35(139), 9-49(197), 11-68(177)
2-CHAR  2-9(74), 9-4(9), 9-32(129), 9-49(198)
3-CHAR  2-9(74), 9-4(9), 9-32(129), 9-49(199)

APPENDIX 65
ABSTRACT SYNTAX AND INTERPRETATION OF PL/I

ABSTRACT

SYNTAX

AN INTRODUCTION

PL/I 30

April 1969

APENDIX

9-4 (9), 9-32 (129)

9-4 (9), 9-32 (129)

9-4 (9), 9-32 (129), 9-46 (181), 9-50 (202), 9-55 (217), 9-56 (219), 9-56 (224)

9-4 (9), 9-32 (129), 9-46 (181), 9-50 (202), 9-55 (217), 9-56 (219), 9-56 (224)

9-4 (9), 9-32 (129), 9-46 (181), 9-50 (202), 9-55 (217), 9-56 (219), 9-56 (224)

9-4 (9), 9-32 (129), 9-46 (181), 9-50 (202), 9-55 (217), 9-56 (219), 9-56 (224)

9-4 (9), 9-32 (129), 9-46 (181), 9-50 (202), 9-55 (217), 9-56 (219), 9-56 (224)