ABSTRACT SYNTAX AND
INTERPRETATION OF PL/I

K. WALK
K. ALBER
K. BANDAT
H. BEKIĆ
G. CHROUST
V. KUDIELKA
P. OLIVA
G. ZEISEL

IBM LABORATORY VIENNA
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 transition function. 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 25.086).
PREFACE

This document is part of a series of documents which represent the formal definition of syntax and semantics of PL/I issued by 28 June 1968:

1/ LUCAS, P., LAUER, P., STIGLEITNER, H.: Method and Notation for the Formal Definition of Programming Languages.


   IBM Laboratory Vienna, Techn. Report TR 25.084.


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

7/ PL/I Definition Group of the Vienna Laboratory: Formal Definition of PL/I.

An outline of the method is given in /1/, which 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 inside and outside IBM.

The language defined in this present version is PL/I as specified in the official PL/I Language Specifications Form No. Y33-6003 with the exception of the following features which are not included:

- optimizing attributes (they are included in the concrete syntax but not in the abstract syntax; they are only tested for compatibility with other attributes and used for implication of default attributes),
- implicit conversion between offsets and pointers,
- the REFER option,
- the implicit rules for ordering initializing actions in the prologues of blocks and procedures.

The draft for this document was completed by 11 December 1967. It has been subject to validation by members of the PL/I Language Department of IBM UK Laboratories Hursley, England. The results of the checking effort conducted in Hursley have been taken into account in this present corrected form.

The formal definition given here includes more details than are given in the Specifications. These details have been confirmed as far as possible by the PL/I Language Department Hursley during the validation process. Some amendments and clarifications to the Specifications were generated during this process and will be published as Technical News Letters to the Specifications.

Contribution to the document:

- Project coordinator: K. Walk
- Authors: their main contributions are given by chapters
  - H. Bekić: 8., revision of 11.
  - G. Chroust: preparatory work for 8.
  - V. Kudielka: 10.4.
  - P. Oliva: 10. (revision of 10.4)
  - K. Walk: 6., 7., parts of 11.
  - G. Zeisel: 5.6.5., 7.2.3., revision of 9.
Coordination of production: F.Schwarzenberger, H.Hoja, W.Pachl
Production of the cross-reference index: K.F.Koch
Typing: H.Deim, C.Nowotny, W.Schatzl
Special graphics: G.Lehmayer

Coordination of validation: C.D.Allen
Validators: their main contributions are listed in chapters
C.D.Allen: 8., 11.
M.R.Barnett: 10. (except 10.5)
J.Fabri: 11.
A.Fletcher: 10. (except 10.5)
G.G.Hay: 11.
M.O'Donnell: 10.5
J.Pugh: 8., 11.
A.Rogers: 5.
R.Steel: 11.
R.W.Thomas: 10., 11.

The validation group was supported in Hursley by H.Bekić.
# CONTENTS

1. NOTATION AND CONVENTIONS 1-1

1.1 Notation of Meta-Expressions 1-1
   1.1.1 Conditional Expressions 1-1
   1.1.2 Functional Composition 1-2
   1.1.3 Objects 1-3
   1.1.4 The μ-function 1-4
   1.1.5 Manipulation of Lists 1-5
   1.1.6 Equality 1-7
   1.1.7 Truth Values, Logical Operators and Quantifiers 1-7
      1.1.7.1 Truth Values 1-7
      1.1.7.2 Logical Operators 1-7
      1.1.7.3 Quantifiers 1-8
      1.1.7.4 Description 1-9
   1.1.8 Arithmetic Operators and Relations 1-9
      1.1.8.1 Operators 1-9
      1.1.8.2 Relations 1-10
      1.1.8.3 Arithmetic Functions 1-10
   1.1.9 Set Operators, Relations and Notation for Sets 1-11
      1.1.9.1 Set Operators 1-11
      1.1.9.2 Relations 1-12
      1.1.9.3 Notation for Sets 1-12
   1.1.10 Rules of Precedence 1-12
   1.1.11 Naming 1-13

1.2 Definitions 1-13
   1.2.1 Metavariables 1-13
   1.2.2 Abbreviations 1-14
   1.2.3 Formulas 1-14
      1.2.3.1 Function Definitions 1-14
      1.2.3.2 Instruction Definitions 1-15
      1.2.3.3 Predicate Definitions 1-19
      1.2.3.4 Logical Statements 1-19
      1.2.3.5 References 1-19

1.3 The Treatment of Program Errors 1-20

1.4 Generally Used Simple Instructions 1-21
2. STATE COMPONENTS AND COMPUTATION OF THE PL/I MACHINE

2.1 The Computation of the PL/I Machine
   2.1.1 The Parallel Task and Input and Output Event Part PA
   2.1.2 The Task and Event Specification TE
   2.1.3 The Control C
   2.1.4 The Time and Date Part TD
   2.1.5 The Event Trace ET
   2.1.6 The Language Function of the PL/I Machine
   2.1.7 Sharing of Storage by Parallel Tasks and Input and Output Events

2.2 Flow of Control
   2.2.1 The Dump D
   2.2.2 The Epilogue Information EI
   2.2.3 The Control Information CI

2.3 Associating Identifiers with Meaning
   2.3.1 Creation of Unique Names
   2.3.2 The Environment E
   2.3.3 The Attribute Directory AT
   2.3.4 The Denotation Directory DN

2.4 Storage Access
   2.4.1 The Storage Part S
      2.4.1.1 Geometric Properties of Storage
      2.4.1.2 The Value Representation
      2.4.1.3 The Allocation State
      2.4.1.4 Usability of Storage Space
      2.4.1.5 Properties of the Assignment Function
      2.4.1.6 Properties of the Allocation Function
      2.4.1.7 Properties of the Freeing Function
      2.4.1.8 Properties of the Clearing Function
      2.4.1.9 The Main Storage
   2.4.2 The Aggregate Directory AG

2.5 State Components for Conditions
   2.5.1 Static and Dynamic Condition Selectors
   2.5.2 Condition State CS

2.6 Input and Output
   2.6.1 The File Directory FD
   2.6.2 The File Union Directory FU
   2.6.3 The External Storage ES
      2.6.3.1 Outer and Inner Data Sets
      2.6.3.2 External Storage Mapping
   2.6.4 The Message Part M
3. PROGRAM INITIALIZATION
3.1 Initial State
3.2 Initial Control
3.3 Prepass

4. TASKS, INPUT AND OUTPUT EVENTS AND PROGRAM SYNCHRONIZATION
4.1 Attaching of Tasks
4.2 Termination of Tasks
4.3 The Wait Statement
   4.3.1 The Waiting Mechanism
   4.3.2 Completion of Input and Output Events
4.4 The Delay Statement

5. FLOW OF CONTROL WITHIN A SINGLE TASK
5.1 Dynamics of Block Structure
   5.1.1 The Dump
   5.1.2 The Environment
   5.1.3 The Attribute Directory
   5.1.4 The Condition State
   5.1.5 The Denotation Directory
   5.1.6 Evaluation of Data Attributes
   5.1.7 The Block Epilogue
5.2 Sequential Interpretation of Statements
   5.2.1 Statement List
   5.2.2 Interpretation of a Single Statement
   5.2.3 Case Distinction of Proper Statements
5.3 If-Statement
5.4 Group
   5.4.1 While-Group
   5.4.2 Controlled Group
5.5 Goto Statement
   5.5.1 Label Searching
   5.5.2 Transfer of Control
5.6 Procedure Call
5.6.1 Call Statement, Actions before Call
5.6.2 Argument Passing
5.6.2.1 Directly Passed Arguments
5.6.2.2 Passed Argument Generations
5.6.2.3 Dummy Arguments
5.6.2.4 Installation of Arguments in the Called Procedure
5.6.3 Call of Procedure Body
5.6.4 Return Statement
5.6.5 Generic Selection

6. ALLOCATION, INITIALIZATION, FREEING OF VARIABLES

6.1 The Allocate Statement
6.1.1 Allocation of Controlled Variables
6.1.2 Allocation of Based Variables
6.1.3 Allocation in Areas
6.1.4 Storage Mapping
6.1.4.1 Storage Space Reserved on Allocation
6.1.4.2 Storage Space Reserved on Buffer Allocation
6.1.4.3 Special Properties of the Mapping Function
6.2 Initialization of Variables
6.2.1 Initialization of Controlled Variables
6.2.2 Initialization of Non-Controlled Variables
6.3 The Free Statement

7. ASSIGNMENT STATEMENT, EXPRESSION EVALUATION, REFERENCE TO VARIABLES
7.2 Evaluation of Expressions
  7.2.1 Evaluation of Scalar Expressions
  7.2.2 Special Cases of Expression Evaluation
    7.2.2.1 Evaluation of Integer Values
    7.2.2.2 Evaluation of Optional Expressions
  7.2.3 Evaluation of the Data Attributes of Expressions

7.3 Evaluation of References to Variables
  7.3.1 Evaluation of Subscripts and Name Qualification
  7.3.2 Reference to Proper Variables
  7.3.3 Reference to Defined Variables
    7.3.3.1 Isub-Defined Variables
    7.3.3.2 Correspondence and Overlay Defined Variables
  7.3.4 Reference to Based Variables
  7.3.5 Subgenerations of Generations

8. DATA, OPERATIONS AND CONVERSIONS

8.1 Values, Value Representations, Operands, and Operators
  8.1.1 A Class of Data Attributes
  8.1.2 Values
    8.1.2.1 Numeric Values
    8.1.2.2 Character Values
    8.1.2.3 Bit Values
    8.1.2.4 Matching Elementary Values and Attributes
  8.1.3 Value Representations
  8.1.4 Transition between a Value and its Representation
    8.1.4.1 Representing and Retrieving Elementary Values
    8.1.4.2 Retrieving a Scalar Value from its Representation
    8.1.4.3 Representing a Scalar Value
  8.1.5 Operands
  8.1.6 Operators
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>8.2</td>
<td>Auxiliary Definitions on Data Attributes</td>
<td>8-12</td>
</tr>
<tr>
<td>8.2.1</td>
<td>Special Data Attributes and Classes of Data Attributes</td>
<td>8-12</td>
</tr>
<tr>
<td>8.2.2</td>
<td>Implementation-Defined Limits Associated with Arithmetic Data Attributes</td>
<td>8-13</td>
</tr>
<tr>
<td>8.3</td>
<td>Evaluation of Infix Expressions</td>
<td>8-14</td>
</tr>
<tr>
<td>8.3.1</td>
<td>Value of the Result</td>
<td>8-15</td>
</tr>
<tr>
<td>8.3.1.1</td>
<td>Arithmetic Operations</td>
<td>8-15</td>
</tr>
<tr>
<td>8.3.1.2</td>
<td>Comparison Operations</td>
<td>8-18</td>
</tr>
<tr>
<td>8.3.1.3</td>
<td>Bit String Operations</td>
<td>8-21</td>
</tr>
<tr>
<td>8.3.2</td>
<td>Data Attributes of the Result</td>
<td>8-21</td>
</tr>
<tr>
<td>8.3.2.1</td>
<td>Target Attributes for Conversion of Operands</td>
<td>8-22</td>
</tr>
<tr>
<td>8.3.2.2</td>
<td>Result Attributes</td>
<td>8-24</td>
</tr>
<tr>
<td>8.3.2.3</td>
<td>The Special Case of Integer Exponentiation</td>
<td>8-26</td>
</tr>
<tr>
<td>8.4</td>
<td>Evaluation of Prefix Expressions</td>
<td>8-27</td>
</tr>
<tr>
<td>8.4.1</td>
<td>Value of the Result</td>
<td>8-28</td>
</tr>
<tr>
<td>8.4.1.1</td>
<td>Arithmetic Prefix Operations</td>
<td>8-28</td>
</tr>
<tr>
<td>8.4.1.2</td>
<td>Negation</td>
<td>8-29</td>
</tr>
<tr>
<td>8.4.2</td>
<td>Data Attributes of the Result</td>
<td>8-29</td>
</tr>
<tr>
<td>8.5</td>
<td>Conversion</td>
<td>8-30</td>
</tr>
<tr>
<td>8.5.1</td>
<td>Value Conversion</td>
<td>8-30</td>
</tr>
<tr>
<td>8.5.1.1</td>
<td>Numeric to Character Conversion</td>
<td>8-31</td>
</tr>
<tr>
<td>8.5.1.2</td>
<td>Numeric to Bit Conversion</td>
<td>8-33</td>
</tr>
<tr>
<td>8.5.1.3</td>
<td>Character to Numeric Conversion</td>
<td>8-34</td>
</tr>
<tr>
<td>8.5.1.4</td>
<td>Character to Bit Conversion</td>
<td>8-35</td>
</tr>
<tr>
<td>8.5.1.5</td>
<td>Bit to Numeric Conversion</td>
<td>8-35</td>
</tr>
<tr>
<td>8.5.1.6</td>
<td>Bit to Character Conversion</td>
<td>8-36</td>
</tr>
<tr>
<td>8.5.2</td>
<td>Completion of Target Attributes</td>
<td>8-36</td>
</tr>
<tr>
<td>8.5.2.1</td>
<td>Completion of Arithmetic Attributes</td>
<td>8-37</td>
</tr>
<tr>
<td>8.5.2.2</td>
<td>Completion of String Attributes</td>
<td>8-38</td>
</tr>
<tr>
<td>8.5.2.3</td>
<td>Transformation into an Arithmetic Attribute</td>
<td>8-38</td>
</tr>
<tr>
<td>8.5.2.4</td>
<td>Transformation into a String Attribute</td>
<td>8-38</td>
</tr>
<tr>
<td>8.6</td>
<td>Pictures</td>
<td>8-40</td>
</tr>
<tr>
<td>8.6.1</td>
<td>Auxiliary Definitions</td>
<td>8-42</td>
</tr>
<tr>
<td>8.6.1.1</td>
<td>Explicit Picture Attributes</td>
<td>8-42</td>
</tr>
<tr>
<td>8.6.1.2</td>
<td>Classification of Pictures and Picture Specifications</td>
<td>8-44</td>
</tr>
<tr>
<td>8.6.2</td>
<td>Checking of Picture Attributes</td>
<td>8-47</td>
</tr>
<tr>
<td>8.6.2.1</td>
<td>Three Classes of Binary Pictures</td>
<td>8-48</td>
</tr>
<tr>
<td>8.6.2.2</td>
<td>Proper Subfields</td>
<td>8-48</td>
</tr>
</tbody>
</table>
8.6.2.3 Additional Restrictions for Sterling Subfields
8.6.2.4 Proper Drifting
8.6.3 Representation in Pictured Form
  8.6.3.1 Representation of Numeric Pictures
  8.6.3.2 Representation of Subfields
  8.6.3.3 Composition of Subfields and Linearization
  8.6.3.4 Representation of Character Pictures
  8.6.3.5 Conversion from Numeric Picture to Character String
8.6.4 Value of a Picture
  8.6.4.1 Value of a Numeric Picture
  8.6.4.2 Testing of String Values against Picture Attributes
8.6.5 Transformation of Picture Attributes

9. CONDITIONS
  9.1 Condition Enabling
  9.2 Condition Action
    9.2.1 Standard System Action for Conditions
    9.2.2 Interpretation of On-Statement
    9.2.3 Interpretation of Revert Statement
  9.3 Condition Builtin Function Status
  9.4 Condition Activation
    9.4.1 Interpretation of the Condition Call
    9.4.2 Interpretation of Condition Action
    9.4.3 The Signal Statement
    9.4.4 Special Condition Activations
      9.4.4.1 Activation of Check Condition
      9.4.4.2 Activation of Conversion Condition

10. INPUT AND OUTPUT
  10.1 Attributes of a File
  10.2 Access to the External Storage
    10.2.1 Access to a Data Set by a Title
    10.2.2 Basic Accessing Instructions
  10.3 Opening and Closing of a File
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
</tr>
</thead>
<tbody>
<tr>
<td>10.3.1</td>
<td>Explicit Opening</td>
</tr>
<tr>
<td>10.3.2</td>
<td>Implicit Opening</td>
</tr>
<tr>
<td>10.3.2.1</td>
<td>Opening with a File Identifier</td>
</tr>
<tr>
<td>10.3.2.2</td>
<td>Opening of the Standard System Print File</td>
</tr>
<tr>
<td>10.3.3</td>
<td>Central Part of Opening</td>
</tr>
<tr>
<td>10.3.4</td>
<td>Closing</td>
</tr>
<tr>
<td>10.4</td>
<td>Record Input and Output</td>
</tr>
<tr>
<td>10.4.1</td>
<td>Positioning of Record Data Sets</td>
</tr>
<tr>
<td>10.4.2</td>
<td>Unbuffered Record Input and Output</td>
</tr>
<tr>
<td>10.4.3</td>
<td>Buffered Record Input and Output</td>
</tr>
<tr>
<td>10.4.4</td>
<td>Record Input and Output On-Conditions</td>
</tr>
<tr>
<td>10.4.5</td>
<td>Input and Output Events</td>
</tr>
<tr>
<td>10.4.6</td>
<td>Exclusive Files and the Unlock Statement</td>
</tr>
<tr>
<td>10.4.7</td>
<td>Record Input and Output Statements</td>
</tr>
<tr>
<td>10.4.7.1</td>
<td>The Write Statement</td>
</tr>
<tr>
<td>10.4.7.2</td>
<td>The Rewrite Statement</td>
</tr>
<tr>
<td>10.4.7.3</td>
<td>The Locate Statement</td>
</tr>
<tr>
<td>10.4.7.4</td>
<td>Read (into) Statement</td>
</tr>
<tr>
<td>10.4.7.5</td>
<td>Read (set) Statement</td>
</tr>
<tr>
<td>10.4.7.6</td>
<td>Read (ignore) Statement</td>
</tr>
<tr>
<td>10.4.7.7</td>
<td>The Delete Statement</td>
</tr>
<tr>
<td>10.5</td>
<td>Stream Transmission</td>
</tr>
<tr>
<td>10.5.1</td>
<td>Data Specifications</td>
</tr>
<tr>
<td>10.5.1.1</td>
<td>Iterated Data Lists</td>
</tr>
<tr>
<td>10.5.1.2</td>
<td>Expansion of Format Lists</td>
</tr>
<tr>
<td>10.5.2</td>
<td>Data Fields</td>
</tr>
<tr>
<td>10.5.2.1</td>
<td>Abstract Syntax of Constants</td>
</tr>
<tr>
<td>10.5.2.2</td>
<td>Abstract Syntax of Names</td>
</tr>
<tr>
<td>10.5.2.3</td>
<td>Function and Predicate Schemes</td>
</tr>
<tr>
<td>10.5.2.4</td>
<td>Arithmetic Data Fields</td>
</tr>
<tr>
<td>10.5.3</td>
<td>Get Statement</td>
</tr>
<tr>
<td>10.5.3.1</td>
<td>Control Oriented Part</td>
</tr>
<tr>
<td>10.5.3.2</td>
<td>Elementary Transmission</td>
</tr>
<tr>
<td>10.5.3.3</td>
<td>List-Directed Transmission</td>
</tr>
<tr>
<td>10.5.3.4</td>
<td>Data-Directed Transmission</td>
</tr>
<tr>
<td>10.5.3.5</td>
<td>Edit-Directed Transmission</td>
</tr>
<tr>
<td>10.5.3.6</td>
<td>Copy Action</td>
</tr>
</tbody>
</table>
10.5.4 Put Statement
10.5.4.1 Control Oriented Part
10.5.4.2 Elementary Transmission
10.5.4.3 List-Directed Transmission
10.5.4.4 Data-Directed Transmission
10.5.4.5 Edit-Directed Transmission
10.5.4.6 Check Standard System Action

10.6 Display Statement
10.7 Auxiliary Definitions

11. BUILTIN FUNCTIONS

11.1 Syntactic Expansion of Builtin Function References
11.2 Evaluation of Builtin Function References
11.3 The Data Attributes of Builtin Function References
11.4 Builtin Functions Passed to Entry Parameters
11.5 Table of Builtin Functions
  11.5.1 Explanation
    11.5.1.1 The Argument Columns id and i
    11.5.1.2 Number of Arguments of Builtin Functions
    11.5.1.3 The Classes of Arguments of Builtin Functions
    11.5.1.4 The Types of Arguments of Builtin Functions
    11.5.1.5 Conversion of Arguments of Builtin Functions
    11.5.1.6 Testing of Arguments of Builtin Functions
    11.5.1.7 Return Types of Builtin Functions
    11.5.1.8 Evaluation of the Individual Builtin Functions
  11.5.2 The Table

11.6 Evaluation of the Individual Builtin Functions
  11.6.1 General Auxiliary Definitions
  11.6.2 Evaluating Instructions
    11.6.2.1 Arithmetic Generic Builtin Functions
    11.6.2.2 Mathematical Generic Builtin Functions
    11.6.2.3 String Generic Builtin Functions
    11.6.2.4 Generic Builtin Functions for Array Manipulation
    11.6.2.5 Condition Builtin Functions
    11.6.2.6 List-Processing Builtin Functions
    11.6.2.7 Other Builtin Functions
12. ABSTRACT SYNTAX OF TEXT

12.1 Notations
12.2 Predicate Definitions. Program
  12.2.1 Declarations
  12.2.1.1 Data Attributes
  12.2.1.2 Pictures
  12.2.1.3 Formats
  12.2.2 Statements
  12.2.2.1 Block, Group
  12.2.2.2 Flow of Control Statements
  12.2.2.3 Storage Manipulating Statements
  12.2.2.4 Condition Handling Statements
  12.2.2.5 Input and Output Statements
  12.2.3 Expressions
  12.2.3.1 References
  12.2.3.2 Constants
12.3 Auxiliary Functions and Predicates on Data Attributes
  12.3.1 Evaluated Data Attributes
  12.3.2 Summarizing Predicates
  12.3.3 Functions on Data Attributes

APPENDIX I: CROSS REFERENCE INDEX
APPENDIX II: LIST OF ELEMENTARY OBJECTS
1. NOTATION AND CONVENTIONS

This chapter presents the notation and conventions adopted for the formal definition of the PL/I-Interpreter. 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, notation and meaning of definitions of functions and instructions is described. Next, the conventions around treatment of program errors are discussed, 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.

In several cases, where the explanation of the notation deserves a more lengthy and carefully written exposition, reference is made to the Introduction /1/.

1.1 Notation of Meta-Expressions

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). In some other cases, reference to /1/ is made (e.g. for the \( \mu \)-function).

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:

\[
\begin{align*}
P_1 & \rightarrow e_1 \\
P_2 & \rightarrow e_2 \\
& \vdots \\
P_n & \rightarrow e_n
\end{align*}
\]

A conditional expression denotes the value of \( e_i \) where \( i \) is the integer \( i = n \), for which \( p_i \) is true and all preceding \( p_j \), \( j < i \), 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 values of the successors of \( p_i \), say \( p_{i+1}, \ldots, p_n \), are irrelevant for the evaluation of the conditional expression and may even be undefined.

1.1.2 Functional composition

The operator is defined by:

\[
(f \circ g)(x_1, \ldots, x_n) = f(g(x_1, \ldots, x_n))
\]

\( f \) 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 \( \circ \) binds stronger than functional application, e.g.:

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

2. \( f_1 \circ f_2 \circ \cdots \circ f_m(x_1, \ldots, x_n) = f_1(f_2(\cdots (f_m(x_1, \ldots, x_n) \cdots))
\]

3. \( f(x_1)(x_2) \cdots (x_n) = (\cdots ((f(x_1))(x_2)) \cdots)(x_n)\)
1.1.3 Objects

The definition of the general class of objects is given in /1/. The class of objects used in the present document is characterized in the following by the class of selectors and the class of elementary objects, used for the construction of objects.

The following selectors are used:

(1) strings of small letters, digits and hyphens, prefixed by s-;
(2) the range of the following functions

<table>
<thead>
<tr>
<th>Selector</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sel(idl)</td>
<td>is-id-list(idl)</td>
</tr>
<tr>
<td>elem(i)</td>
<td>is-intg-val(i)</td>
</tr>
<tr>
<td>sub(i)</td>
<td>is-intg-val(i)</td>
</tr>
<tr>
<td>s(i)</td>
<td>is-intg-val(i)</td>
</tr>
</tbody>
</table>

where the ranges of these one-to-one functions have no elements in common, and do not contain selectors mentioned under (1), (3), and (4);

(3) identifiers, i.e. the set of elements satisfying the predicate is-id;
(4) unique names, i.e. the set of elements satisfying the predicate is-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.

The following elementary objects are used:

(1) the objects listed in Appendix 2;
(2) numbers, satisfying the predicate is-num-val, represented as decimal numbers;
(3) character values, satisfying the predicate is-char-val, but not mentioned under (1) (extralingual characters);
(4) composite selectors;
(5) identifiers, satisfying the predicate is-id;
(6) unique names, satisfying the predicate is-n;
(7) value representations, satisfying the predicate is-vr;
sizes of storage, satisfying the predicate is-size;
pointer values, satisfying the predicate is-ptr-val;
finite sets of objects.

The class of objects used in the document comprises the null-object \( \emptyset \).

1.1.4 The \( \mu \)-function

For the definition of the meaning of the basic form of the \( \mu \)-function \( \mu(A; \langle \chi; B \rangle) \) reference is made to /1/. The following extended forms of the \( \mu \)-function and the \( \delta \)-function are used in the present document:

(1) \[ \mu(A; \langle \chi_1; B_1 \rangle, \langle \chi_2; B_2 \rangle, \ldots, \langle \chi_n; B_n \rangle) \]

The above form is defined iteratively by the equation:

\[ \mu(A; \langle \chi_1; B_1 \rangle, \langle \chi_2; B_2 \rangle, \ldots, \langle \chi_n; B_n \rangle) = \mu(\mu(A; \langle \chi_1; B_1 \rangle); \langle \chi_2; B_2 \rangle, \ldots, \langle \chi_n; B_n \rangle) \]

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

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

(2) \[ \mu(A; \{ \langle \chi; B \rangle \mid p(\chi, B) \}) \]

The second argument of the above form defines a set of pairs \( \langle \chi; B \rangle \), namely the set of pairs for which a certain proposition \( p(\chi, B) \) 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:

\[ \mu(A; \{ \}) = A \]

(3) \[ \mu_0(\langle \chi_1; A_1 \rangle, \langle \chi_2; A_2 \rangle, \ldots, \langle \chi_n; A_n \rangle) \]

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

\[ \mu_0(\langle \chi_1; A_1 \rangle, \langle \chi_2; A_2 \rangle, \ldots, \langle \chi_n; A_n \rangle) = \mu(\emptyset; \langle \chi_1; A_1 \rangle, \langle \chi_2; A_2 \rangle, \ldots, \langle \chi_n; A_n \rangle) \]
1.1.5 Manipulation of lists

Lists are special classes 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 is-list holds for any list, it is defined as

\[
is\text{-}list(L) = \]

\[
(\exists x_1, \ldots, x_n) (L = \mu_o(\{<x_1:\lambda \mid p(x,A)> | p(x,A)\}) \land \
\neg is-Q(x_1) \land \neg is-Q(x_2) \land \ldots \land \neg is-Q(x_n))
\]

An abbreviation for denoting elements of a list is introduced. Let \( L \) be a list, then

\[
elem(i, L) = elem(i)(L).
\]
The length of a list is defined as the largest index of an element which is not the null object.

(7) \[
\text{length}(L) = \\
is-<>(L) \rightarrow 0 \\
is-\text{list}(L) \rightarrow (i,i) (\neg is-\Omega \text{elem}(i,L) \& is-\Omega \text{elem}(i+1,L))
\]

The following three 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) and the tail of a list (which is the original list except the first element).

(8) \[
\text{head}(L) = \\
is-\text{list}(L) \& \neg is-<>(L) \rightarrow \text{elem}(1,L)
\]

(9) \[
\text{last}(L) = \\
is-\text{list}(L) \& \neg is-<>(L) \rightarrow \text{elem}(<\text{length}(L),L)
\]

(10) \[
\text{tail}(L) = \\
\text{length}(L) = 1 \rightarrow <> \\
\text{length}(L) > 1 \rightarrow \mu_0(\{<\text{elem}(i);\text{elem}(i+1,L)> \mid 1 \leq i \leq \text{length}(L)-1\})
\]

The concatenation of two lists is defined by:

\[
L_1 \circ L_2 = is-\text{list}(L_1) \& is-\text{list}(L_2) \rightarrow \\
\mu(L_1;\{<\text{elem}(\text{length}(L_1)+i);\text{elem}(i,L_2)> \mid 1 \leq i \leq \text{length}(L_2)\})
\]

Note that \(L \circ <> = <> \circ L = L\) for any list \(L\).

Multiple concatenation is defined by:

\[
\text{CONC}_{i=1}^n L_i = (n<1 \rightarrow <> , n=1 \rightarrow L_1,T \rightarrow L_1 \circ L_2 \circ ... \circ L_n)
\]

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

\[
< A_1 , A_2 , ..., A_n > = \mu_0( < \text{elem}(1);A_1 > , < \text{elem}(2);A_2 > , ..., < \text{elem}(n);A_n >)
\]

for: \(n \geq 1, A_i \neq \emptyset \text{ is } n\)
An alternative form is:

\[
\text{LIST} \quad A_i = (n \leftarrow \langle \rangle, T \leftarrow \langle A_1, A_2, \ldots, A_n \rangle)
\]

1.1.6 Equality

\[
\begin{align*}
= & \quad \text{equal} \\
\neq & \quad \text{not equal}
\end{align*}
\]

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

1.1.7 Truth values, logical operators and quantifiers

1.1.7.1 Truth values

\[
\begin{align*}
T & \quad \text{true} \\
F & \quad \text{false}
\end{align*}
\]

1.1.7.2 Logical operators

\[
\begin{align*}
\neg & \quad \text{not} \\
& \quad \text{equivalence} \\
& \quad \text{non equivalence} \\
& \quad \text{implication}
\end{align*}
\]

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:

\[
\begin{align*}
(p_1 \land p_2) &= \\
(\neg p_1 \leftarrow F, T \leftarrow p_2) \\
(p_1 \lor p_2) &= \\
(p_1 \leftarrow T, T \leftarrow p_2)
\end{align*}
\]
\((p_1 \equiv p_2) = (p_1 \land p_2) \lor (\neg p_1 \land \neg p_2)\)

\((p_1 \neq p_2) = \neg(p_1 \equiv p_2)\)

\((p_1 \supset p_2) = (\neg p_1 \rightarrow T, T \rightarrow p_2)\)

The following rules for omission of parentheses hold for expressions built from the implication operator:

\(p_1 \supset p_2 \supset \ldots \supset p_{n-1} \supset p_n = (p_1 \supset (p_2 \supset \ldots (p_{n-1} \supset p_n) \ldots))\)

\(\bigwedge_{i=1}^{n} p_i = (n < 1 \rightarrow T, n = 1 \rightarrow p_1, T \rightarrow p_1 \land p_2 \land \ldots \land p_n)\)

\(\bigvee_{x \in X} p_x = (\{s - \{\} (x) \rightarrow T, T \rightarrow (\forall y)(\exists x_1, \ldots, x_n) (x = \{x_1, \ldots, x_n\} \land y = \bigvee_{i=1}^{n} p_{x_i})\))\)

Note that \(\bigvee_{x \in X} p_x\) is defined only if \(\bigvee_{i=1}^{n} p_{x_i}\) is independent of the ordering of \(x_1, \ldots, x_n\).

1.1.7.3 Quantifiers

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

The above symbols will be used in expressions of the following forms:

\((\exists x_1, x_2, \ldots, x_n) (p(x_1, x_2, \ldots, x_n))\)
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 $p(x_1, x_2, \ldots, x_n)$ is true, otherwise the expression is false.

$$\forall x_1, x_2, \ldots, x_n (p(x_1, x_2, \ldots, x_n))$$

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) $p(x_1, x_2, \ldots, x_n)$ 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.

### 1.1.7.4 Description

**\iota\text{-}operator**

The symbol will be used in expressions of the following form:

$$(\iota x)(p(x))$$

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

### 1.1.8 Arithmetic operators and relations

Arithmetic operators and relations are defined for entities satisfying the predicate is-num-val, see chapter 8.

#### 1.1.8.1 Operators

+ prefix plus, infix plus
- prefix minus, infix minus
, multiplication
/ division
↑ exponentiation
1.1.8.2 Relations

- <  less
- \leq  less or equal
- =  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 \( e_1 \) is an arithmetic expression and \( R_1 \) is one of the above relational operators. This form has the meaning:

\[ (e_1 R_1 e_2) \land (e_2 R_2 e_3) \]

A meta-expression of the form

\[ e_1 R_1 e_1', e_1'' R_2 e_3 \]

has the meaning

\[ (e_1 R_1 e_1' R_2 e_3) \land (e_1 R_1 e_1'' R_2 e_3) \]

1.1.8.3 Arithmetic functions

In this section satisfy:

- \( x, y \)  is-real-val
- \( z \)  is-num-val
- \( n, m \)  is-intg-val
(11) \(\text{abs}(x) = (x \geq 0 \rightarrow x, T \rightarrow -x)\)

(12) \(\text{sign}(x) = (x \geq 0 \rightarrow 1, T \rightarrow -1)\)

(13) \(\text{modulo}(x, y) =\)
\[
(\exists m)(\text{is-intg-val}(m) \& x = m \cdot y + r \& 0 \leq r < y)
\]
for: \(y > 0\)
The function gives the remainder if \(x\) is divided by \(y\).

(14) \(\text{ceil}(x) =\)
\[
(\exists n)(\text{is-intg-val}(n) \& n-1 < x \leq n)
\]
The function gives the smallest integer not exceeded by \(x\).

(15) \(\text{floor}(x) =\)
\[
(\exists n)(\text{is-intg-val}(n) \& n \leq x < n+1)
\]
The function gives the greatest integer not exceeding \(x\).

(16) \(\text{trunc}(x) = (x \geq 0 \rightarrow \text{floor}(x), T \rightarrow \text{ceil}(x))\)

(17) \(\text{max}(x, y) = (x \geq y \rightarrow x, T \rightarrow y)\)

(18) \(\text{min}(x, y) = (x \leq y \rightarrow x, T \rightarrow y)\)

(19) \(\text{cplx}(x, y) = x + iy\)

(20) \(\text{real}(z)\) gives the real part of the complex number \(z\).

(21) \(\text{imag}(z)\) gives the imaginary part of the complex number \(z\).

1.1.9 Set operators, relations and notation for sets

1.1.9.1 Set operators

\(\cup\) union
\(\cap\) intersection
\(-\) difference

\[\bigcup_{i=1}^{n} \text{set}_i = (n < 1 \rightarrow \{} \), \(n = 1 \rightarrow \text{set}_1, T \rightarrow \text{set}_1 \cup \text{set}_2 \cup ... \cup \text{set}_n\)\]
1.1.9.2 Relations

\[ \epsilon \]  is element of
\[ \# \]  is not element of
\[ \subset \]  is proper subset
\[ \subseteq \]  is subset or equal

1.1.9.3 Notation for sets

\{a,b,c,...\}  the elements a,b,c,... are the elements of the set
\{\}  the empty set
\{e(x_1,...,x_n) \mid p(x_1,...,x_n)\}  implicit definition of a set

where \( e(x_1,...,x_n) \) is an expression, depending on the n-tuple \( x_1,...,x_n \). This notation defines the set of the values of this expression for all n-tuples \( x_1,...,x_n \) for which the predicate \( p(x_1,...,x_n) \) is true.

1.1.10 Rules of precedence

Parentheses may be omitted according to the following rules of precedence:

\[ \uparrow, \text{prefix +, prefix -} \]
\[ \cdot, / \]
\[ \cup, \cap, \land, \lor, \text{infix +, infix -} \]
\[ <, \leq, \neq, \geq, \text{infix +, infix -} \]
\[ \& \]
\[ \lor \]
\[ \equiv, \neq, \subseteq \]

highest precedence:
(binds strongest)

lowest precedence
(binds weakest)
1.1.11 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 $\Omega$, $\ast$, $\langle$, $\rangle$, $\{$, and numerical values).
4) 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 12 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 subchapter 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 'Abbreviations' 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:

\[ fn(x_1, x_2, \ldots, x_n) = \text{cond-expr}(x_1, x_2, \ldots, x_n) \]

where \( fn \) is a function name,
\( x_1 \) are metavariables,
\( \text{cond-expr} \) is a conditional expression in the metavariables \( x_1 \).
The special case

$$fn(x_1,x_2,\ldots,x_n) =
\begin{array}{c}
T \longrightarrow e(x_1,x_2,\ldots,x_n)
\end{array}$$

may be written as

$$fn(x_1,x_2,\ldots,x_n) =
\begin{array}{c}
e(x_1,x_2,\ldots,x_n)
\end{array}$$

A conditional expression is an expression formed from the notational elements described in chapter 1.1.

### 1.2.3.2 Instruction definitions

Instruction definitions are described in detail in chapter 4 of /1/. Only the essentials of the notation and meaning are repeated here.

An instruction definition has the form:

$$\text{instr-name}(x_1,x_2,\ldots,x_n) =
\begin{array}{c}
p_1(x_1,x_2,\ldots,x_n) \rightarrow \text{group}_1(x_1,x_2,\ldots,x_n)
p_2(x_1,x_2,\ldots,x_n) \rightarrow \text{group}_2(x_1,x_2,\ldots,x_n)
\vdots
p_m(x_1,x_2,\ldots,x_n) \rightarrow \text{group}_m(x_1,x_2,\ldots,x_n)
\end{array}$$

where $p_i(x_1,x_2,\ldots,x_n)$ are propositional expressions and $\text{group}_i(x_1,x_2,\ldots,x_n)$ define a value and a state transformation (for cases where the instruction is value returning) or a control-tree (for cases where the instruction is self-replacing).

The special case:

$$\text{instr-name}(x_1,x_2,\ldots,x_n) =
\begin{array}{c}
T \rightarrow \text{group}(x_1,x_2,\ldots,x_n)
\end{array}$$

may also be written:
The $x_i$ are metavariables, $\xi$ is the state of the PL/I machine. Groups for the value returning alternatives of instruction definitions have the format:

\[
\begin{align*}
\text{PASS: } & e_0(x_1, x_2, \ldots, x_n, \xi) \\
\text{s-sc}_1 & : e_1(x_1, x_2, \ldots, x_n, \xi) \\
\text{s-sc}_2 & : e_2(x_1, x_2, \ldots, x_n, \xi) \\
& \text{etc.} \\
\text{s-sc}_r & : e_r(x_1, x_2, \ldots, x_n, \xi)
\end{align*}
\]

where $e_i$ are meta-expressions in $x_1, x_2, \ldots, x_n$, and $\xi$, which for any specific value assignment to these variables denote objects (or are undefined). The meta-expressions may also involve control-representations in their fully parenthesized form (see next page).

The selectors $s-sc_i$ are selectors pointing to the immediate components of the state as listed in chapter 2. Not all components need be referred to in a group. The first line may be missing, which by convention is equivalent to PASS: $\emptyset$.

Groups for self-replacing alternatives of instruction definitions are control representations. A control representation $c$-rep is either a single instruction

\[
i \text{instr}
\]

or an instruction (the instruction at the top node of the represented control tree) followed by a semi-colon and a set of successors. The set of successors may be given explicitly, or by implicit set notation (cf. 1.1.9.3) or by the union of sets of successors.

An instruction $\text{instr}$ consists of an instruction name and a list of arguments:

\[
\text{instr-name}(a_1, \ldots, a_p)
\]

An instruction name is a string of small letters, digits, and hyphens, which is underlined. An argument $a_i$ is either a meta-expression in $x_1, x_2, \ldots, x_n$ and $\xi$, or a dummy name.

1.2.3.2
A successor $\text{succ}_i$ is a control representation (the representation of the $i$-th successor control tree) or a prefix followed by a colon followed by a control representation:

prefix:c-rep

A prefix has the form:

$\chi$(dummy-name)

or the form:

dummy-name

where $\chi$ is a composite selector and the dummy-name is a string of small letters, digits, and hyphens.

A control representation of the form

$\text{instr}\{\text{succ}_1, \text{succ}_2, \ldots, \text{succ}_n\}$

usually is written in the alternative form, using indentation and omitting brackets:

$\text{instr;}
\text{succ}_1,
\text{succ}_2,
\vdots
\text{succ}_n$

The meaning of an instruction definition is as follows (for a more detailed exposition see /1/). If the instruction $\text{instr-name}(\text{arg}_1, \text{arg}_2, \ldots, \text{arg}_n)$ in the control $\mathcal{C}$ of the PL/I machine in state $\xi_1$ is to be executed, the $\text{arg}_i$ being objects, the list of propositions $p_j(\text{arg}_1, \text{arg}_2, \ldots, \text{arg}_n, \xi_1')$ are inspected from top to bottom, where $\xi_1'$ is the state $\xi_1$ with the instruction to be executed being deleted. If $p_{k,1}$ is the first proposition denoting $T$, then group $k$ is executed. If there is no such $k$, then the instruction is undefined.

If group $k$ specifies a value returning instruction the following actions are performed in the specified order:

1.2.3.2
a) Substitution of the value of the instruction into the appropriate argument places in the control. The value returned by the instruction is that denoted by the meta-expression $e_0(\text{arg}_1, \text{arg}_2, \ldots, \text{arg}_n, \xi')$ following the word PASS in the group. If the instruction has been prefixed by dummy-name then this value is inserted in all argument-places that had been specified with dummy-name. If the instruction has been prefixed by $\chi(\text{dummy-name})$ then the value is inserted in the $\chi$-parts of all argument places in the control that had been specified with dummy-name.

b) The so modified state $\xi''$ is then transformed into the successor state $\xi_{i+1}$ by

$$
\xi_{i+1} = \mu(\xi'', <s-sc_1:e_1(\text{arg}_1, \text{arg}_2, \ldots, \text{arg}_n, \xi')>, <s-sc_2:e_2(\text{arg}_1, \text{arg}_2, \ldots, \text{arg}_n, \xi')>, \ldots, <s-sc_r:e_r(\text{arg}_1, \text{arg}_2, \ldots, \text{arg}_n, \xi')>)
$$

If group $k$ specifies a self-replacing instruction, the following action is performed:

The control tree specified by the group is inserted in the control $\xi$ in the place of the instruction being executed. This control tree is obtained by replacing the argument expressions of the instructions specified in the control representation by the values they denote for $x_i = \text{arg}_i$, $1 \leq i \leq n$, and the state $\xi'$.

If the place in the control which contained the executed instruction was prefixed, this prefix is retained. This means that the replacement of an instruction in the control does not destroy the information where to return the value, which is defined by the prefix.

In case of any conflict between dummy names in the control and dummy names in the control tree replacing the instruction, suitable changes of the dummy names in the replacing control trees are effected before the actual substitution. This guarantees that dummy names have a completely local meaning within the control tree where they are specified.

The actions performed on execution of an instruction which are independent of the specific instruction are described in chapter 2.1.
1.2.3.3 Predicate definitions

Notation for the definition of predicates is introduced in chapter 12.1.

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 12 (Abstract Syntax) are not referenced.
3) Names of elementary objects are not referenced.
4) All names defined in the same chapter or in one of the sub-chapters are not referenced.
1.3 The Treatment of Program Errors

The function $\bigwedge (\bar{c})$, defining the set of successor states for a given state $\bar{c}$ of the PL/I-Machine (see chapter 2.1) is a partial function, i.e. there are states for which no successor states are defined. If the computations arrive at a state which has no defined successor state, then the program which led to that situation is in error.

A successor state of a certain state is determined by the definition of one of the instructions which are candidate for execution in that state (see chapter 2.1), i.e. a successor state is undefined if the relevant instruction is undefined for the state. For indicating that this situation occurs the instruction

(22) error

and the function

(23) error

are used. An instruction is undefined for a given state:

1) if when inspecting the propositions $p_i$ in the instruction definition from top to bottom (see chapter 1.2.3.2), 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) if 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 error.

3) if 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

1) if it is a function undefined in that instance,

2) if one of its sub-expressions is undefined,

3) 'a case it has the form $(x)(e(x))$, if $e(x)$ denotes T for no $x$ or for more than one $x$.

A function is undefined in a given instance

1) if it denotes the function error in that instance,

2) if one of its arguments is undefined,
3) if it is defined by a meta-expression which is undefined,
4) if the function is $\varnothing$ (applied to any argument),
5) if the function is one of the functions head, tail, and length, applied to non-lists.

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 these functions and instructions are specified in the definition as far as they are known.

1.4 Generally used simple Instructions

(24) **null**

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

(25) **pass(e)** =

$$\text{PASS}: e$$

for any meta-expression e.

(26) **pass- [function] (e_1, \ldots, e_n)** =

$$\text{PASS}: [\text{function}](e_1, \ldots, e_n)$$

A special convention has been adopted for the meaning of instructions of the form **pass- [function]**, where [function] stands for any function name. An instruction of this form returns the value of the named function applied to the arguments of the instruction.

(27) **mk-list(x, list)** =

$$\text{PASS}: <x> \sim \text{list}$$

(28) **mk-list-l(list, x)** =

$$\text{PASS}: \text{list} \sim <x>$$
2. STATE COMPONENTS AND COMPUTATION OF THE PL/I MACHINE

The PL/I machine is an abstract sequential machine as described in chapter 4 of the "Method and Notation for the Formal Definition of Programming Languages"/1/.

The PL/I machine defines PL/I by interpreting abstract text, a modification of program text. Abstract text conforms with the rules of the Abstract Syntax of PL/I (cf. 12.2).

The PL/I machine is described by the set of all possible states which the machine can assume. This set is defined by the Language Function, which applied to a given state yields a set of successor states, and by the set of possible initial states of the machine. An initial state of the machine contains the abstract text to be interpreted. Any state from the set of possible states satisfies the predicate is-state.

(1) is-state = (<s-s:is-mstg>,
<s-es:is-es>,
<s-un:is-un>,
<s-dn:is-dn>,
<s-at:is-at>,
<s-fu:is-fu>,
<s-td:is-td>,
<s-et:is-et>,
<s-m:is-m>,
<s-pa:is-pa>,
<s-te:is-te v is-Ø>,
<s-ag:is-ag>,
<s-fd:is-fd>,
<s-e:is-e>,
<s-ei:is-ei v is-Ø>,
<s-cs:is-cs v is-Ø>,
<s-d:is-d>,
<s-ci:is-ci>,
<s-c:is-c>)
Abbreviations and Terms

For the convenience of reference to parts of the state abbreviations for major components of a given state $\xi$ are introduced and will be used throughout the document. The terms given name the major state parts according to their contents and use.

<table>
<thead>
<tr>
<th>Component</th>
<th>Abbreviation</th>
<th>Term</th>
</tr>
</thead>
<tbody>
<tr>
<td>s-s($\xi$)</td>
<td>S</td>
<td>Internal Storage</td>
</tr>
<tr>
<td>s-es($\xi$)</td>
<td>ES</td>
<td>External Storage</td>
</tr>
<tr>
<td>s-un($\xi$)</td>
<td>UN</td>
<td>Unique Name Counter</td>
</tr>
<tr>
<td>s-dn($\xi$)</td>
<td>DN</td>
<td>Denotation Directory</td>
</tr>
<tr>
<td>s-at($\xi$)</td>
<td>AT</td>
<td>Attribute Directory</td>
</tr>
<tr>
<td>s-fu($\xi$)</td>
<td>FU</td>
<td>File Union Directory</td>
</tr>
<tr>
<td>s-td($\xi$)</td>
<td>TD</td>
<td>Time and Date Part</td>
</tr>
<tr>
<td>s-m($\xi$)</td>
<td>M</td>
<td>Message Part</td>
</tr>
<tr>
<td>s-et($\xi$)</td>
<td>ET</td>
<td>Event Trace</td>
</tr>
<tr>
<td>s-pa($\xi$)</td>
<td>PA</td>
<td>Parallel Task and Event Part</td>
</tr>
<tr>
<td>s-te($\xi$)</td>
<td>TE</td>
<td>Task and Event Specification</td>
</tr>
<tr>
<td>s-ag($\xi$)</td>
<td>AG</td>
<td>Aggregate Directory</td>
</tr>
<tr>
<td>s-fd($\xi$)</td>
<td>FD</td>
<td>File Directory</td>
</tr>
<tr>
<td>s-e($\xi$)</td>
<td>E</td>
<td>Environment</td>
</tr>
<tr>
<td>s-ei($\xi$)</td>
<td>EI</td>
<td>Epilogue Information</td>
</tr>
<tr>
<td>s-cs($\xi$)</td>
<td>CS</td>
<td>Condition Status</td>
</tr>
<tr>
<td>s-d($\xi$)</td>
<td>D</td>
<td>Dump</td>
</tr>
<tr>
<td>s-ci($\xi$)</td>
<td>CI</td>
<td>Control Information</td>
</tr>
<tr>
<td>s-c($\xi$)</td>
<td>C</td>
<td>Control</td>
</tr>
</tbody>
</table>

2.1 The Computation of the PL/I Machine

By iterated application of the Language Function to a given initial state containing a given abstract text, and to the successor states, the PL/I machine passes through a sequence of states. This sequence of states which the machine assumes is the computation of the given text on the PL/I machine (chapter 4 of /1/).

The control component $C$ of a state of the machine governs the transition from the state to its successor state. The control $C$ contains instructions which on execution change the state as defined for the individual instructions in their definitions.
There exist several places in PL/I where the sequence of operations is relevant, but not specified by the language. Any choice of sequence for sequential execution is valid though it may result in different computations for different choices. If a choice of sequence is permitted, the parallel execution of operations still is excluded. Examples are the evaluation of operands in expressions or the evaluation of expressions in options.

Furthermore the language provides means to specify the legality of the parallel execution of parts of a program. Procedures can be called with one of the options specifying the creation of a task which can be executed in parallel to the calling part of the program. I/O statements can possess an event option specifying the potential parallel data transmission; this parallel action is called an "input or output event" (or shortly an "I/O event") in the following. For tasks the language specifies means for establishing and changing relative priorities. The language does not specify by which algorithm the priority values are evaluated and a decision on the selection of a highest priority task is made. The language would also permit the parallel execution of several high priority tasks. The language specifies that in most cases the sharing of storage by parallel tasks or I/O events which are not synchronized by wait statements, can result in undefined states. Only the sharing of storage occupied by event variables never can lead to an undefined situation (cf., 2.1.7).

The effects of the actual parallel execution of program parts are not described by the language. Without loss of description precision for the defined situations of the language, a sequentialized execution can be assumed. The computation of the PL/I machine is so defined that only one instruction is executed at any instant in the interpretation of a program. This instruction is taken from the task selected by the priority evaluation, and from the set of candidates for immediate execution of the control part of the selected task.

2.1.1 The parallel tasks and input and output event part PA

The major state component $s$-pa($\xi$) contains at any instant during a computation all those tasks or I/O events which could be executed in parallel. Each task or I/O event is connected by a selector which either is the selector $s$-main for the first external procedure of a program, or a unique name (cf., 2.3.1) for any other parallel action.
is-pa = (\{\text{ten:is-pt v is-pe}\} || \text{is-s-main(ten) v is-n(ten)})

Ref.: is-n 2-17(35)

is-s-main(ten) =
\text{ten = s-main}

A parallel task in \text{PA} satisfying the predicate is-pt is composed of all those parts which contain information relevant for the interpretation of this task only. All parts of a task have the same structure as the corresponding parts of the state itself.

is-pt = (\langle \text{st:is-te} \rangle, \langle \text{ag:is-ag} \rangle, \langle \text{fd:is-fd} \rangle, \langle \text{e:is-e} \rangle, \langle \text{ei:is-ei} \rangle, \langle \text{cs:is-cs} \rangle, \langle \text{d:is-d} \rangle, \langle \text{ci:is-ci} \rangle, \langle \text{c:is-c} \rangle)

Ref.: is-te 2-5(6), is-ag 2-27(94), is-fd 2-34(109), is-e 2-18(37), is-ei 2-15(31), is-cs 2-31(102), is-d 2-15(30), is-ci 2-16(32), is-c 2-7(11)

A parallel I/O event in \text{PA} satisfying the predicate is-pe contains the parts relevant for the execution of those actions of an event which can be executed in parallel to other activities of the program interpretation.

is-pe = (\langle \text{te:is-te} \rangle, \langle \text{c:is-c} \rangle)

Ref.: is-te 2-5(6), is-c 2-7(11)

2.1.2 The task and event specification \text{TE}

The parts of the state satisfying the predicate is-te, i.e. the major part \text{TE} and all te-parts of tasks or I/O events contain information relevant for the parallel execution. Some components of \text{TE} are relevant only for tasks, and some only for I/O events.
The first six components (in the above order) are relevant for a task; the first, the fourth and the last three components are relevant for an I/O event. In both cases the non-relevant components are $\Omega$. A unique distinction between a task and an I/O event is given by the fact, whether the third component, $s$-$tv$, is a generation of a task variable or $\Omega$. The components of a task-event specification are, in the above order:

1) A task-event name: In the case of a task that one selecting the task from $PA$ (cf. 2.1.1), in the case of an I/O event that of its invoking task.

2) The wait state flag of the task: $\Omega$ if the task is active, i.e., may be selected by the priority scheduler pri-sched (cf. 2.1.6); $T$ if the task is waiting, i.e., may currently not be selected by pri-sched. I/O events are always active.

3) The generation of the task variable associated with the task. The storage of this generation contains the priority of the task (cf. the priority scheduler in 2.1.6). The priority of an I/O event is contained in the task variable generation of its invoking task.

4) The generation of the event variable associated with the task or I/O event. Its storage contains completion (O-BIT as long as the task or I/O event is contained in $PA$) and status value of the task or I/O event.

5) The set of task-event names of all I/O events invoked by the task. This set is used to terminate all those of them which are yet active at task end.

6) The set of all pointers of based storage allocated and not yet freed by the task. This set is used to free this storage at task end.
7) An unlock argument for the I/O event, used by the completing wait statement to unlock the transmitted record if necessary.

8) A list of input-output condition arguments, accumulated by the I/O event, used by the completing wait statement for activation of input-output condition calls.

9) A check condition argument, used by the completing wait statement in case of input for activation of check condition calls.

(7) \( \text{is-io-cond-arg} = (\langle s\text{-cond} : \text{is-f-io-cond} \rangle, \langle s\text{-cbif} : \text{is-cbif} \rangle, \langle s\text{-type} : \text{is-intg-val} \rangle) \)

Ref.: is-f-io-cond 2-32(106), is-cbif 9-12(20)

(8) \( \text{is-check-cond-arg} = (\langle s\text{-ref-list} \rangle \text{is-ref-list} >, \langle s\text{-e} : \text{is-e} \rangle) \)

Ref.: is-e 2-18(37)

(9) \( \text{is-unlock-arg} = (\langle s\text{-uis-n} \rangle, \langle s\text{-rec-opt} \rangle \text{is-rec-opt} >) \)

Ref.: is-n 2-17(35)

(10) \( \text{is-rec-opt} = \)

\( (\langle s\text{-event} : \text{is-gen v is-?} \rangle, \langle s\text{-from} : \text{is-gen v is-Ω} \rangle, \langle s\text{-into} : \text{is-gen v is-Ω} \rangle, \langle s\text{-key} : \text{is-char-val-list v is-Ω} \rangle, \langle s\text{-keyto} : \text{is-gen v is-ps-gen v is-Ω} \rangle, \langle s\text{-set} : \text{is-gen v is-Ω} \rangle, \langle s\text{-nolock} : \text{is-opt} \rangle) \)

Ref.: is-char-val 8-4(9), is-ps-gen 7-11(26), is-gen 2-27(96)

2.1.3 The control C

The control part \( \mathcal{C} \) of a state \( \mathcal{E} \) of the PL/I machine contains a set of instructions. The control part is an abstract object described by a control representation (chapter 4, in /1/). The instructions of a control part can be considered as arranged in the form of a control tree where each instruction may have a set of successor instructions and the instructions at the terminal nodes of the tree are candidates for immediate execution. The execution of such an instruction in \( \mathcal{C} \) modifies the state of the machine as specified for the individual instructions.
A predicate is-terminal is defined over a control tree which characterizes the set of all selectors pointing to all terminal nodes of the control tree.

An object of type control satisfies the predicate

\[(11) \text{is-c,}\]

2.1.4 The time and date part TD

PL/I specifies the builtin functions TIME and DATE. Both functions receive initial input values when the program interpretation is started. The DATE value remains constant throughout the interpretation. The TIME value is updated in the computation, dependent on the state \( \mathcal{E} \) and the instruction being interpreted. The actual updating values are implementation defined.

A delay statement may specify that a task shall be waiting up to a specified point of time. The time updating function upd-time(cf. 2.1.6) inspects at each instruction execution whether a point of time is passed for which a delay statement is waiting.

The information necessary for these purposes is contained in the time and date part TD of the machine state \( \mathcal{E} \), which consists of: the current time, the date and the set of times currently waited for by delay statements.

\[(12) \text{is-td} = \langle s\text{-time:is-intg-val}, s\text{-date:is-intg-val}, s\text{-delay:is-intg-val-set} \rangle\]

2.1.5 The event trace ET

A wait statement in PL/I has to inspect a series of specified event variable generations for completion, since control of the task may not continue before they have been set "complete". In this respect the wait statement has not only to recognize those generations which are "complete" at the time when being inspected, but also those which have been "complete" at any time during the wait statement (and possibly have been set "incomplete" in the meantime). For this purpose serves a global state component, the event
trace ET, which keeps track of all completions of event variable generations and of all starts of executions of wait statements. It may be considered as a time axis, recording the time ordering of event variable completions and wait statement starts by listing the completed event variable generations and unique names representing the wait statement executions. This list enables each wait statement to ask which generations have been completed later than its own start.

\[(13) \text{is-et} = \text{is-et-entry-list}\]

\[(14) \text{is-et-entry} = \text{is-gen} \lor \text{is-n}\]

Ref.: is-gen 2-27(96), is-n 2-17(35)

2.1.6 The language function of the PL/I machine

The PL/I machine describes the interpretation of a PL/I program by defining the set of possible computations resulting from the program. A computation is a sequence of states of the PL/I machine:

\[\xi_0, \xi_1, \xi_2, \ldots\]

satisfying the following two conditions:

1) \(\xi_0\) is an initial state corresponding to \(t\) as given by the function initial-state (cf.3-2(2)):

\[(\exists \text{stg,es,time,date,tev,funct-gen,call})\]

\[(\xi_0 = \text{initial-state(stg,es,time,tev,funct-gen,call)}).\]

2) Each state \(\xi_{i+1}\) is produced by the language function \(A\) from its predecessors:

\[\xi_{i+1} \in A(\xi_i).\]

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

\[\text{is-Os-pa}(\xi_n).\]
The language function $\Lambda$, which for the PL/I machine defines the set of possible successor states of a given state $\xi$, is defined in the following. It corresponds to the language function $\Lambda$ as defined in /1/ for a single task machine. In order to include the possibility of parallel tasks and I/O events (cf. the introduction to section 2.1) the concept of an abstract machine as described in /1/ has to be modified; it is this modified machine which is referred to as "the PL/I machine" throughout this document.

A state $\xi$ contains in its parallel action part PA a set of parallel tasks or I/O events, each of them under an individual selector ten. Based on the priorities of the different parallel actions the priority scheduler of the PL/I machine selects one of the active parallel actions (i.e. those not waiting and containing at least one instruction to be executed) for execution.

This is performed by the function pri-sched which returns the selector ten of this parallel action.

All components of the selected parallel action are copied into the corresponding components of the state of the machine itself (by the function copy). To this modified state the function $\Psi(\xi, z)$ is applied for one of the selectors $z$ satisfying the condition $x \in \text{tn-s-c}(\xi)$ as defined in /1/, i.e. one of the terminal instructions of the control of the selected parallel action is executed as described in section 1.2.3.2. Then the copied state components are recopied into the selected component of PA (by the function recopy).

Of course, one could imagine an equivalent model which, instead of this copying and recopying the selected parallel actions, executes the instructions "in place". But the presented model presents some notational conveniences when describing the individual instruction definitions: Instead of speaking, e.g., about "the environment of the current task" $s\text{-e\text{-ten\text{-s\text{-pa}(\xi)}}}$, where the selector ten of the current task has to be explicitly known, in the presented model one needs only to speak about "the environment" $s\text{-e(\xi)}$. Only one instruction definition (4-9(15)) really reflects the presented copying and recopying model.

Finally the time is updated by the function upd-time, which also tests whether a point of time is passed for which a delay statement is waiting. The time increment is given by the implementation defined function exec-time, which yields the execution time of the executed instruction.
The final result of the described processes is a successor state of the original state.

**Metavariables**

- $\xi$: is-state, the machine state to which the language function $\Lambda$ is applied or one of the intermediately modified states.
- ten: is-s-main v is-n, the selector of the task or I/O event selected by the priority scheduler.
- $\alpha$: is-c, the selector selecting the instruction to be executed from the control of the PL/I machine (and of the selected task).
- ti: is-intg-val, the execution time of the executed instruction.

**(15)**

$$\Lambda(\xi) =$$

$$\{ \exists \text{ten}(\text{is-active-pa}(\text{ten}, \xi)) \rightarrow \{ \xi_4 \mid \forall x \in \text{ten}(c_{\text{pri}}) \} \}

T \rightarrow \{ \text{upd-time}(\xi, \text{UPD-TIME}) \}$$

where:

- $\text{ten}_{\text{pri}} = \text{pri-sched}(\xi)$
- $c_{\text{pri}} = s \cdot \text{ten}_{\text{pri}} \cdot \text{s-pa}(\xi)$
- $\xi_1 = \text{copy}(\xi, \text{ten}_{\text{pri}})$
- $\xi_2 = \psi(\xi_1, \alpha)$
- $\xi_3 = \text{recopy}(\xi_2, \text{ten}_{\text{pri}})$
- $\xi_4 = \text{upd-time}(\xi_3, \text{exec-time}(\xi, \text{ten}_{\text{pri}}, \alpha))$

Note: If all tasks are waiting, only the time is updated in the state $\xi$, until possibly the waiting condition of a delay statement is satisfied.

**(16)**

$$\text{is-active-pa}(\text{ten}, \xi) =$$

$$\{ \text{is-s-main}(\text{ten}) \lor \text{is-n}(\text{ten}) \} \land$$

$$\neg \text{is-}O \cdot s \cdot \text{c-ten-s-pa}(\xi) \land$$

$$\text{is-}O \cdot \text{wait} \cdot s \cdot \text{te} \cdot \text{ten} \cdot s \cdot \text{pa}(\xi)$$

Ref: is-s-main 2-4(3), is-n 2-17(35)

Note: This predicate is satisfied if ten is the selector of a task or I/O event which is not waiting and has at least one instruction to be executed in its control.
(17) pri-sched(\xi)

Note: This function, the priority scheduler, is implementation defined, it returns the selector of an active task or I/O event. It satisfies axiom (18) and should satisfy also axiom (19), though the latter needs not be necessarily true.

(18) is-active-pa(pri-sched(\xi), \xi)

(19) \neg(\exists \text{ten})(\text{is-active-pa(ten, } \xi) \& \text{priority(ten, } \xi) > \text{priority(pri-sched(\xi), } \xi))

(20) \text{priority(ten, } \xi) = \text{op-val } \text{gen-op}(s-tv \ast s-te \ast \text{ten} \ast s-pa(\xi), s-s(\xi))

where: ten \text{ten} = s-ten \ast s-te \ast \text{ten} \ast s-pa(\xi)

Ref.: op-val 8-11(40), gen-op 7-19(47)

Note: This function determines the priority value of the task or I/O event selected by ten. In the case of a task is ten \text{ten} = \text{ten} and the function yields the value of the associated task variable. In the case of an I/O event it yields the priority of the calling task.

(21) \text{copy(}\xi, \text{ten}) = \mu(\xi; s-te:s-te(pa_{ten})>, s-ag:s-ag(pa_{ten})>, s-fd:s-fd(pa_{ten})>, s-e:s-e(pa_{ten})>, s-ei:s-ei(pa_{ten})>, s-cs:s-cs(pa_{ten})>, s-d:s-d(pa_{ten})>, s-ci:s-ci(pa_{ten})>, s-c:s-c(pa_{ten})>)

where: pa_{ten} = \text{ten} \ast s-pa(\xi)

(22) \Psi(\xi, \omega)

Note: This function is as defined in /1/. It describes the execution of the instruction \omega \ast s-c(\xi) on the state \xi (cf. also section 1.2.3.2 of this document).
(23) $tn(c)$

Note: This function is defined in /1/. It yields the set of all selectors which select from a control $c$ a terminal instruction, i.e. an instruction which is a candidate for immediate execution.

(24) $\text{recopy}(\xi, \text{ten}) =$

$$
\mu(\xi; <\text{ten}\cdot \text{pa}; \mu_{\theta}(<\text{ste}\cdot \text{te}(\xi)>, \\
<\text{ag}\cdot \text{ag}(\xi)>, \\
<\text{fd}\cdot \text{fd}(\xi)>, \\
<\text{se}\cdot \text{se}(\xi)>, \\
<\text{ei}\cdot \text{ei}(\xi)>, \\
<\text{cs}\cdot \text{cs}(\xi)>, \\
<\text{d}\cdot \text{d}(\xi)>, \\
<\text{ci}\cdot \text{ci}(\xi)>, \\
<\text{c}\cdot \text{c}(\xi)>)>
$$

(25) $\text{upd-time}(\xi, \text{ti}) =$

$$
\neg(\exists \text{time})(\text{time} \in \text{delay} \cdot \text{td}(\xi) \& \text{time} < \text{time}_{\text{ti}}) 
\quad \rightarrow 
\mu(\xi; <\text{time}\cdot \text{td}; \text{time}_{\text{ti}}>) \\
T \quad \rightarrow 
\mu(\xi; <\text{time}\cdot \text{td}; \text{time}_{\text{ti}}>, \\
<\text{delay}\cdot \text{td}; \text{delay}_{\text{ti}}>, \\
<\text{pa}\cdot \text{activate-tasks}\cdot \text{pa}(\xi)>)
$$

where: $\text{time}_{\text{ti}} = \text{time} \cdot \text{td}(\xi) + \text{ti}$

$\text{delay}_{\text{ti}} = \{\text{time} | \text{time} \in \text{delay} \cdot \text{td}(\xi) \& \text{time} > \text{time}_{\text{ti}}\}$

Ref: activate-tasks 4-18(44)

Note: This function increments the time component of the state by $\text{ti}$. If thereby a point of time is passed, for which a delay statement is waiting, then all tasks are set active (cf.4.4).

(26) $\text{exec-time}(\xi, \text{ten}, \infty)$

Note: This function is implementation defined. It yields the execution time of the instruction $\infty \cdot \text{c} \cdot \text{ten} \cdot \text{pa}(\xi)$ for the given state $\xi$. It satisfies the condition (27).

(27) $\text{is-intg-val} \cdot \text{exec-time}(\xi, \text{ten}, \infty) \& \text{exec-time}(\xi, \text{ten}, \infty) > 0$

2.1.6
2.1.7 Sharing of storage by parallel tasks and input and output events

PL/I specifies that parallel tasks and I/O events can access the same piece of storage or external storage. If at least one of them changes this shared piece of storage or external storage, the language specifies that the resulting state is unpredictable unless an explicit synchronization by a wait statement ensures that the different accesses cannot be executed simultaneously.

The language specifies that access to event variables is uninterruptable, thus guaranteeing that the sharing of storage containing event data by parallel tasks or I/O events does not result in undefined situations.

This document, which defines the interpretation of PL/I actions, uses instructions whose execution inherently is uninterruptable due to their functional definition. The description of the tasking mechanism assumes that the interpretation of one task is interrupted by the priority selection only at the level of the instructions of the definition. An actual implementation may take this decision based on a priority evaluation at some point in the flow of interpretation which is below the level of detail described by the language and thus not defined in the current document. If this situation arises an axiom could be given which at the termination of a computation specifies whether the computation due to storage sharing potentially may have yielded an undefined result.
2.2 Flow of Control

This section describes the state components of the PL/I machine which govern the flow of control through a PL/I program within a single task. It does not include the control of the PL/I machine itself, which might be thought of as the "hardware" control of the machine, while the state components described here govern the flow "at PL/I level".

These components are: the dump D (cf. 5.1.1), the epilogue information EI (cf. 5.1.7), and the control information CI (cf. 5.2.1).

2.2.1 The dump D

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

All information in the state of the PL/I machine that is valid only for the time of a block activation is contained in the six block-local state components: the environment E (cf. 2.3.2), the epilogue information EI (cf. 2.2.2), the condition state CS (cf. 2.5), the dump D itself, the control information CI (cf. 2.2.3), and the control C (cf. 2.1.3). These components have to be available for the complete time of a block activation; especially, they have to be reserved for the time of a nested block activation which may maintain its own block-local state components, in order to be available after termination of the nested block activation. The information contained in these state components may partially be inherited into nested block activations, but it may not be given back from nested block activations into outer ones.

All this is guaranteed by the dump mechanism acting as described in 5.1.1 and initiating with Q (cf. 3.1): Whenever a new block activation is established the block-local state components of the old one become components of the dump; thereby the block-local state components of the before last block activation become components of the dump component of the dump, and so on. Whenever a block activation is terminated, the block-local state components are reinstalled into the state from the dump.
2.2. The epilogue information \( \text{EI} \)

The epilogue information \( \text{EI} \) contains all information necessary to terminate a block activation correctly. This is the set of all local automatic and dummy variables to be freed at block end (represented by their unique aggregate names) and the set of all tasks attached during the block activation and to be detached at latest at its end (represented by their unique task-event names).

In addition to this information necessary at each block end, the return from a procedure activation, especially by means of the return statement (cf. 5.6.4), needs further information:

1) whether the current block activation is a begin block, a regular procedure call, a task call or an on-unit,
2) if the last procedure activation was by a function reference, the generation of the dummy variable created for the function value,
3) whether the last procedure activation is the main procedure of the program, i.e. the finish condition is to be raised on return, or not.

For handling of the epilogue information at block entry and exit cf. 5.1.7, at program initiation cf. 3.1 and 3.2, at procedure call cf. 5.6.1, at procedure exit cf. 5.6.4, at condition call cf. 9.4.2.

\[
\text{(30)} \quad \text{is-d} = \text{is-} \emptyset \lor \langle \text{s-e:is-e}, \\
\langle \text{s-ei:is-\text{ei}}, \\
\langle \text{s-cs:is-cs}, \\
\langle \text{s-d:is-d}, \\
\langle \text{s-ci:is-ci}, \\
\langle \text{s-c:is-c} \rangle \\
\rangle
\]

Ref.: is-e 2-18(37), is-ei 2-15(31), is-cs 2-31(102), is-ci 2-16(32), is-c 2-7(11)

\[
\text{(31)} \quad \text{is-ei} = \langle \text{s-block-act:is-BLOCK} \lor \text{is-PROC} \lor \text{is-TASK} \lor \text{is-ON}, \\
\langle \text{s-free-set:is-n-set}, \\
\langle \text{s-task-set:is-n-set}, \\
\langle \text{s-funct-gen:is-gen} \lor \text{is-} \emptyset, \\
\langle \text{s-main:is-opt} \rangle \\
\rangle
\]

Ref.: is-n 2-17(35), is-gen 2-27(96)
2.2.3 The control information CI

The control information CI governs the flow of control within a single block activation. Its structure and the governing mechanism is described in detail in 5.2.1. It is used also by the goto statement (cf. 5.5).

\[(32) \text{is-ci} = \text{is-o} \lor (\langle \text{s-tx:is-p-}\text{-proc-st-list} \lor \text{is-p-if-st}, \langle \text{s-sc:is-intg-val}, \langle \text{s-cd:is-cd} \rangle)\]

Ref.: is-p-\text{-proc-st 3-10(17)}, is-p-if-st 3-10(17)

Note: CI = 0 is characteristic for the time of prologue and epilogue actions of a block activation. An if-statement as s-tx component of CI may only occur immediately during execution of the goto statement (cf. 5.5.2).

\[(33) \text{is-cd} = (\langle \text{s-ci:is-ci}, \langle \text{s-c:is-c} \rangle)\]

Ref.: is-c 2-7(11)

2.3 Associating Identifiers with Meaning

For each declaration occurring in the text of a block on activation of this block a unique name is created. This unique name is entered into the environment for the declared identifier. The declaration, essentially as occurring in the program text, is entered under the unique name into the attribute directory. That information which is necessary to know the meaning of an identifier besides its declaration is composed and entered under the unique name into the denotation directory. So, when an identifier occurs in the text, its meaning is found by the following chain: The unique name is found in the environment, and by this unique name the declaration is found in the attribute directory and the denotation in the denotation directory.

This section describes: the creation of unique names (2.3.1), the structure of the environment (2.3.2), of the attribute directory (2.3.3), and of the denotation directory (2.3.4).
2.3.1 Creation of unique names

In many situations during the interpretation of a PL/I program unique names are needed. These unique names are on the one hand elementary objects entered as components into state parts. On the other hand, they are simple selectors used to insert components into state parts. They serve as a link between those locations in the state, where they occur as objects, and that information which is inserted by them as selectors. Thereby the same information may be linked to different locations.

The unique names are an infinite, ordered subset of the set of elementary objects. There is a state component, the unique name counter \(UN\), which has the only purpose to count the unique names already used, thereby guaranteeing that no unique name is used twice. Whenever a new unique name is needed, the instruction \texttt{un-name} returns one which is different from all unique names used before.

\begin{align}
(34) \text{is-un} &= \text{is-intg-val} \\
(35) \text{is-n} &= \\
\text{Note: This predicate characterizes the enumerably infinite set of unique names } n_0, n_1, n_2, \ldots \text{ which are elementary objects.} \\
(36) \text{un-name} &= \\
\text{PASS:} & n_{\text{UN}} \\
& s_{\text{UN}} := \text{UN} + 1
\end{align}

2.3.2 The environment \(E\)

The environment \(E\) connects each declaration occurring in the program text, or better the (qualified and unqualified) names declared by it, with a unique name. The same name declared in different block activations receives different unique names and thereby different meaning. The environment is a block-local state component (cf. 2.2.1); the "current" environment installed in the state contains only the unique names leading to the meaning of names valid in the current block activation.
A qualified or unqualified name is an identifier list idl (e.g. the s-id-list component of a reference, cf. 12.2.3.1). The corresponding unique name is contained in the environment under the selector sel(idl) (cf. 1.1.3(1)). The creation of the environment is described in 5.1.2.

Often names occurring in the text of a block are to be interpreted at a time when a nested block is active and has installed its own updated environment into the state. Therefore often an environment is passed to instructions and used by them instead of the current environment component E of the state (e.g. allocation of controlled variables, cf. 6.1.1).

(37) is-e = ([sel(idl):is-n] || is-id-list(idl))
Ref.: is-n 2-17(35)

2.3.3 The attribute directory AT

The attribute directory AT contains for each declaration of each block activation, which had already been established, under its unique name an entry which consists of its attribute and the environment of that block activation. It is described in detail in 5.1.3.

The attribute of a declaration is in general the completely unchanged text of the declaration itself, without the s-den component of an entry declaration if existent.

There are two exceptions from this general rule (cf. 5.6.2.4):

1) A file parameter has as its attribute the attribute of the passed argument instead its own declaration text.

2) An entry parameter to which a builtin function has been passed has in addition to its usual form an additional component denoting this fact.

(38) is-at = ([<n:(<s-attr:is-attr>,<s-e:is-e>) || is-n(n)])
Ref.: is-n 2-17(35), is-e 2-18(37)

(39) is-attr = is-prop-var v is-entry v is-file v is-defined v is-based v is-builtin v is-LABEL v is-format-attr v is-generic v is-COND

(40) is-builtin = is-BUILTIN v is-param-builtin

2.3.3
The denotation directory $\text{DN}$ contains for each declaration of each block activation, which had already been established, under its unique name an entry which is called its denotation. The denotation contains all information necessary for interpretation of a name, as far as it is not yet contained in the attribute in $\text{AT}$ (cf. 2.3.3).

The denotation directory and the denotations of the different types of declarations are described in detail in 5.1.5, for a builtin function passed as argument to an entry parameter in 5.6.2.1.

\begin{verbatim}
(41) $\text{is-param-builtin} = \langle \text{s-builtin:is-T},$
\qquad \langle \text{s-scope:is-PARAM},$
\qquad \text{\langle s-param-list:is-descr-list \& is-\rangle},$
\qquad \langle \text{s-ret-type:is-ret-type} \rangle$
\end{verbatim}

\subsection*{2.3.4 The denotation directory $\text{DN}$}

The denotation directory $\text{DN}$ contains for each declaration of each block activation, which had already been established, under its unique name an entry which is called its denotation. The denotation contains all information necessary for interpretation of a name, as far as it is not yet contained in the attribute in $\text{AT}$ (cf. 2.3.3).

The denotation directory and the denotations of the different types of declarations are described in detail in 5.1.5, for a builtin function passed as argument to an entry parameter in 5.6.2.1.

\begin{verbatim}
(42) $\text{is-dn} = \langle \langle \text{n:is-den} \mid \text{is-n(n)} \rangle \rangle$
\end{verbatim}

\begin{verbatim}
(43) $\text{is-den} = \text{is-n} \& \text{is-entry-den} \& \text{is-eda} \& \text{is-builtin-den} \&$
\qquad \text{is-label-den} \& \text{is-format-den}$
\end{verbatim}

\begin{verbatim}
(44) $\text{is-entry-den} = \langle \text{s-id:is-id},$
\qquad \langle \text{s-body:is-p-body},$
\qquad \langle \text{seis-e},$
\qquad \langle \text{s-bpp:is-dyn-pref-part},$
\qquad \text{\langle s-param-list:is-edescr-list \& is-\rangle},$
\qquad \langle \text{s-ret-type:is-e-ret-type} \rangle$
\end{verbatim}

\begin{verbatim}
(45) $\text{is-edescr} = \text{is-var-edescr} \& \text{is-entry-descr} \& \text{is-file-descr} \& \text{is-\}$
\end{verbatim}

\begin{verbatim}
(46) $\text{is-var-edescr} = \langle \text{s-scope:is-CTL \& is-\rangle},$
\qquad \langle \text{sa:is-eda},$
\qquad \langle \text{s-dens:is-dens} \rangle$
\end{verbatim}

\begin{verbatim}
(47) $\text{is-e-ret-type} = \text{is-arith} \& \text{is-string-eda} \& \text{is-area-eda} \& \text{is-pic} \& \text{is-PTR} \&$
\qquad \text{is-OFFSET}$
\end{verbatim}

\begin{verbatim}
(48) $\text{is-builtin-den} = \langle \text{s-id:is-id},$
\qquad \text{\langle s-param-list:is-descr-list \& is-\rangle}$
\end{verbatim}
2.4 Storage Access

2.4.1 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 unessential as to which one is chosen. The definition leaves these possibilities open. An object $\text{stg}$ satisfies the predicate

(51) $\text{is-stg}(\text{stg})$

if it has the properties described in this chapter.

Metavariables

<table>
<thead>
<tr>
<th>Metavariable</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>$p$</td>
<td>$\text{is-ptr-val}$</td>
</tr>
<tr>
<td>$ps$</td>
<td>$\text{is-ptr-val-set}$</td>
</tr>
<tr>
<td>stg</td>
<td>$\text{is-stg}$</td>
</tr>
<tr>
<td>$z$</td>
<td>$\text{is-size}$</td>
</tr>
</tbody>
</table>
2.4.1.1 Geometric properties of storage

P is a set of pointers, i.e., a set of elements p which satisfy the implementation defined predicate

(52) is-\text{ptr-val}(p)

(53) is-\text{ptr-val-set}(P)

P includes the unit pointer I:

(54) is-\text{ptr-val}(I)

P is closed under functional composition:

(55) \(( \forall p-1,p-2) (p-1 \in P \ & \ p-2 \in P \Rightarrow p-1 \circ p-2 \in P)\)

(56) \(( \forall p)(p \circ I = I \circ p = p)\)

An element p of P may or may not be applicable to a specific storage. The predicate

(57) is-applic(p, stg)

is true if p is applicable to stg, false otherwise. If p is applicable to stg then the result of the functional application is also of type storage:

(58) \(( \forall p, stg) (\text{is-applic}(p, stg) \Rightarrow \text{is-stgop}(stg))\)

For any storage stg, there is the function

(59) size(stg)

defined. The range of the function size is an implementation-defined quantity. The size of a storage is related to the range of values that can be kept in that storage (if used directly as data storage), as well as to the number and type of allocations that can be made in it (if used as main storage or as an area), see chapter 6.1.4. The size of a storage satisfies the implementation-defined predicate

(60) is-size(z)

The applicability of a pointer to storage is only a function of the size of that storage. Moreover, the size of a storage resulting from the application of a pointer to some other storage, is only a function of the pointer.
For any two storages, \( \text{stg-1} \) and \( \text{stg-2} \), the predicate

\[
\text{is-indep(stg-1, stg-2)}
\]

is either true or false. Two storages being independent describes the property of having no common storage. The following properties hold for the predicate:

\[
(\forall \text{stg-1, stg-2}) (\text{is-indep(stg-1, stg-2)} \supset \text{is-indep(stg-2, stg-1)})
\]

The properties of a storage which are implied by its size are called the geometric properties. If a storage is made part of the PL/I machine, then these properties remain unchanged throughout the computation.

### 2.4.1.2 The value representation

The "contents" of a storage are called "value representation". A value representation is an implementation defined entity used to represent a value, of some type, in storage (cf. 8.1). The value representation associated with a storage \( \text{stg} \) is obtained by the function

\[
\text{el-rf(stg)}
\]

The result satisfies the implementation defined predicate \( \text{is-vr} \) (a further characterization of value representations is given in 8.1.3).

The value representation associated with a part \( p(\text{stg}) \) of \( \text{stg} \) can be changed by the elementary assignment function:

\[
\text{el-ass(p, vr, stg)}
\]

The result is a storage \( \text{stg-1} \) differing from \( \text{stg} \) only in that \( \text{vr} \) is now associated with \( p(\text{stg-1}) \).
The set of value representations which can be assigned to a storage of size \( z \) is given by the implementation defined function \( \text{range}(z) \):

\[
(\forall z) \left( \text{is-vr-set} \Rightarrow \text{range}(z) \right)
\]

Given the value representation \( vr \) of a storage \( stg \) (e.g. of an area) one may ask for the value representation associated with the \( p \)-part of \( stg \) (e.g. for the value representation associated with an allocated part of the area). This value representation is given by the function \( \text{vr-rf}(p,vr) \), defined by

\[
(\forall stg,p) \left( \text{is-applic}(p, stg) \Rightarrow \text{vr-rf}(p, \text{el-rf}(stg)) = \text{el-rf}\cdot p(stg) \right)
\]

### 2.4.1.3 The allocation state

A storage if used as main storage or as an area is associated with an allocation state. The allocation state is a set of pointers identifying those parts of the storage which have already been reserved by allocation in that storage. The allocation state is a function of the value representation associated with the storage. It is obtained by the function

\[
\text{alloc-state} \cdot \text{el-rf}(stg)
\]

The function \( \text{alloc-state} \) is not defined for all possible value representations. Value representations giving a defined allocation state satisfy the predicate \( \text{is-area-type} \).

\[
(\forall stg) \left( \text{is-area-type} \Rightarrow \text{el-rf}(stg) \Rightarrow \text{is-ptr-val-set} \Rightarrow \text{alloc-state} \cdot \text{el-rf}(stg) \right)
\]

On allocation the pointer identifying the storage part being reserved is entered by the elementary allocation function. The function

\[
\text{el-alloc}(p-1,p, stg)
\]

used on reserving the part \( p-1 \) of \( p(stg) \) gives a new storage differing from \( stg \) only in that the allocation state of \( p(stg-1) \) is \( \{p-1\} \) added to the allocation state of \( p(stg) \).

The function

\[
\text{el-free}(ps,p, stg)
\]

used on freeing the parts of \( p(stg) \) identified by the pointer set \( ps \), gives a new storage \( stg-1 \) differing from \( stg \) only in that the allocation state of \( p(stg-1) \) is the allocation state of \( p(stg) \) diminished by \( ps \).
The function

(77) clear-alloc-state(p, stg)

gives a new storage stg-1 differing from stg only in that the allocation state of p(stg-1) is the empty set.

2.4.1.4 Usability of storage space

Not all the main storage and not all parts of an area can be used for making allocations. There is a certain amount of storage reserved for organizational purposes. The part of a storage of size z which is used to keep the allocation state is defined by the pointer given by the function allst-part(z):

(78) (\forall z)((is-ptr-valo allst-part(z) & is-applic-1(allst-part(z),z))

The allocation state of a certain storage therefore is given by a function, alloc-state-1, of the allocation state part:

(79) \forall (stg) (alloc-state-1(stg) = alloc-state-1*(allst-part*size(stg))(stg))

The predicate is-usable(p,z) is true if the part p of a storage of size z can be used, without destroying the allocation state:

(80) (\forall p,z)(is-usable(p,z) \equiv is-applic-1(p,z) & is-indep-1(p,allst-part(z)))

2.4.1.5 Properties of the assignment function

With the assumption that is-applic(p, stg) and is-applic(p-1, stg) the following statements hold:

(81) (\forall p,vr, stg)(size(stg) = size*el-ass(p,vr, stg))

(82) (\forall p,vr, stg)(vr \in \text{range} \cdot size \cdot p(stg) \supset el-rf*el-ass(p,vr, stg) = vr)

(83) (\forall p-1,p,vr, stg)(is-indep-1(p-1,p) \supset p-1(stg) = p-1*el-ass(p,vr, stg))

cont'd
Note: (81) states that an assignment leaves the geometric properties of storage unchanged, (82) states the essential functional property of the assignment function, and (83) implies that the value representations of all storage places that are independent of the one to which the assignment is made, are left unchanged.

### 2.4.1.6 Properties of the allocation function

With the assumption that is-applic(p, stg), is-applic(p-1, stg), and is-area-type-el-rf*p(stg), the following statements hold:

(84) \((\forall p-O,p,stg) (\text{size} \cdot \text{el-alloc}(p-O,p,stg) = \text{size}(stg))\)

(85) \((\forall p-O,p,stg) (\text{is-usable}(p-O,\text{size} \cdot p(stg)) \land \text{alloc-state} \cdot \text{el-rf} \cdot p \cdot \text{el-alloc}(p-O,p,stg) = \{p-O\} \cup \text{alloc-state} \cdot \text{el-rf} \cdot p(stg))\)

(86) \((\forall p-O,p-1,p,stg) (\text{is-indep-1}(p-1,(\text{allst-part} \cdot \text{size} \cdot p(stg)) \cdot p) \land p-1 \cdot \text{el-alloc}(p-O,p,stg) = p-1(stg))\)

Note: (84) states that the geometric properties of storage are left unchanged by the allocation function, (85) represents the essential functional property of the allocation function, and (86) states that the properties of all storage parts that are independent of the relevant allocation state part are left unchanged.

### 2.4.1.7 Properties of the freeing function

With the assumption that is-applic(p, stg), is-applic(p-1, stg), and is-area-type-el-rf*p(stg) the following statements hold:

(87) \((\forall ps,p,stg) (\text{size} \cdot \text{el-free}(ps,p,stg) = \text{size}(stg))\)

(88) \((\forall ps,p,stg) (\text{alloc-state} \cdot \text{el-rf} \cdot p \cdot \text{el-free}(ps,p,stg) = (\text{alloc-stat} \cdot \text{el-rf} \cdot p(stg) = ps))\)

(89) \((\forall ps,p-1,p,stg) (\text{is-indep-1}(p-1,(\text{allst-part} \cdot \text{size} \cdot p(stg)) \cdot p) \land p-1 \cdot \text{el-free}(ps,p,stg) = p-1(stg))\)

cont'd

2.4.1.7
Note: (87) states that the geometric properties of storage are left unchanged by the freeing function, (88) represents the essential functional property of the freeing function, and (89) states that the properties of all storage parts that are independent of the relevant allocation state part are left unchanged.

2.4.1.8 Properties of the clearing function

With the assumption that is-applic(p, stg) and is-applic(p-1, stg) the following statements hold:

(90) \((\forall p, stg) \text{size\_clear\_alloc\_state}(p, stg) = \text{size}(stg)\)

(91) \((\forall p, stg) \text{alloc\_state\_el\_rf}(p, stg) \subseteq \text{clear\_alloc\_state}(p, stg) = \{\}\)

(92) \((\forall p-1, p, stg) \text{is\_indep\_l}(p-1, p) \supset p-1\text\_clear\_alloc\_state(p, stg) = p-1(stg)\)

Note: (90) states that the geometric properties of storage are left unchanged by the clearing function, (91) states the essential functional property, and (92) states that the property of all storage parts that are independent of the part being cleared are left unchanged.

2.4.1.9 The main storage

The part \(S\) of the PL/I machine satisfies the predicate is-mstg(\(S\)). The main storage \(S\) always must have a defined allocation state.

(93) \((\forall stg) \text{is\_mstg}(stg) \supset \text{is\_stg}(stg) \& \text{is\_area\_type\_el\_rf}(stg)\)
2.4.2 The aggregate directory AG

The aggregate directory associates unique aggregate names with lists of generations.

The unique name of each proper variable is associated in the denotation directory with a unique aggregate name, which in turn is associated in the aggregate directory with the list of generations existing for that variable. A generation contains all information necessary to perform the access to the storage part $S$.

Before the initial allocation of an automatic, static, controlled, or dummy variable, the null generation NULL is entered in AG under the unique aggregate name of that variable. Each subsequent allocation puts the new generation on the top of the list of generations (possibly the NULL-generation) already present in AG. It is only for controlled variables, however, that more than one generation may be associated with an aggregate name.

On freeing a variable the top-most generation associated with the aggregate name of that variable in AG is removed. At the termination of a task all generations entered in the aggregate directory of that task, with the exception of the bottom generations (i.e. the first entries made for the aggregate names), are freed.

If generations are passed to non-controlled parameters these generations are entered under the aggregate names of the parameters in AG. These entries are made as bottom generations, i.e. NULL is not entered as the bottom generation, so that no attempt is made to free these generations at task end.

On creation of a new task a new aggregate directory is formed. All top generations of all entries in the aggregate directory of the mother task are entered also in the new aggregate directory, under the same aggregate names, as bottom generations. These inherited generations consequently are not freed at termination of the daughter task.

(94) \[
\text{is-ag} = \{ \langle b : \text{is-ag-entry-list} \rangle \mid \text{is-n(b)} \} \]

(95) \[
\text{is-ag-entry} = \text{is-gen} \lor \text{is-NULL}
\]

(96) \[
\text{is-gen} = \langle s-da : \text{is-eda} \rangle, \\
\quad \langle s-mi : \text{is-mi} \rangle, \\
\quad \langle s-pp : \text{is-pp} \rangle
\]
The part $s$-da($gen$) of the generation $gen$ of a variable is relevant for referencing the variable (cf. 7.3.1). It specifies the structuring and data-type of the variable. The part $s$-mi($gen$), the mapping information, gives the arguments to the storage mapping function (cf. 6.1.4) which determines how the various parts of the variable are located in storage. The data attributes $s$-da($gen$) and $s$-da,$s$-mi($gen$) may differ in array bounds and string lengths (cf. 7.3.5). The pointer part $s$-pp($gen$) specifies in which parts of the storage $\mathcal{S}$ the variable is allocated. For a newly allocated variable the pointer part 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. The pointer part is a list of pointers in that case (cf. 7.3.5).

(97) $is$-$mi$ = ($s$-da:$is$-eda),$s$-dens:$is$-dens$)$

(98) $is$-$pp$ = $is$-ptr-val $\lor$ $is$-$pp$-$list$

Ref.: $is$-ptr-val 2-21(52)
2.5 State Components for Conditions

The major state component dealing with the interpretation of condition situations is the Condition Status CS. Its constituents control the condition enabling state as specified for statements, blocks and procedures, and the condition actions as specified for system action or by executed on statements. One component of CS contains a variety of subparts which are required for interpreting condition builtin function references and condition pseudovariables.

The Condition Status initially is established at program initiation. In order to reflect the dynamic inheritance rules for condition actions and condition builtin function values on block entry, procedure call, function reference and condition call interpretation the current CS is stacked and a new CS is formed from the current CS by setting a new block-prefix-part, copying the condition action part and copying and updating where required the condition-builtin-function-part. On interpretation of a begin block also the block-prefix-part is inherited from CS of the nesting block and updated if explicit block condition prefixes exist in the abstract text.

2.5.1 Static and dynamic condition selectors

For the access to components of major subparts of CS condition selectors are generated by specific functions cond-sel and cond-dyn-sel, which return selector values. The auxiliary function sel is used which generates a selector value from a name list.

(99) cond-sel(cond) =

<table>
<thead>
<tr>
<th>Function</th>
<th>Selector Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>is-CONV(cond)</td>
<td>s-conv</td>
</tr>
<tr>
<td>is-FOFL(cond)</td>
<td>s-fofl</td>
</tr>
<tr>
<td>is-OFL(cond)</td>
<td>s-ofl</td>
</tr>
<tr>
<td>is-SIZE(cond)</td>
<td>s-size</td>
</tr>
<tr>
<td>is-STRG(cond)</td>
<td>s-strg</td>
</tr>
<tr>
<td>is-SUBR3(cond)</td>
<td>s-subrg</td>
</tr>
<tr>
<td>is-UFL(cond)</td>
<td>s-ufl</td>
</tr>
<tr>
<td>is-ZDIV(cond)</td>
<td>s-zdiv</td>
</tr>
<tr>
<td>is-AREA(cond)</td>
<td>s-area</td>
</tr>
<tr>
<td>is-ERROR(cond)</td>
<td>s-error</td>
</tr>
</tbody>
</table>

cont'd
The function \( \text{cond-dyn-sel} \) is used to generate unique selectors from condition names where the names of constants, variables, or files appearing in the condition name need not be unique. The generated dynamic selectors are used to link the enabling status from \( s\text{-spp}(CS) \) and the condition action from \( s\text{-cap}(CS) \) with references requiring condition actions. The differences in the qualification using unique names from the environment or denotation directory serve the rules of linking scope for internal names, linking declarations for external names and linking an equivalent file set for file names.

\[
(100) \text{cond-dyn-sel}(\text{cond}, \text{env}, \text{dn}, \text{at}) =
\]

\[
\begin{align*}
\text{is-check}(\text{cond}) & \quad \rightarrow \quad \text{sel}(\text{s-id-list}(\text{cond}))(\text{env})(\text{at}) \\
\text{is-name-ed-io-cond}(\text{cond}) & \quad \rightarrow \quad \text{sel}(<\text{s-file}(\text{cond})>)(\text{env})(\text{dn}) \cdot \text{cond-io-sel}(\text{s-io-cond}(\text{cond})) \\
\text{is-f-io-cond}(\text{cond}) & \quad \rightarrow \quad s\text{-f-den}(\text{cond}) \cdot \text{cond-io-sel}(\text{s-io-cond}(\text{cond})) \\
\text{T} & \quad \rightarrow \quad \text{cond-sel}(\text{cond})
\end{align*}
\]

Ref.: is-f-io-cond 2-32(106)

Note: A cond for which \( \text{is-f-io-cond}(\text{cond}) \), is used in cases in the interpretation of files where only the denotation of the file-name is known. This occurs every time when an I/O condition is raised by an interrupt. When the raising is done via SIGNAL, the I/O condition satisfies the predicate \( \text{is-named-io-cond} \) of the abstract syntax. This predicate applies otherwise only in conjunction with on and revert statements.

An auxiliary function \( \text{cond-io-sel} \) generates selectors from io-condition names as obtained by \( \text{s-io-cond}(\text{cond}) \).
2.5.2 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 of a block as obtained by merging the block-prefix-part of the surrounding block with the explicit prefixes of the block itself. The statement-prefix-part selected by s-spp contains all prefixes valid during interpretation of a statement and is obtained by merging the block-prefix-part with explicit prefixes of the 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.

(101) \[
\text{cond-io-sel}(\text{iocond}) = \\
\begin{align*}
\text{is-N\,AME}(\text{iocond}) & \rightarrow s\text{-name} \\
\text{is-ENDFILE}(\text{iocond}) & \rightarrow s\text{-endfile} \\
\text{is-ENDPAGE}(\text{iocond}) & \rightarrow s\text{-endpage} \\
\text{is-KEY}(\text{iocond}) & \rightarrow s\text{-key} \\
\text{is-RECORD}(\text{iocond}) & \rightarrow s\text{-record} \\
\text{is-TRANSMIT}(\text{iocond}) & \rightarrow s\text{-transmit} \\
\text{is-UNDF}(\text{iocond}) & \rightarrow s\text{-undf}
\end{align*}
\]

(102) \[
\text{is-cs} = (\text{s-bpp:is-dyn-pref-part}, \\
\text{s-spp:is-dyn-pref-part}, \\
\text{s-cap:is-cond-act-part}, \\
\text{s-cbif:is-cond-bif-part})
\]

(103) \[
\text{is-dyn-pref-part} = \left(\text{cond-dyn-sel(\text{cond,env,dn,at}):is-opt} \mid \\
\text{is-e(env) \& is-dn(dn) \& is-at(at) \&} \\
\text{(is-CONV(\text{cond}) \lor is-POFL(\text{cond}) \lor is-OFLO(\text{cond}) \lor} \\
\text{is-SIZE(\text{cond}) \lor is-STRG(\text{cond}) \lor is-SUBRG(\text{cond}) \lor} \\
\text{is-UFL(\text{cond}) \lor is-ZDIV(\text{cond}) \lor is-check(\text{cond}))}\right)
\]

Ref.: cond-dyn-sel 2-30(100), is-e 2-18(37), is-dn 2-19(42),
\hspace{1cm}is-at 2-18(38)
(104) is-cond-act-part =

\[
\{ <\text{cond-dyn-sel}(\text{cond},\text{env},\text{dn},\text{at}) : \text{is-cond-act}> \mid
\]

\[
(\text{is-cond}(\text{cond}) \lor \text{is-f-io-cond}(\text{cond})) \& \text{is-e}(\text{env}) \&
\]

\[
\text{is-dn}(\text{dn}) \& \text{is-at}(\text{at}) \}\}
\]

Ref.: is-e 2-18(37), is-dn 2-19(42), is-at 2-18(38), cond-dyn-sel 2-30(100)

(105) is-cond-act =

\[
( <\text{s-e} : \text{is-e} > ,
\]

\[
<\text{s-cond} : \text{is-cond} > ,
\]

\[
<\text{s-bpp} : \text{is-dyn-pref-part} > ,
\]

\[
<\text{s-snap} : \text{is-opt} > ,
\]

\[
<\text{s-on-unit} : \text{is-p-st} \lor \text{is-SYSTEM} > )
\]

Ref.: is-e 2-18(37), is-p-st 3-10(17)

(106) is-f-io-cond =

\[
( <\text{s-f-den} : \text{is-n} > ,
\]

\[
<\text{s-io-cond} : \text{is-io-cond} > )
\]

Ref.: is-n 2-14(22)

Note: For an explanation of the use of is-f-io-cond see Note to formula 2-30(100)

The condition builtin function part of CS contains components containing values of the various condition builtin functions and some additional parts to obtain these values. The parts for ONSOURCE and ONCHAR contain a selector to a generation and a counter value respectively. Additional parts contain the entry name which on specified occasions becomes the new value of ONLOC, the current file on which an I/O operation is performing, the type of the statement being interpreted, and the task or event name for proper completion of I/O events.
(107) is-cond-bif-part =

\[
\langle s\text{-onloc}\text{-id} \ is-\varnothing>, \\
\langle s\text{-oncode}\text{-is-intg\text{-val} \ is-\varnothing}, \\
\langle s\text{-oncount}\text{-is-intg\text{-val} \ is-\varnothing}, \\
\langle s\text{-onfile}\text{-is-id} \ is-\varnothing>, \\
\langle s\text{-onkey}\text{-is-char-val\text{-list} \ is-\varnothing}, \\
\langle s\text{-datafield}\text{-is-char-val\text{-list} \ is-\varnothing}, \\
\langle s\text{-onchar}\text{-is-intg\text{-val} \ is-\varnothing}, \\
\langle s\text{-onsource}\text{-is-n} \ is-\varnothing>, \\
\langle s\text{-entry}\text{-is-id} \ is-\varnothing>, \\
\langle s\text{-curr\text{-file}\text{-is-id} \ is-\varnothing}, \\
\langle s\text{-st\text{-type}\text{-is-st-type} \ is-\varnothing}, \\
\langle s\text{-ten}\text{-is-n} \ is-\varnothing> \\
\rangle
\]

Ref.: is-char-val 8-4(9), is-n 2-17(35)

(108) is-st-type =

is-IF v is-GOTO v is-CALL v is-RETURN v is-WAIT v is-DELAY v is-EXIT v
is-STO P v is-ASSIGN v is-ALLOCATE v is-FREE v is-ON v is-REVERT v
is-SIGNAL v is-OPEN v is-CLOSE v is-GET v is-PUT v is-READ v is-WRITE v
is-REWRITE v is-LOCATE v is-DELETE v is-UNLOCK v is-DISP LAY v is-NUL L
2.6 Input and Output

The state parts which deal with input and output are the file directory FD, the 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 on-conditions.

A file is the total information which is accessible by a file identifier through the state components E, AT, DN, FD, FU and ES. The existence of a file depends on opening and closing; the access of the file identifier to the state parts E, AT, and DN is independent from opening and closing.

2.6.1 The File Directory FD

The file directory FD is local to a task. Entries are made at attaching a new task (see attach-task) and at successful opening (see open-fd-1 and open-fd-2). Deletions are made at proper closing (see close and close-fd-fu). The whole FD is deleted at termination of the task to which it is local (see delete-task). Besides of these changes of an FD no other changes occur.

\[
(109) \text{is-fd} = (\{ \text{f: is-fd-elem} \mid \text{is-f}(f) \})
\]

\[
(110) \text{is-fd-elem} = (\text{s-fu: is-n} \land \text{is-STAND-PRINT}), \\
\text{\langle s-orig: is-ORI \land is-INH \rangle}
\]

Ref.: is-n 2-17(35)

Note: The elements of FD are selected by file names which are with one exception the denotations of file identifiers as fixed by the prepass, and which satisfy the predicate is-n. The element of FD which is the entry for the standard system print file is the only exception to this general rule. This entry is selected through the file name s-stand-print which belongs to the class is-f but not to the class is-n. Whenever an entry for the standard system print file exists, there can be also another entry which is linked with it. This linkage is indicated by the elementary object STAND-PRINT.

Generally, elements of a FD are linked with elements of FU through a file union name belonging to the class is-n and selected by s-fu. The
entry which is linked with the entry of the standard system print file
(if it exists at all) is again the only exception.

Elements of FD are either owned by the task to which the FD is local
or they are inherited.

(111) \text{is-f}(f) =
\text{is-n}(f) \lor f = \text{s-stand-print}

Note: This predicate defines the class of file names
Ref.: is-n 2-17(35)

2.6.2 The File Union Directory $FU$

The file union directory $FU$ is a global state component. New entries
are made at proper opening of a file (see open-fu) and before the inter-
pretation of a string option of a get or put statement (see init-string-1
and init-string-2). Deletions are made at proper closing of a file (see
close and close-fd-fu). Besides making new entries and deletions of whole
elements of the $FU$, the entries are updated at various points during the
interpretation.

The $FU$ gathers all information that neither must be in $ES$, nor needs by
necessity be local to a task, nor is carried in the control.

(112) \text{is-fu} = \{\langle u: is-fu-elem \rangle \mid \text{is-n}(n)\}
Ref.: is-n 2-17(35)
Note: The elements of $FU$ are selected by file union names.

(113) is-fu-elem = is-fu-string \lor is-fu-file

(114) is-fu-string = is-fu-get \lor is-fu-put

(115) is-fu-get = (\langle s:\text{data}: \text{is-char-vr-list}\rangle,
\langle s:\text{comma}: \text{is-opt}\rangle,
\langle s:\text{onsource}: \text{is-char-val-list} \lor \text{is-∅}\rangle,
\langle s:\text{format-list}: \text{is-format-list-list} \lor \text{is-∅}\rangle)
Ref.: is-char-vr 8-5(16), is-char-val 8-4(9)
Note: The list of character value representations selected by $s$-data is the
source for the data transmission. The three other components have the
same meaning as with stream input transmission by a file.
Ref.: is-char-vr 8-5(16), is-gen 2-27(96), is-ps-gen 7-11(26)

Note: The list of character value representations selected by s-data is the intermediate target for the data transmission. This target is assigned to the generation of the character string after completion of the transmission.

(117) \( \text{is-fu-file(fu-elem)} = \)
\[
\text{is-unrest-fu-file(fu-elem) \& is-rest-fu-file(} \delta(\text{fu-elem};s\text{-csa},s\text{-at-title},s\text{-io-env},s\text{-f},s\text{-title}, s\text{-transmit}), s\text{-csa(fu-elem)})
\]

for : is-unrest-fu-file (fu-elem)

Note: An entry of FU for a file, i.e. fu-elem, first of all satisfies a predicate is-unrest-fu-file. The components of fu-elem which depend on the complete set of attributes csa, i.e. s-csa(fu-elem), must additionally satisfy a more restrictive predicate is-rest-fu-file. The component csa, the components of fu-elem selected by s-linesize and and s-pagesize, and all the components of fu-elem which do not depend on csa (except the component selected by s-transmit) remain constant throughout the existence of fu-elem as entry of FU.
(118) \texttt{is-unrest-fu-file} = (\texttt{s-csa} : \texttt{is-csa},
\texttt{s-at-title} : \texttt{is-char-val-list},
\texttt{s-io-env} : \texttt{is-io-opt},
\texttt{s-f} : \texttt{is-f},
\texttt{s-title} : \texttt{is-char-val-list},
\texttt{s-transmit} : \texttt{is-opt},
\texttt{s-position} : \texttt{is-intg-val} v \texttt{is-\_Q},
\texttt{s-endfile} : \texttt{is-opt},
\texttt{s-column} : \texttt{is-intg-val} v \texttt{is-\_Q},
\texttt{s-count} : \texttt{is-bintg-op} v \texttt{is-\_Q},
\texttt{s-format-list} : \texttt{is-format-list-list} v \texttt{is-\_Q},
\texttt{s-comma} : \texttt{is-opt},
\texttt{s-onsource} : \texttt{is-char-val-list} v \texttt{is-\_Q},
\texttt{s-linesize} : \texttt{is-intg-val} v \texttt{is-\_Q},
\texttt{s-line} : \texttt{is-bintg-op} v \texttt{is-\_Q},
\texttt{s-pagetype} : \texttt{is-intg-val} v \texttt{is-\_Q},
\texttt{s-bufferpointer} : \texttt{is-\_ptr-val} v \texttt{is-\_Q},
\texttt{s-key} : \texttt{is-char-val-list} v \texttt{is-\_Q},
\texttt{s-io-ev} : \texttt{is-n-set} v \texttt{is-\_Q},
\texttt{s-read} : \texttt{is-opt},
\{ \texttt{ten} : \texttt{is-char-val-list-set} || \texttt{is-s-main}(\texttt{ten}) \texttt{v is-n}(\texttt{ten}) \})

\textbf{Ref.:} \texttt{is-char-val} 8-4(9), \texttt{is-f} 2-35(111), \texttt{is-csa} 10-2(2),
\texttt{is-bintg-op} 8-11(45), \texttt{is-\_ptr-val} 2-21(52), \texttt{is-s-main} 2-4(3),
\texttt{is-n} 2-17(35)
(119) \text{is-rest-fu-file}(fu-el, csa) =
\begin{align*}
&(\text{is-intg-val}s-position(fu-el) \in \text{DIR } \# \text{ csa}) \land \\
&(\text{is-T}s-endfile(fu-el) \supset \text{OUT } \# \text{ csa}) \land \\
&(\text{is-intg-val}s-column(fu-el) = \text{STREAM } \in \text{ csa}) \land \\
&(\text{is-bintg-op}s-count(fu-el) \lor \text{is-format-list-list } s \cdot \text{format-list}(fu-el) \supset \\
\text{STREAM } \in \text{ csa}) \land \\
&(\text{is-T}s:\text{-comma}(fu-el) \lor \text{is-char-val-list}s-onsource(fu-el) \supset \\
\text{STREAM } \in \text{ csa} \land \text{OUT } \in \text{ csa}) \land \\
&(\text{is-intg-val}s-linesize(fu-el) \supset \text{STREAM } \in \text{ csa} \land \text{OUT } \in \text{ csa}) \land \\
&(\text{is-intg-val}s-pagesize(fu-el) = \text{PRINT } \in \text{ csa}) \land \\
&(\text{is-bintg-op}s-line(fu-el) = \text{PRINT } \in \text{ csa}) \land \\
&(\text{is-ptr-val}s-bufferpointer(fu-el) \supset \text{BUF } \in \text{ csa}) \land \\
&(\text{is-char-val-list}s-key(fu-el) \supset \text{BUF } \in \text{ csa} \land \text{KEYED } \in \text{ csa}) \land \\
&(\text{is-n-set}s-io-ev(fu-el) = \text{RECORD } \in \text{ csa} \land \text{BUF } \notin \text{ csa}) \land \\
&(\text{is-T}s-read(fu-el) \supset \text{SEQ } \in \text{ csa} \land \text{OUT } \notin \text{ csa}) \land \\
(\exists \text{ ten}) ((\text{is-s-main}(\text{ten}) \lor \text{is-n}(\text{ten})) \land \text{is-char-val-list-set } s \cdot \text{ten}(fu-el) \supset \\
\text{EXCL } \in \text{ csa}) \\
\text{for: } \text{is-csa}(\text{csa}) \land (\exists \text{fu-elem}) (\text{is-unrest-fu-file}(\text{fu-elem}) \land \text{fu-el} = \\
\delta(fu-elem; s-csa, s-at-title, s-io-env, s-f, s-title, s-transmit))
\end{align*}

Ref.: \text{is-char-val 8-4(9), is-bintg-op 8-11(45), is-ptr-val 2-21(52)}
\text{is-n 2-17(35), is-s-main 2-4(3), is-csa 10-2(2)}

2.6.3 The External Storage ES

The external storage ES is a global state component. It is a set of
titled data sets which in turn consist of a set of titles and descriptions
and a data set. The titles serve as a link between the FU and ES. Individual
data sets are inspected by the instruction \text{tk-es} and updated by \text{upd-es}.
The data part of a data set is accessed (inspected or updated) at various
points during the computation; the labels of a data set only during explicit
opening and closing.

(120) \text{is-es} = \text{is-titled-dataset-set}
(121) \( \text{is-titled-dataset} = (\langle s\text{-title-descr-set}\rangle, \langle s\text{-dataset}\rangle) \)

(122) \( \text{is-dataset} = (\langle s\text{-header}\rangle, \langle s\text{-char-val-list}\rangle, \langle s\text{-trailer}\rangle, \langle s\text{-char-val-list}\rangle, \langle s\text{-data}\rangle, \langle s\text{-outer-dataset}\rangle) \)

Ref.: is-char-val \( 8-4(9) \)

Note: The header and trailer labels are selected by \( s\text{-header} \) and \( s\text{-trailer} \), respectively.

(123) \( \text{is-title-descr} = (\langle s\text{-title}\rangle, \langle s\text{-descr}\rangle) \)

Note: The title selected by \( s\text{-title} \) names the data set, the data set description selected by \( s\text{-descr} \) gives additional information about the data set.

(124) \( \text{is-title} = \)

Note: This predicate is implementation defined.

(125) \( \text{is-ds-descr} = \)

Note: This predicate is implementation defined.

2.6.3.1 Outer and inner data sets

This section and section 2.6.3.2 deal with the data part \( ods \) (is-outer-dataset(ods)) of a titled data set. This data part is mapped into a so called inner data set whenever the data part is accessed during the computation. Depending on whether the access is only inspection (input case) or change (output or update case) the changed inner data set is mapped back in the representation of an outer data set and inserted in external storage.

(126) \( \text{is-outer-dataset}(ods) = \)

Note: This predicate is implementation defined. Together with the predicates is-io-opt and is-ds-descr, the predicate is-outer-dataset characterizes the domain of the mapping functions of external storage.
(127) \( \text{is-inner-dataset(ids)} = \)

\[
\exists \text{csa} \left( \text{is-csa(csa)} \land \text{is-proper-dataset(csa)}(\text{ids}) \right)
\]

Ref.: is-csa 10-2(2)

Note: An inner data set is called a proper (inner) data set if it is an object satisfying the predicate is-proper-dataset(csa) where the complete set of attributes csa is not a property of the data set but a property of the file accessing the data set.

(128) \( \text{is-proper-dataset(csa)} = \)

\[
\begin{align*}
\text{PRINT} \in \text{csa} & \rightarrow \text{is-print-data-list} \\
\text{STREAM} \in \text{csa} & \rightarrow \text{is-stream-data-list} \\
\text{SEQ} \in \text{csa} \land \text{KEYED} \in \text{csa} & \rightarrow \text{is-keyed-seq-data-list} \\
\text{SEQ} \in \text{csa} & \rightarrow \text{is-seq-data-list} \\
\text{DIR} \in \text{csa} & \rightarrow \text{is-keyed-data-set}
\end{align*}
\]

Note: Access with the stream or sequential attribute maps the outer data set into a list; direct access yields a set.

(129) \( \text{is-print-data} = \text{is-stream-data} \lor \text{is-PAGE-DELIMITER} \lor \text{is-CARR-RETURN} \lor \text{is-TABULATOR} \)

(130) \( \text{is-stream-data} = \text{is-char-val} \lor \text{is-LINE-DELIMITER} \lor \text{is-FILEMARK} \)

Ref.: is-char-val 8-4(9)

Note: The predicates (128),(129) and (130) define the abstract syntax of all proper data sets accessible in stream transmission. Proper inner data sets for stream input and stream output non-print satisfy the same predicate is-stream-data-list.

(131) \( \text{is-keyed-seq-data} = \text{is-keyed-data} \lor \text{is-FILEMARK} \)

(132) \( \text{is-seq-data} = \text{is-data} \lor \text{is-FILEMARK} \)

(133) \( \text{is-keyed-data} = (\langle \text{s-key:is-char-val-list} \rangle, \langle \text{s-size:is-size} \rangle, \langle \text{s-data:is-vr} \rangle) \)

Ref.: is-char-val 8-4(9), is-size 2-21(60), is-vr 2-22(69)
(134) is-data = ( &lt;size:is-size&gt;,
             &lt;data:is-vr&gt;)
Ref.: is-size 2-21(60), is-vr 2-22(69)

Note: The predicates (128), and (131) until (134) define the abstract syntax
of all proper data sets accessible in record transmission. Any individual record data satisfies the predicate is-keyed-data if it is keyed,
otherwise it satisfies the predicate is-data.

2.6.3.2 External storage mapping

There are two three-place functions decipher and cipher which map
outer data sets into inner data sets or inner data sets into outer
data sets, respectively.

Metavariables:
csa  is-csa         a complete set of (file) attributes
io-env is-io-opt    an environment attribute
descr is-ds-descr   a data set description
ods  is-outer-dataset a data set in outer representation
ids  is-inner-dataset a data set in inner representation
i    is-intg-val     an index for an element of a list

(135) decipher(csa,io-env,descr) =
    Note: This function is characterized by the axioms (137), (139)
    (140), (141), (142)

(136) cipher(csa,io-env,descr) =
    Note: This function is characterized by the axioms (138), (139),
    (141), (142).

(137) (∀ csa,io-env,descr,ods)
    (is-inner-dataset×decipher(csa,io-env,descr)(ods))
Ref.: is-inner-dataset 2-40 (127)

Note: Application of the function decipher to its arguments yields a
function which maps an outer data set into an inner data set.
\[(138) \ (\forall \text{csa}, \text{io-env}, \text{descr}, \text{ids}) \]
\[(\text{is-outer-dataset} \cdot \text{cipher(} \text{csa,io-env,descr} \text{)}(\text{ids})) \]

Ref.: is-outer-dataset 2-39 (126)

Note: Application of the function cipher to its arguments yields a function which maps an inner data set into an outer data set.

\[(139) \ (\forall \text{csa}, \text{io-env}, \text{descr}, \text{ids}, \text{ods}) \]
\[(\text{is-proper-dataset(} \text{csa} \text{)}(\text{ids}) \land \text{decipher(} \text{csa,io-env,descr} \text{)}(\text{ods}) = \text{ids} \Rightarrow \] \[\text{IN} \cdot \text{csa} \lor \text{cipher(} \text{csa,io-env,descr} \text{)}(\text{ids}) = \text{ods}) \]

Note: This axiom relates the function decipher with the function cipher. Under the condition that decipher and its arguments define a mapping of a certain outer data set ods into a proper inner data set ids either cipher is the inverse function of decipher or the complete set of attributes csa contains the input attribute in which case the function needs not to be the inverse.

\[(140) \ (\forall \text{csa}, \text{ids})(\exists \text{io-env}, \text{descr}, \text{ods}) \]
\[(\text{is-proper-dataset(} \text{csa} \text{)}(\text{ids}) \land (\text{is-list(} \text{ids}) \Rightarrow \] \[(\forall i)( \neg \text{is-FILEMARK} \cdot \text{elem(}i, \text{ids}))) \Rightarrow \]
\[\text{decipher(} \text{csa,io-env,descr} \text{)}(\text{ods}) = \text{ids}) \]

Note: This axiom states that for nearly any proper inner data set ids satisfying the predicate is-proper-dataset(\text{csa}) there must be an environment attribute io-env, a data set description descr and an outer data set ods which by the function decipher applied to its arguments is mapped into ids. This is not a requirement for inner data sets containing filemarks.

Equations (139) and (140) state that any meaningful proper inner data set, i.e. any data set accessible by input, output or update files must have an outer data set as its equivalent. In other words, an inner data set can be considered as a certain mode of appearance of an outer data set. A special outer data set can have many modes of appearance as an inner data set.
(141) \((\forall \text{csa, ids, ods})\)
\[ ((\exists \text{io-env, descr})(\text{IN} \notin \text{csa} \& \text{PRINT} \notin \text{csa} \& \]
\[ \text{is-proper-dataset} (\text{csa})(\text{ids}) \& \text{cipher} (\text{csa,io-env, descr})(\text{ids}) = \text{ods}) \Rightarrow \]
\[ (\forall \text{csa-1})(\exists \text{io-env, descr})(\text{is-csa}_1(\text{csa-1,csa}) \Rightarrow \]
\[ \text{is-proper-dataset} (\text{csa-1})(\text{ids}_1) \& \]
\[ \text{decipher}(\text{csa-1,io-env, descr})(\text{ods}) = \text{ids}_1)\]
\]

where: 
\[ \text{is-csa}_1(\text{csa-1,csa}) = \text{OUT} \notin \text{csa-1} \& \]
\[ ((\text{csa} = \{\text{OUT,UPD,BUF,UNBUF,EXCL}\}) \subseteq \text{csa-1} ) \]
\[ \text{ids}_1 = (\text{is-list}(\text{ids}) \rightarrow <\text{FILEMARK}> \text{ids} <\text{FILEMARK}>, T \rightarrow \text{ids}) \]

Note: This is a statement on the access to data sets created by output or update files through input or update files. It is guaranteed that any created data set which has not been created by a print file is accessible again. As long as the attributes differ only in the attributes input, output, update, buffered, unbuffered, exclusive and backwards the inner data set accessed will be the same as the created inner data set, except for filemarks, and under the additional restriction that io-env and descr are appropriate.

(142) \((\forall \text{ids, ods})\)
\[ ((\forall \text{csa})(\exists \text{io-env, descr})(\text{IN} \notin \text{csa} \& \text{SEQ} \in \text{csa} \& \text{KEYED} \in \text{csa} \& \]
\[ \text{is-proper-dataset} (\text{csa})(\text{ids}) \& \text{cipher} (\text{csa,io-env, descr})(\text{ids}) = \text{ods}) \Rightarrow \]
\[ (\forall \text{csa})(\exists \text{io-env, descr})(\text{OUT} \notin \text{csa} \& \text{DIR} \in \text{csa} \Rightarrow \]
\[ \text{is-proper-dataset} (\text{csa})(\text{ids}_d) \& \text{decipher} (\text{csa,io-env, descr})(\text{ods}) = \text{ids}_d)\]
\]

where: 
\[ \text{ids}_d = \{\text{elem}(i,\text{ids})| \text{is-length}(\text{ids}) \& \text{-FILEMARK}\text{elem}(i,\text{ids})\} \]

Note: This is a statement on the meaning of data sets created by a sequential keyed file which are accessed by a direct file. In this case - also under the prerequisite of proper io-env and descr- the accessed record data are the same which have been created.
2.6.4 The message part M

The message part M is a global state component. The component of M selected by s-display stores the messages put out during the execution of display statements. The messages are lists of character values which are uniquely identified by a name. The reply messages put in have the same format as the messages displayed, the name serving for relating a reply with a special display message. The component of M selected by s-reply is the only state component which is subject to changes from outside the PL/I machine.

Comments put out as part of the standard system action of certain on-conditions are collected in the component of M selected by s-comment.

During the computation the lists selected by s-display and s-comment are incremented only, and always on their tails. Both lists are never inspected, i.e., have no meaning for the computation. The component selected by s-reply is inspected during the computation only. Also this state component can be considered as list which is incremented but from outside the PL/I machine.

(143) is-m = (<s-display:is-named-message-list>,
           <s-reply:is-named-message-list v is-Ø>,
           <s-comment:is-comment-list>)

(144) is-named-message = (<s-name:is-n>,
                       <s-message:is-char-val-list>)

Ref.: is-n 2-14(22), is-char-val 8-4(9)

(145) is-comment =

Note: This predicate is characterized by axiom 9-4(4)
3. PROGRAM Initialization

This chapter defines the start of the interpretation of a PL/I program, including all those actions which are performed only once during the beginning of the program interpretation and never later during its regular run.

Section 3.1 describes the initial state of the PL/I machine, section 3.2 the initial instruction contained in the control of this initial state, which starts the first actions of the machine, especially the so-called "pre-pass", described in section 3.3.

3.1 Initial State

The interpretation of a PL/I program starts with an initial state $\xi_0$ of the PL/I machine which is essentially a cleared machine. This initial state contains one active task (the "main task") with the task-event name $\text{s-main}$. Its control contains the initial instruction $\text{int-program}$, described in 3.2.

The initial state is a function depending on the following nine parameters:

- $\text{stg}_0$: the initial main storage (cf. 2.4.1). It should be noted that the effect of the interpretation of a program may well depend on $\text{stg}_0$, and that it is the responsibility of the programmer to avoid the influence of $\text{stg}_0$ on his program if he wants (by initialization of storage before use).
- $\text{es}_0$: the initial external storage (cf. 2.6.3). It contains in particular the input data for program execution.
- $\text{time}_0$: an integer value denoting the initial time (cf. 2.1.4).
- $\text{date}_0$: an integer value denoting the date (cf. 2.1.4).
- $\text{tv}_0$: the generation of the task variable associated with the main task (cf. 2.1.2). The corresponding piece of storage is allocated in $\text{stg}_0$ and contains the priority value of the main task.
the generation of the event variable associated with the main task (cf. 2.1.2). The corresponding piece of storage is allocated in stg. Its completion value is O-BIT ("incomplete", cf. 4.1).

funct-gen either \( \Omega \), if the program is activated as a "subroutine", or a generation, if the program is activated as a "function". In the latter case, the corresponding piece of storage is allocated in stg. Into this piece of storage the value to be returned by the program is assigned by the last return statement. So the end state of the computation \( \mathbb{E}_n \) will contain the return value stored in this piece of its storage component.

to the text of the program to be executed. It satisfies the predicate is-program defined by the abstract syntax of text (cf. 12.2).

call \( o \) the initial call statement. It specifies the entry name and arguments of the external procedure to be activated initially (the "main procedure"). This call statement does not contain a pa-option, the task and event variable and priority of the main task are specified by tv and ev directly.

\[
\begin{align*}
\mathbb{E}_o &= \text{initial-state}(stg_o, es_o, time_o, date_o, tv_o, ev_o, funct-gen_o, t_o, call_o) \\
\text{initial-state}(stg, es, time, date, tv, ev, funct-gen, t, call) &= \\
\text{valid-init-state}(stg, es, time, date, tv, ev, funct-gen, t, call) \rightarrow \\
\mu_o(\langle s-s: stg, s-es: es, s-un: 0, s-td: td, s-et: \langle \rangle, s-m: M, s-main: s-pa: \mu_o(\langle s-te: te, s-ei: ei, s-es: CS, s-o: \text{int-program}(t, call) \rangle) \rangle) \rightarrow \\
T &\rightarrow \text{error}
\end{align*}
\]

where: \( td_o = \mu_o(\langle s-time: time, s-date: date, s-display: \{\} \rangle) \),

cont'd
\( M_0 = \mu_0(<s\text{-}display:<>>,
<s\text{-}comment:<>>) \)

\( te_0 = \mu_0(<s\text{-}ten:s\text{-}main>,
<s\text{-}tv:tv>,
<s\text{-}ev:ev>,
<s\text{-}based\text{-}free\text{-}set:{}}>,
<s\text{-}io\text{-}ev:{}}> \)

\( ei_0 = \mu_0(<s\text{-}block\text{-}act:TASK>,
<s\text{-}free\text{-}set:{}}>,
<s\text{-}task\text{-}set:{}}>,
<s\text{-}funct\text{-}gen:funct\text{-}gen>,
<s\text{-}main:T>) \)

\( CS_0 = \mu_0(s\text{-}bpp: \mu_0(<s\text{-}conv:T>,
<s\text{-}fofl:T>,
<s\text{-}ofl:T>,
<s\text{-}ufl:T>,
<s\text{-}zdiv:T>)) \)

Ref: int-program 3-5(5)

Note: The initial epilogue information has a component \( s\text{-}main \) indicating the main procedure of the program. This component is used by the return statement (cf. 5.6.4) to recognize whether the finish condition is to be raised. It is set to \( \Omega \) by all nested procedure calls (cf. 5.6.1 and 5.1.7).

(3) \( \text{valid-init-state}(\text{stg,es,time,date,tv,ev,funct-gen,t,call}) = 
\)
\( \text{is-mstg(stg)} \&
\text{is-es(es)} \&
\text{is-intg-val(time)} \& \text{is-intg-val(date)} \&
\text{is-allocated(tv,TASK,\text{stg})} \&
\text{is-allocated(ev,EVENT,\text{stg})} \& \text{is-O-DIT*\text{-}s\text{-}compl\text{*}op\text{*}val\text{*}gen\text{-}op}(ev,\text{stg}) \&
\text{(is-} \Omega \text{ (funct-gen)} \lor \text{is-e-ret-type*\text{-}s\text{-}\text{-}da(funct-gen)} \&
\text{is allocated(funct-gen,\text{-}s\text{-}\text{-}da(funct-gen),\text{stg})} \&
\text{is-program(t)} \&
\text{is-call-st(call)} \& \text{is-} \Omega \text{*s\text{-}\text{-}pa-opt(call)} \)

Ref.: is-mstg 2-26(93), is-es 2-38(120), op-val8-11(40), gen-op 7-19(47), is-e-ret-type 2-19(47),

Note: This predicate tests whether the parameters of the function initial-state are consistent, i.e. yield a valid initial state.
(4) is-allocated(gen,eda,stg) =
    is-gen(gen) & s-da(gen) = eda &
    ( \exists p,p-1) (p \in alloc-state \& el-rf(stg) & s-pp(gen) = p \cdot p-1) &
    enough-scalar-size(eda,size(s-pp(gen)(stg)))

for: is-gen(gen), is-scalar-eda(eda), is-mstg(stg)

Ref.: is-gen 2-27(96), alloc-state 2-23(73), el-rf 2-22(68),
      enough-scalar-size 6-12(23), size 2-21(59)

Note: This predicate tests whether gen is a valid generation with data
      attributes eda allocated in stg.

3.2 Initial Control

The control of the initial state consists of the instruction
int-program, which starts the interpretation of a program. It has two argu-
ments: the program t to be executed and the initial call statement call.

First, all external procedures, which are combined in the declaration
part and the body list of the program, are entered into the environment E,
into the attribute directory AT and, after the prepass has modified the body
list (cf. 3.3), into the denotation directory DN. These entries into the
directories, which initially were \emptyset (cf. 3.1), are performed in the same way
as it is normally done after establishing a new block activation (cf. 5.1).

Then the initial call statement call is executed. In the contrary to
a normal procedure call (cf. 5.6), no new block activation is established for
this call; instead only two local state components are updated by information
given in call.

Finally, the control is left with the instruction int-proc-body, which
starts regular flow of control of the PL/I machine (cf. 5.6.3).
(5) \texttt{int-program}(t, \text{call}) =
\begin{align*}
& \texttt{int-program-call}(\text{call}); \\
& \texttt{update-dn}(d_{pt}, bl); \\
& \quad \texttt{bl: prep-text}(s-\text{body-list}(t)); \\
& \quad \texttt{update-at}(s-\text{decl-part}(t)); \\
& \quad \texttt{update-env}(s-\text{decl-part}(t))
\end{align*}

where: \(d_{pt} = \mu(s-\text{decl-part}(t); \{<s-\text{scope.id:INT} \mid \text{is-decl.id} \cdot s-\text{decl-part}(t)\})\)

for: \text{is-program}(t), \text{is-call-st}(\text{call})

Ref.: \texttt{update-env} 5-5(4), \texttt{update-at} 5-6(7), \texttt{update-dn} 5-9(10), \texttt{prep-text} 3-7(8)

Note: In order to enter entry denotations into \(DN\) in the same way as it is normally done for internal entry declarations on block activation, the modified declaration part \(d_{pt}\) is used (cf. 5.1.5).

(6) \texttt{int-program-call}(\text{call}) =
\begin{align*}
& \texttt{int-proc-body}(s-\text{id}(\text{call}), s-\text{body}(d_{en}), \text{arg-list}); \\
& \texttt{update-program-state}(s-\text{id}(\text{call}), \text{arg-list}); \\
& \quad \text{arg-list: eval-arg-list}(s-\text{arg-list}(\text{call}), \star, \star, \Omega)
\end{align*}

where: \(d_{en} = \text{sel}(<s-\text{id}(\text{call})>)^{(E)}(\text{DN})\)

for: \text{is-call-st}(\text{call})

Ref.: \texttt{eval-arg-list} 5-36(58), \texttt{int-proc-body} 5-50(90)

(7) \texttt{update-program-state}(\text{id}, \text{arg-list}) =
\begin{align*}
& \texttt{s-el: }\mu(\mathcal{E}; <s-\text{free-set} : \text{free-set} (\text{arg-list})>) \\
& \quad \texttt{s-ce: }\mu(\mathcal{C}; <s-\text{entry} : s-\text{cbif}: id>)
\end{align*}

for: \text{is-id}(\text{id}), \text{is-arg-list}(\text{arg-list})

Ref.: \texttt{free-set} 5-33(54),

3.2
3.3 Prepass

The prepass modifies the text of the bodies of the external procedures before they are entered as parts of entry denotations into the denotation directory. It adds to each internal or external declaration of a static or controlled variable or file and to each external entry declaration a unique name as its denotation component. All external declarations of the same identifier get the same unique name, in order to link all external declarations of the same identifier. All internal declarations get distinct unique names. The external entry declarations get their unique names from the environment, in order to link them with their entries in the attribute directory AT and the denotation directory DN (cf. 3.2).

Besides this modification of text, the following initial actions are performed for all internal and external declarations of static and controlled variables and files:

For static variables, allocation of storage and initialisation is performed. If more than one declaration of the same external static variable occurs, their evaluated data attributes are tested for equality.

For controlled variables, the null generation is entered into the aggregate directory AG, and their attributes are entered into the attribute directory AT under their unique aggregate names. This information is used for testing of more than one external declaration of the same variable for matching and for testing of passed arguments against parameters of a procedure call (cf. 5.6.2.4).

For files essentially the same is done as for controlled variables, except the null allocation.

It should be noted that after the prepass all those parts of the external procedure bodies, which may contain declarations, need no more satisfy the predicates given in the abstract syntax of program text (cf. 12.2). Instead of that corresponding predicates for the modified text are defined. The names of these predicates start with the prefix is-p-... instead of is-... (e.g. is-p-block instead of is-block).
Metavariables

t is-body-list the list of external procedure bodies
decl is-decl a single declaration
id is-id the main identifier of decl
b is-n the unique denotation name given to decl
en (\{<id: is-n>\} \{is-id(id)\}) an auxiliary object containing a unique

un

name for each externally declared identifier of the program
an auxiliary object containing for each declaration \( \chi(t) \) its denotation component
as \( \chi(un) \)

(8) \[
\text{prep-text}(t) =
\]

\[
\text{prep-text-2}(t,un);
\]

\[
\text{un: prep-text-1}(t,en);
\]

\[
\{id(en): un-name \mid (\exists \chi-1)(is-EXT\cdot s\cdot \text{scope}\cdot id\cdot \chi-1(t)) \& \is\Omega (sel(<id>)(E)) \} \cup
\]

\[
\{id(en): pass(sel(<id>)(E)) \mid \neg is\in\Omega (sel(<id>)(E))\}
\]

Ref.: un-name 2-17(36)

(9) \[
\text{prep-text-1}(t,en) =
\]

\[
\text{pass}(un);
\]

\[
\{\% (un): \text{prep-decl}(id,\chi(t),id(en)) \mid is\cdot EXT\cdot s\cdot \text{scope}\cdot \chi(t) \& \is\Omega (sel(<id>)(E)) \} \cup
\]

\[
\{\% (un): \text{prep-decl}(id,\chi(t),b),\ b: un-name \mid is\cdot INT\cdot s\cdot \text{scope}\cdot \chi(t) \& (\exists \chi-1)(\chi = id\cdot \chi-1)\}
\]

Ref.: un-name 2-17(36).

(10) \[
\text{prep-decl}(id,decl,b) =
\]

\[
is\cdot \text{correct-static}(id,decl) \longrightarrow
\]

\[
pass(b);
\]

\[
\text{static-initialize}(id,decl,gen);
\]

\[
gen: \text{prep-static}(b,eda,s\cdot \text{dens}(decl));
\]

\[
edata: eval-da(s\cdot da(decl),\Omega)
\]

cont'd
is-CTL:s-stg-cl(decl) →

\textbf{pass}(b);

\textbf{prep-ctl}(decl,b)

\begin{align*}
is-\text{AUTO}:s-stg-cl(decl) & \& is-\text{INT}:s-scope(decl) \rightarrow \textbf{null} \\
is-file(decl) & \rightarrow \\
\textbf{pass}(b);

\textbf{prep-file}(decl,b)
\end{align*}

\begin{align*}
is-entry(decl) & \& is-\text{INT}:s-scope(decl) \rightarrow \text{PASS}:s-den(decl) \\
is-entry(decl) & \& is-\text{EXT}:s-scope(decl) \& is-\varnothing:s-den(decl) \rightarrow \text{PASS}:b
\end{align*}

T \rightarrow \textbf{error}

\textbf{Ref.: eval-da} 5-13(18)

\textit{Note:} Though a unique name \( b \) was created for each internal declaration in order to simplify the definition (8) above, for automatic variable and entry declarations these unique names are not used.

This instruction also tests that the program contains no automatic external declaration.

\begin{align*}
(11) \text{is-correct-static}(id,decl) = \\
is-\text{STATIC}:s-stg-cl(decl) & \& \\
(\forall \text{extent})(\text{extent} \in \text{extents-set}s-da(decl) \supset \text{is-intg-const}(\text{extent})) & \& \\
(\forall x)(x \in \text{s-init-set}(decl) \supset \text{head}s-id-lists-ref(x) = id & \\
\text{is-init-elem-list}s-init-spec(x) & \\
\& \\
\& (\forall x)(\neg \text{is-}\varnothing:s-rep-factor\%x(x) \supset \\
\text{is-intg-const}s-rep-factor\%x(x))
\end{align*}

\textbf{Ref.:} is-intg-const 5-41(69)
(12) \( \text{prep-static}(b, \text{eda}, \text{dens}) = \)
\[ \text{is-}\rho \ast b(\text{AT}) \quad \& \quad \text{is-}\rho \ast b(\text{AG}) \quad \& \quad \text{is-correct-eda}(\text{eda}) \quad \& \quad \]
\[ \text{is-mstg-applic}(p_1, S) \]
\[ \text{PASS}: \text{gen}_1 \]
\[ s-s: \text{el-alloc}(p_1, I, S) \]
\[ s-ag: (\text{AG}; \langle b; \langle \text{gen}_1, \text{NULL} \rangle \rangle) \]
\[ \text{is-}\theta \ast b(\text{AT}) \quad \& \quad \text{is-}\theta \ast b(\text{AG}) \quad \& \quad \text{is-correct-eda}(\text{eda}) \]
\[ \text{mstg-overflow} \]
\[ s-da(\text{gen}_b) = \text{eda} \quad \& \quad s-dens \ast \text{mi}(\text{gen}_b) = \text{dens} \quad \rightarrow \text{null} \]
\[ T \quad \rightarrow \quad \text{error} \]

where: \( p_1 = \text{alloc-space}(\text{eda}, \text{dens}, S) \)
\( \text{gen}_1 = \text{mk-gen}(\text{eda}, \text{dens}, I, S) \)
\( \text{gen}_b = \text{head-b}(\text{AG}) \)

for: \( \text{is-eda}(\text{eda}), \text{is-dens}(\text{dens}) \)

Ref.: \( \text{is-mstg-applic} \ 6-5(5), \text{is-correct-eda} \ 6-5(7), \text{el-alloc} \ 2-23(75), \)
\( \text{mstg-overflow} \ 6-5(6), \text{alloc-space} \ 6-11(19), \text{mk-gen} \ 6-5(8) \)

(13) \( \text{static-initialize}(\text{id}, \text{decl}, \text{gen}) = \)
\[ \neg \text{is-}\rho(\text{gen}) \quad \rightarrow \quad \text{initialize}(\text{id}, \text{s-init-set}(	ext{decl}), s-da(\text{decl}), \text{gen}, \rho) \]
\[ T \quad \rightarrow \quad \text{null} \]

for: \( \text{is-gen}(\text{gen}) \quad \neg \text{is-}\rho(\text{gen}) \)

Ref.: \( \text{initialize} \ 6-16(34) \)

Note: For external static declarations of the same identifier only one leads
to an allocation by \text{prep-static}. Only for this declaration \text{prep-static}
returns a generation, while for all other declarations \text{prep-static}
returns \( \rho \). So, the initial attribute of only one declaration is used for
initialization.

(14) \( \text{prep-ctl}(\text{decl}, b) = \)
\[ \text{is-}\rho \ast b(\text{AT}) \quad \& \quad \text{is-}\rho \ast b(\text{AG}) \]
\[ s-ag: \mu(\text{AG}; \langle b; \langle \text{NULL} \rangle \rangle) \]
\[ s-at: \mu(\text{AT}; \langle s-attr \ast b; \text{decl} \rangle) \]
\[ \text{ctl-match}(\text{decl}, s-attr \ast b(\text{AT})) \quad \rightarrow \quad \text{null} \]
\[ T \quad \rightarrow \quad \text{error} \]

cont'd
for: is-prop-var(decl) & is-ctl-s-stg-cl(decl)
Ref.:ctl-match 5-37(61)

(15) prep-file(decl,b) =
    is-\& b(AT) & is-\& b(AG) →
    s-at:\&μ(\&AT; s-attr:b:decl>)
    s-attr:b(AT) = decl → null
    T → error

for:is-file(decl)

(16) prep-text-2(t,un) =
    PASS:μ(t;\{<s-den•X;X(un)> | is-INT•s-scope•X(t) \& is-EXT•s-scope•X(t)\})

(17) is-p-[pred](t) =
    (\exists t-O,K)(t-O=\&t;\{s-den•X | X ∈ K\}) \& is-[pred](t-O) \&
    \[is-decl•X(t-O) \& is-n•s-den•X(t)\])

Note: This definition schema defines the following predicates is-p-[pred] for objects of the text, as modified by the prepass; thereby the corresponding predicates is-[pred] of the abstract syntax of program text are used.

<table>
<thead>
<tr>
<th>modified predicate:</th>
<th>predicate of abstract syntax:</th>
</tr>
</thead>
<tbody>
<tr>
<td>is-p-prop-var</td>
<td>is-prop-var</td>
</tr>
<tr>
<td>is-p-entry</td>
<td>is-entry</td>
</tr>
<tr>
<td>is-p-file</td>
<td>is-file</td>
</tr>
<tr>
<td>is-p-decl</td>
<td>is-decl</td>
</tr>
<tr>
<td>is-p-decl-part</td>
<td>is-decl-part</td>
</tr>
<tr>
<td>is-p-block</td>
<td>is-block</td>
</tr>
<tr>
<td>is-p-prop-st</td>
<td>is-prop-st</td>
</tr>
<tr>
<td>is-p-st</td>
<td>is-st</td>
</tr>
<tr>
<td>is-p-proc-st</td>
<td>is-proc-st</td>
</tr>
<tr>
<td>is-p-body</td>
<td>is-body</td>
</tr>
<tr>
<td>is-p-while-group</td>
<td>is-while-group</td>
</tr>
<tr>
<td>is-p-contr-group</td>
<td>is-contr-group</td>
</tr>
<tr>
<td>is-p-group</td>
<td>is-group</td>
</tr>
<tr>
<td>is-p-if-st</td>
<td>is-if-st</td>
</tr>
<tr>
<td>is-p-on-st</td>
<td>is-on-st</td>
</tr>
</tbody>
</table>
4. TASKS, INPUT AND OUTPUT EVENTS AND PROGRAM SYNCHRONIZATION

PL/I provides two means for specifying that parts of a program can be executed in parallel. A call statement may specify by a parallel action option that the called procedure is to be executed in parallel with the calling block, i.e. that a separate task is to be attached, which executes the called procedure. An input or output statement may specify by an event option that the data transmission may be performed in parallel, i.e. that an input or output event is established. The attaching of a task or input or output event has considerable logical consequences. In particular, if storage is shared by parallel tasks or input or output events, the reference to such storage by parallel actions potentially is undefined. In order to prevent undefinedness in such cases, parts of programs which otherwise could be executed in parallel, may be synchronized by the programmer explicitly by wait statements.

The present chapter describes the attaching, terminating and synchronizing of tasks and input and output events. This is done by a sequentialized model performing one action of one task after another action of possibly 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 of 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) may be represented in a concrete implementation by a series of uninterruptable actions which may be performed in parallel with, or intermixed with, actions in 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 or accessing state components of other tasks.

Section 4.1 describes the attaching, section 4.2 the termination of tasks. Section 4.3 describes the synchronization of tasks 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 4.4 describes the display statement which provides another means for synchronization, governed by the time. The attaching of input and output events is described in 10.4.5.
4.1 Attaching of Tasks

A task is attached by a call statement (cf. 5.6) containing a parallel action option pa-opt (cf. 12-11(84)). The actions of the call statement before the call (cf. 5.6.1), in particular the passing of arguments (cf. 5.6.2), are essentially the same as for normal call statements, except that no check conditions are raised for arguments. The actions specific for a task call are defined in this section by the instruction `int-task-call`, which is activated by the instruction `int-call` (5-50(89)).

Before the proper attaching of the new task, the following two actions are performed:

The task-event-specification TE of the new task is prepared by the instruction `mk-te`; in particular the generations of the associated task and event variables are evaluated or, if none are specified, dummy generations are allocated.

The priority of the new task is determined by the instruction `eval-pri` from the value of the priority expression or of the task variable, if specified in the call statement, and from the priority of the current task. The priority is evaluated as a binary integer operand with an implementation defined precision PRI-PREC and then transformed into a TASK operand so that its value representation can be assigned to a task variable.

The proper attaching of the new task is performed by the instruction `attach-task`, which enters the initial components of the task under a unique name ten into the parallel task and event part PA. The control of this task starts with the instruction `int-proc-body` (5-50(90)) which performs the same actions after a task call as are performed after a normal procedure call (cf. 5.6.3). At the same time `attach-task` tests whether the associated task and event variable generations are not yet associated with another task or input or output event and assigns the priority to the task variable generation and an initial value to the event variable generation.

The value of an event variable consists of two components: A bit denoting "incomplete" (O-BIT) or "complete" (1-BIT), and an integer value denoting the status of the event (if not explicitly assigned a different value, cf. 7-12(29)/7-16(38), 0 denotes "normal" and 1 "abnormal").
### Metavariables

<table>
<thead>
<tr>
<th>Id</th>
<th>Id</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>is-id</td>
<td>the called entry identifier</td>
</tr>
<tr>
<td>body</td>
<td>is-body</td>
<td>the called procedure body</td>
</tr>
<tr>
<td>arg-list</td>
<td>is-arg-list</td>
<td>the passed argument list of the procedure call (cf. 5.6.2)</td>
</tr>
<tr>
<td>env</td>
<td>is-e</td>
<td>the initial environment of the attached task</td>
</tr>
<tr>
<td>ei</td>
<td>is-ei</td>
<td>the initial epilogue information of the attached task</td>
</tr>
<tr>
<td>cs</td>
<td>is-cs</td>
<td>the initial condition status of the attached task</td>
</tr>
<tr>
<td>pa-opt</td>
<td>is-pa-opt</td>
<td>the parallel action option as occurring in the text of the call statement</td>
</tr>
<tr>
<td>ref</td>
<td>is-ref v is-*</td>
<td>the reference of the task or event variable in the call statement, if specified; * otherwise</td>
</tr>
<tr>
<td>gen</td>
<td>is-gen</td>
<td>the evaluated or allocated generation of the task or event variable</td>
</tr>
<tr>
<td>da</td>
<td>is-TASK v is-EVENT v is-PRI-DA</td>
<td></td>
</tr>
<tr>
<td>op</td>
<td>is-op</td>
<td>the priority operand to be assigned to the task variable or an intermediate operand for priority evaluation</td>
</tr>
<tr>
<td>te</td>
<td>is-te</td>
<td>the task-event specification TE of the new task</td>
</tr>
<tr>
<td>pa</td>
<td>is-pa</td>
<td>the complete parallel task and event part PA of the machine, to be inspected for active task or event variables</td>
</tr>
<tr>
<td>bit</td>
<td>is-bit-val</td>
<td>the completion value of an event variable: O-BIT denoting &quot;incomplete&quot;, 1-BIT denoting &quot;complete&quot;</td>
</tr>
<tr>
<td>i</td>
<td>is-intg-val</td>
<td>the status value of an event variable</td>
</tr>
</tbody>
</table>

(1) \[
\text{int-task-call}(id, body, arg-list, env, ei, cs, pa-opt) = \\
\text{attach-task}(id, body, arg-list, env, ei, cs, te, op); \\
\text{op:eval-pri}(op-1, te, s-task(pa-opt)); \\
\text{op-1:eval-pri-1}(s-pri(pa-opt)), \\
\text{te:mk-te}(pa-opt)
\]
(2) \[ \text{mk-te(pa-opt)} = \]
\[
\begin{align*}
\text{pass(te);} \\
\text{s-ten(te)} : \text{un-name} \\
\text{s-tv(te)} : \text{eval-te-gen(s-task(pa-opt),TASK)} \\
\text{s-ev(te)} : \text{eval-te-gen(s-event(pa-opt),EVENT)} \\
\text{s-io-ev(te)} : \text{pass(\{\}} \\
\text{s-based-free-set(te)} : \text{pass(\{\}}
\end{align*}
\]
Ref.: un-name 2-17(36)

(3) \[ \text{eval-te-gen(ref,da)} = \]
\[
\begin{align*}
\text{is-ref(ref)} & \quad \rightarrow \\
\text{test-gen(gen,da);} \\
\text{gen} & : \text{eval-ref-gen(ref,E)} \\
\text{T} & \quad \rightarrow \\
\text{auto-allocate(b,da,PACKED);} \\
\text{b} & : \text{un-name}
\end{align*}
\]
Ref.: eval-ref-gen 7-25(59), auto-allocate 5-11(13), un-name 2-17(36)

Note: If no task or event variable is specified, a dummy generation is allocated. This dummy generation is freed automatically in the epilogue of the calling block like automatic variables (cf. auto-free 5-15(22)).

(4) \[ \text{test-gen(gen,da)} = \]
\[
\begin{align*}
da & = \text{s-da(gen)} \quad \rightarrow \quad \text{PASS:gen} \\
\text{T} & \quad \rightarrow \quad \text{error}
\end{align*}
\]

(5) \[ \text{eval-pri-l(expr)} = \]
\[
\begin{align*}
\text{-is-}\Omega (expr) & \quad \rightarrow \\
\text{convert(PRI-DA,op);} \\
\text{op} & : \text{eval-expr(expr,E)} \\
\text{T} & \quad \rightarrow \quad \text{null}
\end{align*}
\]
for : is-opt-expr(expr)
Ref.: eval-expr 7-18(43), convert 8-30(118)

cont'd
Note: This instruction has nearly the same effect as the instruction `eval-intg-expr`, with the difference that conversion is performed to an integer with a special implementation defined precision. Note that `null` returns \( \emptyset \).

(6) \( \text{PRI-DA} = \) 
\[ \text{bintg-da(PRI-PREC)} \]
Ref.: bintg-da 8-12(51)
Note: PRI-PREC is an implementation defined integer, i.e. satisfying `is-intg-val`, the default precision for priority values.

(7) \( \text{eval-pri}(op,te,ref) = \) 
\[ \begin{align*}
\text{is-op}(op) & \rightarrow \text{eval-pri-2}(op) \\
\text{is-ref}(ref) & \rightarrow \text{pass-gen-op}(s-tv(te),S) \\
T & \rightarrow \text{pass-gen-op}(s-tv(TE),\emptyset)
\end{align*} \]
for: \( \text{is-op}(op) \) \( \not\in \emptyset(op) \)
Ref.: gen-op 7-19(47), is-op 8-11(38)

(8) \( \text{eval-pri-2}(op) = \) 
\[ \begin{align*}
\text{convert-2}(\text{TASK},op-3); \\
op-3:\text{convert}(\text{PRI-DA},op-2); \\
op-2:\text{eval-infix-expr}(op,op-1,\text{ADD}); \\
op-1:\text{convert-2}(\text{PRI-DA},\text{gen-op}(s-tv(TE),\emptyset))
\end{align*} \]
Ref.: gen-op 7-19(47), eval-infix-expr 8-14(65), convert 8-30(118)

(9) \( \text{convert-2}(da,op) = \) 
\[ \text{PASS:val-op}(da,op-val(op)) \]
Ref.: op-val 8-11(40), val-op 8-11(41)
Note: This instruction is used to transform operands with PRI-DA data attributes (i.e. integer operands) into operands with TASK data attributes, and vice versa. Both types of operands have the same values (namely integers) but possibly different value representations. The former are necessary for arithmetic operations, the latter for assignment to task variables. The normal conversion between both operands is not possible (cf. 8-31(120)).
(10) \texttt{attach-task}(id, \texttt{body}, \texttt{arg-list}, \texttt{env}, ei, cs, te, op) = \\
\neg \text{is-active}(s-tv(te), PA) \& \neg \text{is-active}(s-ev(te), PA) \& \\
is\text{-indep}-1(s-pp+s-tv(te), s-pp+s-ev(te)) \\
\texttt{s-s}_{:} \text{el-ass}(s-pp_s-tv(te), s-vr(op), \\
el-ass(s-pp+s-ev(te), ev-vr(O-BIT, O, 0))) \\
\texttt{s-ag}_{:}\delta(AG; \{b \mid (\exists i)(b=s-den(arg_i) \& \text{is-DUMMY}\_s-type(arg_i)\}) \\
\texttt{s-ei}_:\mu(EI; <s-task-set:s-task-set(\overline{FI}) \cup \{s-ten(te)\}> \\
\texttt{s-pa}_:\mu(PA; <s-ten(te)> \\
\mu_o(<s-te;te>, \\
\texttt{s-ag}_{ten}_, \\
\texttt{s-fd}_{ten}, \\
\texttt{s-e};env, \\
\texttt{s-ei};ei, \\
\texttt{s-cs};cs, \\
\texttt{s-d};\mu_o(<s-cs;cs>), \\
\texttt{s-c};\texttt{int}-proc-body(id, \texttt{body}, \texttt{arg-list}>)>) \\
\text{T} \rightarrow \text{error} \\
\
\text{where: ag}_{ten} = \mu(\mu_o\{<b:\text{head}\_b(AG)>, \neg \text{is-}\_\Omega\_b(AG)\} \\
\{<b:b(AG) \mid (\exists i)(b=s-den(arg_i) \& \text{is-DUMMY}\_s-type(arg_i)\}) \\
arg_i = \text{elem}(i, \text{arg-list}) \\
\text{fd}_{ten} = \mu(\texttt{FD}; \{<s\_orig\_s;INH, \neg \text{is-}\_\Omega\_s(FD)\}) \\

\text{Ref.: el-ass 2-22(70), int-vec-body 5-50(90), is\_indep-1 2-22(67)} \\

\text{Note: This instruction performs the proper attaching of the new task.} \\
\text{At the same time it assigns the priority value to the task variable} \\
\text{and the incomplete-normal-value to the event variable and enters} \\
\text{the task-event-name ten of the new task into the epilogue information} \\
\text{of the current block, causing thereby that the new task is} \\
\text{terminated at block end if it does not terminate earlier.} \\

\text{The state components TE, E, EI, CS of the new task are as prepared} \\
\text{before and passed as arguments of attach-task. The aggregate directory AG} \\
\text{consists of the heads of the aggregate directory entries of the} \\
\text{current task (which may not be freed by the new task, cf. 6-19(42)).} \\
\text{Only for the passed dummy arguments the complete entries, consisting} \\
\text{of their proper generations and the NULL generation are taken over} \\
\text{in order to free them at task end (these entries are deleted from} \\

4.1
the aggregate directory of the current task in order to avoid another freeing). The file directory $FD$ is the same as that of the current task with changed $s$-orig components to avoid closing of not owned files. The dump $D$ consists only of a $s$-cs component, in order to have this information available for the revert statement.

The control of the new task starts with normal execution of the called procedure body (cf. 5.6.3).

(11) $\text{is-active}(\text{gen,pa}) =$

$\exists \text{ten} (\neg \text{is-indep-1}(\text{pp}(\text{gen}), \text{pp}\cdot \text{tv}\cdot \text{te*ten}(\text{pa})) \lor$

$\neg \text{is-indep-1}(\text{pp}(\text{gen}), \text{pp}\cdot \text{ev}\cdot \text{te*ten}(\text{pa})))$

Ref.: is-indep-1 2-22(67)

Note: This relation tests whether a given generation is the generation of a task or event variable, which has already been associated with a task or input or output event, or even if it has overlapping storage with such a task or event variable.

(12) $\text{ev-vr}($bit,$i) =$

$\text{represent}(\text{EVENT}, \mu^{\text{bit}}(\text{s-compl:bit},$

$\text{s-status:i}))$

Ref.: represent 8-9(33)

(13) $\text{is-event-val} =$

$\langle \text{s-compl:is-bit-val},$

$\text{s-status:is-intg-val} \rangle$

Ref.: is-bit-val 8-5(14)
4.2 Termination of tasks

A normal termination of a task is caused by exhaustion of its state­­ment list or by a return statement (cf. 5.6.4, in particular the instruc­­tion return 5-53(98)). In this case all nested block activations have al­­ready been terminated and the instruction task-epilogue described below is executed.

An abnormal termination of a task is caused by a stop or exit state­­ment or by termination of the block activation in which the task was at­­tached. In this case all blocks of the task currently active are terminated one by one by task exit. The termination of each block involves terminating all tasks attached in it by term-tasks and the termination of the last, i.e. outermost, active block involves terminating the current task by task-epilogue.

The task termination, performed by the instruction task-epilogue, consists of: Termination of all tasks attached during its outermost block activation, closing of all files opened by the task, termination of all input and output events started by the task, freeing of all storage allocated by the task and finally setting the associated event variable generation to complete and deletion of the task from PA.

Task termination, whether normal or abnormal, is performed by the instructions epilogue, task-epilogue, term-tasks, task-exit activating each other mutually. This algorithm guarantees that each block or task which terminates causes all tasks attached by it to terminate abnormally and there­­by themselves to cause all their descendent tasks to terminate abnormally. The instruction term-task-wait ensures that no block or task terminates fully until all tasks attached by it have themselves terminated. By this al­­gorithm termination of the main task means successive termination of all tasks of the program, which is used by the stop statement.

Metavaraibles

\[
\begin{align*}
\text{status} & \quad \text{is-NORMAL } \lor \text{is-ABNL} \\
\text{b} & \quad \text{is-n} \\
\end{align*}
\]

completion status of a task or event aggregate name of a proper variable
(14) \texttt{int-stop-st} =
\begin{verbatim}
program-stop;
call-cond(FINISH, 201)
\end{verbatim}
Ref.: \texttt{call-cond 9-11(18)}

(15) \texttt{program-stop} =
\begin{verbatim}
is-s-main\cdot s\cdot ten(TE) \rightarrow task-exit
T \rightarrow s-pa; \mu(\text{PA}; \langle s\cdot c\cdot s\cdot main\cdot task-exit\rangle)
\end{verbatim}
Note: The case distinction is necessary since by the mechanism described in 2.1.6 changes performed in \texttt{PA} for the current task would be overwritten when recopying the current task into \texttt{PA}.

(16) \texttt{int-exit-st} =
\begin{verbatim}
is-s-main\cdot s\cdot ten(TE) \rightarrow
\text{task-exit;}
call-cond(FINISH, 202)
T \rightarrow \text{task-exit}
\end{verbatim}
Ref.: \texttt{call-cond 9-11(18)}

(17) \texttt{task-exit} =
\begin{verbatim}
is\cdot c\cdot s\cdot c(D) \rightarrow
s-d; \mu(D; \langle s\cdot c\cdot task-exit\rangle)
s-c: \text{epilogue}
T \rightarrow \text{task-epilogue(ABNL)}
\end{verbatim}
Ref.: \texttt{epilogue 5-14(20)}

(18) \texttt{task-epilogue(status)} =
\begin{verbatim}
delete-task(status);
free-task;
term-tasks-wait;
term-tasks,
task-fd-close,
term-events,
unlock-taskend
\end{verbatim}
(19) \[ \text{term-tasks} = \]
\[ \{s-pa : \mu(\text{PA}; \{<s-\text{cten:task-exit}> \mid \text{ten} \in \text{active-descendants}(EI, PA)\}) \}
\]

(20) \[ \text{active-descendants}(ei, pa) = \]
\[ \{\text{ten} \mid \text{ten} \in \text{s-task-set}(ei) \land \neg \text{is-}\Omega\cdot\text{ten}(pa)\}\]

for : \text{is-ei}(ei), \text{is-pa}(pa)

Note: This function collects all tasks attached by the current block activation which are not yet terminated, if \( ei = EI \) and \( pa = PA \).

(21) \[ \text{term-tasks-wait} = \]
\[ \text{is-}\{\} \cdot \text{active-descendants}(EI, PA) \rightarrow \text{null} \]
\[ T 
\]
\[ \text{term-tasks-wait;} \]
\[ \text{set-wait-state-1} \]

Note: This instruction causes the current task to wait until all tasks whose termination was started by term-tasks have completely been terminated. For the waiting mechanism cf. 4.3.1.

(22) \[ \text{set-wait-state-1} = \]
\[ \text{is-}\{\} \cdot \text{active-descendants}(EI, PA) \rightarrow \text{null} \]
\[ T 
\]
\[ s-te: \mu(\text{TE}; <s-wait:T>) \]

Note: Cf. the instruction set-wait-state 4-18(43)

(23) \[ \text{task-fd-close} = \]
\[ \text{null;} \]
\[ \{\text{close}(f) \mid \text{is-OWN\cdot s-orig\cdot FD}\} \]

Ref.: close 10-22(55)

(24) \[ \text{term-events} = \]
\[ \text{null;} \]
\[ \{\text{delete-event}(\text{ten}, \text{ABNL}) \mid \text{ten} \in \text{s-10-ev}(\text{TE}) \land \neg \text{is-}\Omega\cdot\text{ten}(PA)\}\]

cont'd
Ref.: delete-event 4-20(49)

Note: This instruction deletes all input and output events started by the current task (cf. attach-io-event, 10-32(89)) which have not been terminated before.

(25) unlock-taskend =
    \( s_{fu} : \delta(FU; \{ (s-ten(TE)) \cdot u \mid \neg is-\Omega \circ u(FU) \}) \)
    \( s_{pa} : activate-tasks(PA) \)

Ref.: activate-tasks 4-18(44)

Note: An instruction wait-for-unlock (10-35(96)) in any other task may be reactivated by this instruction.

(26) free-task =
    null;
    free-ag,
    free-based-set

(27) free-ag =
    null;
    \{ iterate-free(b) \mid \neg is-\Omega \circ b(AG) \}

(28) iterate-free(b) =
    length \circ b(AG) = 1 \rightarrow \text{null}
    T \rightarrow
    iterate-free(b);
    free(b)

Ref.: free 6-19(42)

(29) free-based-set =
    \( s_{-s} : \text{el-free}(s-based-free-set(TE), I, S) \)
    \( s_{-te}: \mu(TE; :s-based-free-set: \{ \}) \)
(30) \( \text{delete-task}(\text{status}) = \)
\[
\begin{align*}
\text{s-s} &: \text{el-ass}(\text{s-pp} \cdot \text{s-ev(TE)}, \text{ev-vr(1-BIT, i_{status}}), \mathbb{S}) \\
\text{s-et} &: <\text{s-ev(TE)} > \bigcirc \text{ET} \\
\text{s-pa} &: \text{activate-tasks(PA)} \\
\text{s-te} &: \Omega \\
\text{s-ag} &: \Omega \\
\text{s-fd} &: \Omega \\
\text{s-e} &: \Omega \\
\text{s-ei} &: \Omega \\
\text{s-cs} &: \Omega \\
\text{s-d} &: \Omega \\
\end{align*}
\]

where: \( i_{status} = (\text{is-ABNL(status)} \& \text{s-status} \cdot \text{op-val} \cdot \text{gen-op}(\text{s-ev(TE)}, \mathbb{S}))=0 \rightarrow 1, \)
\( T \rightarrow s-\text{status} \cdot \text{op-val} \cdot \text{gen-op}(\text{s-ev(TE)}, \mathbb{S})) \)

Ref.: el-ass 2-22(70), op-val 8-11(40), ev-vr 4-7(12),
\( \text{gen-op} 7-19(47), \text{activate-tasks} 4-18(44) \)

Note: This instruction finally terminates the current task by deleting all its state components from \( \text{PA} \). Additionally it sets the completion value of the associated event variable generation to 1-BIT ("complete") and its status value to 1, if the termination is abnormal and the status value is yet 0. The event variable generation is entered into the event trace \( \text{ET} \) for use by any wait statements, and all waiting tasks are reactivated (cf. 4.3).
4.3 The Wait Statement

The wait statement is a means to synchronize the actions of different tasks or input or output events. By a wait statement a programmer may specify that the current task shall not continue its actions before a certain number out of a given list of event variable generations have been set complete, usually by other tasks. This complete setting may be performed either by an explicit assignment statement to the event variable (cf. 7.1.4 and 7.1.6) or automatically by normal or abnormal completion of a task or input or output event associated with the event variable generation (cf. 4.2 and 4.3.2).

If an event variable in the list given with the wait statement 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 generation but the test is for the completion of the actions of the associated input or output event, i.e. the completion of its data transmission. 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 and output statement, including the setting to complete of the event variable generation. This language feature is necessary in order that these actions, and their possible side effects, are performed at a well-defined point of time by the current task and not at an unpredictable point of time by the input or output event.

Section 4.3.1 describes the wait statement proper, while section 4.3.2 describes the special actions necessary for completion of semi-complete input and output events.

4.3.1 The waiting mechanism

The interpretation of a wait statement first results in evaluation of the generation of each scalar event variable specified or implied in the list and evaluation of the count option if present. If the count option is not present the number of events to be waited for is the total number of scalar event variable generations.
Next the event variable generations are placed in a list in unspecified order. This list is scanned, a test being carried out on each generation; the test is whether either the event variable generation is complete or whether it is associated with an I/O event which is semi-complete. Each semi-complete I/O event encountered in this way is completed as described in 4.3.2. Each recognized complete or semi-complete event variable generation results in a reduction by one of the number of events to be waited for.

When the whole list has been scanned the instruction \texttt{int-wait} is activated which sets the current task into the wait state until a relevant event completes or semi-completes. When this happens the above actions are carried out again but this time with a possibly shorter list of event variable generations. The process repeats itself until the list becomes empty.

Each task has in its \texttt{TE} state component a "wait state flag" \texttt{s-wait(TE)} which is either \texttt{Q} ("active") or \texttt{T} ("waiting"). The priority scheduler, i.e. the function \texttt{pre-sched} described in 2.1.6, selects only such tasks for execution whose wait state flag is \texttt{Q}. Each wait type instruction, e.g. \texttt{int-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, i.e. sets the wait state flag to \texttt{T}. Each action which possibly may cause a waiting condition of any wait type instruction of any task to be satisfied, e.g. complete setting of an event variable, resets the wait state flags of all tasks to \texttt{Q}. Then governed by priority selection, each wait type 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 variables which are complete at the time when they are inspected by the scan of the wait statement, but also those which are set complete and possibly reset to incomplete during the wait statement, there is a special global state component, the event trace \texttt{ET}. It may be considered as a time axis in which the points in time of complete setting of event variable generations and of starts of execution of wait statements are recorded. The event trace is a list on top of which each instruction setting complete an event variable generation enters this generation. Each starting wait statement enters a unique name on top of \texttt{ET} and may later ask whether in front of this unique name in \texttt{ET} one of its event variable generations occurs, i.e. whether one of them is completed later than the start of the wait statement.
Metavariables

\( t \) \hspace{1cm} \text{is-wait-st} \hspace{1cm} \text{the text of the wait statement}

\( w \) \hspace{1cm} \text{is-n} \hspace{1cm} \text{the unique name of the wait statement entered into ET}

\( \text{count} \) \hspace{1cm} \text{is-intg-val} \hspace{1cm} \text{the number of events waited for}

\( \text{evl} \) \hspace{1cm} \text{is-gen-list} \hspace{1cm} \text{list of scalar event variable generations, for whose completion is waited}

\( \text{genl} \) \hspace{1cm} \text{is-gen-list} \hspace{1cm} \text{list of, scalar or non-scalar, generations}

\( \text{ev} \) \hspace{1cm} \text{is-gen} \hspace{1cm} \text{generation of an event variable}

\( \text{pa} \) \hspace{1cm} \text{is-pa} \hspace{1cm} \text{parallel action state component PA}

\( \text{te} \) \hspace{1cm} \text{is-te} \hspace{1cm} \text{task or event specification TE}

\( \text{et} \) \hspace{1cm} \text{is-et} \hspace{1cm} \text{event trace ET}

(31) \hspace{5mm} \text{int-wait-st}(t) =
\hspace{5mm} \text{int-wait-st-l}(t,w);
\hspace{5mm} \text{trace-wait-st}(w);
\hspace{5mm} w:\text{name}

Ref.: \text{name} 2-17(36)

(32) \hspace{5mm} \text{trace-wait-st}(w) =
\hspace{5mm} s-\text{set}:(w) \cap \text{ET}

(33) \hspace{5mm} \text{int-wait-st-l}(t,w) =
\hspace{5mm} \text{int-wait-l}(\text{count},\text{evl},w);
\hspace{5mm} \text{count}:\text{pass-wait-count}(i,\text{evl});
\hspace{5mm} i:\text{eval-intg-exp}(s\text{-event-number}(t),E),
\hspace{5mm} \text{evl}:\text{test-event-list}(%\text{genl-1});
\hspace{5mm} \text{genl-1}:\text{conc-gen-list}(%\text{genl});
\hspace{5mm} \{\text{elem}(i)(%\text{genl})\cdot\text{eval-ref-gen}(\text{event}_{i},E) | i\text{islength}\cdot s\text{-event-list}(t)\}

where: \text{event}_{i} = \text{elem}(i,s\text{-event-list}(t))

Ref.: \text{eval-ref-gen} 7-25(59), \text{eval-intg-exp} 7-21(52)
(34) \[
\text{conc-gen-list}(\text{genl}) = \\
\quad \text{length}(\text{genl}) \\
\quad \text{PASS: CONC gen-list : elem}(i, \text{genl}) \\
\quad i = 1 \\
\]

Ref.: gen-list 6-17(39)
Note: This instruction expands the list of arbitrary generations genl into a list of scalar generations.

(35) \[
\text{test-event-list}(\text{genl}) = \\
\quad \text{pass}(\text{evl}) ; \\
\quad \{\text{elem}(i)(\text{evl}) : \text{test-gen}(\text{elem}(i, \text{genl}), \text{EVENT}) \mid 1 \leq \text{length}(\text{genl})\} \\
\]

Ref.: test-gen 4-4(4)

(36) \[
\text{wait-count}(i, \text{evl}) = \\
\quad \text{is-} \Omega(i) \vee i > \text{length}(\text{evl}) \rightarrow \text{length}(\text{evl}) \\
\quad T \rightarrow i \\
\quad \text{for: is-} \text{intg-val}(i) \vee \text{is-} \Omega(i) \\
\]

(37) \[
\text{int-wait-1}(\text{count, evl, w}) = \\
\quad \text{int-wait-2}(\text{count, permute(\text{evl})}, <>, w) \\
\]

(38) \[
\text{permute}(\text{list}) = \\
\quad \text{for: is-list(\text{list})} \\
\]
Note: The value of this function is a list consisting of the same elements which constitute the argument list but in some unspecified order. The function could actually be defined by an instruction which makes use of the unspecified order of the activation of successor instructions.

(39) \[
\text{int-wait-2}(\text{count, evl-1, evl-2, w}) = \\
\quad \text{counts}0 \rightarrow \text{null} \\
\quad \text{is-} <> (\text{evl-1}) \rightarrow \text{int-wait}(\text{count, evl-2, w}) \\
\quad \text{is-1-BIT-s-compl-op-val-gen-op(evl-1, 2)} \vee \text{is-compl(evl-1, w, ET)} \rightarrow \\
\quad \text{int-wait-2}(\text{count - 1, tail(\text{evl-1}), evl-2, w}) \\
\]

cont'd
is-semi-compl(ev₁,PA,TE) →

int-wait-2(count-1,tail(evl-1),evl-2,w);
compl-event(ten₁)

T →

int-wait-2(count,tail(evl-1),<ev₁>₁,evl-2,w)

where: ev₁ = head(evl-1)

ten₁ = (λten)(ev₁ = s-ev·s-te·ten(PA))

Ref.: op-val 8-11(40), gen-op 7-19(47), compl-event 4-19(45).

Note: This instruction scans the list of event variable generations evl-1 one by one until either enough complete or semi-complete ones have been found or the list is exhausted. It removes the complete or semi-complete generations from the list, completing the semi-complete ones, and collects the remaining list in evl-2. The integer count counts down the number of events yet to be waited for.

(40) is-compl(ev,w,et) =

(∃i)(ev = elem(i,et) & i≤(l,k)(w = elem(k,et)))

Note: This function, applied to et = ET, tests whether the event variable generation ev has been set to complete later than the start of the wait statement. It would be insufficient to merely test the current completion value of ev.

(41) is-semi-compl(ev,pa,te) =

(∃ten)(ev = s-ev·s-te·ten(pa) & is-Ω·s-c·ten(pa) & ten ∈ s-io-ev(te))

Note: This function, applied to pa = PA, te = TE, tests whether the event variable generation ev is associated with a semi-complete input or output event started by the current task.

(42) int-wait(count,evl,w) =

(∃j)(is-compl(evj,w,ET) ∨ is-semi-compl(evj,PA,TE)) →

int-wait-1(count,evl,w)

T →

int-wait(count,evl,w);
set-wait-state(evl,w)

where: evj = elem(j,evl)

Note: This instruction performs the proper waiting. Depending on the test
whether there is a complete or semi-complete event in evl, it either starts the scanning of evl anew, or it inserts itself back into the control and sets the current task into the wait state. When, by replacing PA by activate-tasks(PA), the task is reactivated, int-wait performs the test again until finally there is a complete or semi-complete event in evl.

(43) \[ \text{set-wait-state}(\text{evl},\text{w}) = \]
\[ (\exists j)(\text{is-compl}(\text{ev}_j, \text{w}, \text{ET}) \lor \text{is-semi-compl}(\text{ev}_j, \text{PA}, \text{TE})) \rightarrow \text{null} \]
\[ \text{s-te;u}(\text{TE};<\text{s-wait};\text{T}>) \]

where: \( \text{ev}_j = \text{elem}(j, \text{evl}) \)

Note: This instruction in general sets the current task actually into the wait state. The test, which had already been performed by int-wait, is necessary since between insertion of set-wait-state into the control by int-wait and execution of set-wait-state another task might have satisfied the waiting condition, i.e. completed or semi-completed an event out of evl. In this case, of course, the task should no more wait.

(44) \[ \text{activate-tasks}(\text{pa}) = \]
\[ d(\text{pa};\{\text{s-wait};\text{e-ten} \mid \neg \text{is-} \Omega \cdot \text{ten}(\text{pa})\}) \]

Note: Each instruction for which another task might wait, e.g. complete setting of an event variable, termination of a task, unlocking of a record, replaces PA by activate-tasks(PA) thereby reactivating all waiting tasks.

4.3.2 Completion of input and output events

A semi-complete input or output event, i.e. one whose data transmission has been terminated but which has not been deleted from PA, is completed during the corresponding wait statement by the instruction compl-event. This instruction performs raising of input-output conditions accumulated by the event during data transmission, raising of check-condition in case of input, setting of the associated event variable generation, unlocking of the transmitted record if necessary and deletion of the input or output event from the state.
Since the last three actions (complete setting of event variable generation, unlocking and deleting of input or output event) have also to be performed, if a goto leads abnormally out of a raised on unit, the task-event name ten of the input or output event is passed into the cbif-component of CS of the called on units (cf. 5-28(47)). The goto statement in this case activates **compl-event-1**.

**Metavariables**

- ten is-n task-event-name of the completed input or output event
- cond-list is-io-cond-arg-list list of input-output condition arguments collected during the event
- status is-NORMAL v is-ABNL

(45) \[ \textbf{compl-event}(\text{ten}) = \]
\[ \text{call-check-cond}(\text{permute}(\text{check-list}_{\text{ten}}),\text{env}_{\text{ten}},203) ; \]
\[ \text{compl-event-1}(\text{ten}) ; \]
\[ \text{io-cond-iterate}(\text{ten}) \]

where: \( \text{check-list}_{\text{ten}} = \text{s-ref-list} (\text{check}_{\text{ten}}) \)
\( \text{env}_{\text{ten}} = \text{s-e} (\text{check}_{\text{ten}}) \)
\( \text{check}_{\text{ten}} = \text{s-check-list-s-te\_ten(PA)} \)

Ref. : call-check-cond 9-15(27), permute 4-16(38)

(46) \[ \textbf{io-cond-iterate}(\text{ten}) = \]
\[ \text{is-<>}(\text{cond-list}_{\text{ten}}) \rightarrow \text{null} \]
\[ T \rightarrow \]
\[ \text{io-cond-iterate-1}(\text{permute}(\text{cond-list}_{\text{ten}}),\text{ten}) ; \]
\[ \text{assign-status}(\text{ev}_{\text{ten}},\text{bintg-op(1)}) \]

where: \( \text{cond-list}_{\text{ten}} = \text{s-cond-list-s-te\_ten(PA)} \)
\( \text{ev}_{\text{ten}} = \text{s-ev\_s-te\_ten(PA)} \)

Ref. : permute 4-16(38), assign-status 7-16(38), bintg-op 8-11(43)

Note: If no conditions are to be raised, no status value is assigned to the event variable generation (normal completion); otherwise the status value 1 is assigned before condition raising (default value for abnormal completion).
(47) \( \text{io-cond-iterate-1} \text{(cond-list, ten)} = \)

\[
\begin{align*}
\text{is-} & \rightarrow (\text{cond-list}) \rightarrow \text{null} \\
T & \rightarrow \\
o & \rightarrow \\
\text{io-cond-iterate-1} & \text{(tail(cond-list), ten);} \\
\text{call-cond-1} & \text{(s-cond(cond\(_1\)), \Omega, cbif\(_1\), r-type(cond\(_1\)))}
\end{align*}
\]

where: \( \text{cond\(_1\)} = \text{head} \text{(cond-list)} \)

\( \text{cbif\(_1\)} = \mu(s\text{-cbif(cond\(_1\))}, <\text{oncount: length} \text{(cond-list)}, <\text{ten:ten}>) \)

Ref.: \text{call-cond-1 9-12(19)}

(48) \( \text{compl-event-1} \text{(ten)} = \)

\[
\begin{align*}
\text{null}; \\
\text{delete-event} & \text{(ten, NORMAL),} \\
\text{int-unlock-1} & \text{(u\text{ten}, opt\text{ten})}
\end{align*}
\]

where: \( u\text{ten} = s\text{-u} \text{(unlock\text{ten})} \)

\( \text{opt\text{ten}} = s\text{-rec-opt} \text{(unlock\text{ten})} \)

\( \text{unlock\text{ten}} = s\text{-unlock s-te} \text{ten(PA)} \)

Ref.: \text{int-unlock-1 10-34(94)}

(49) \( \text{delete-event} \text{(ten, status)} = \)

\[
\begin{align*}
\text{s-s: el-ass} & \text{(s-pp(ev\text{ten}), ev-vr(l-BIT, i\text{status}), S)} \\
\text{s-et:} & \text{<ev\text{ten}, \text{et}}} \\
\text{s-pa: activate-tasks} & \text{(PA; ten))}
\end{align*}
\]

where: \( ev\text{ten} = s\text{-ev s-te} \text{ten(PA)} \)

\( \text{i\text{status}} = (\text{is-ABNL} \text{(status)} \rightarrow 1 \)

\( T \rightarrow s\text{-status op-val gen-op(ev\text{ten}, S)} \)

Ref.: \text{el-ass2-22(70), ev-vr 4-7(12), op-val 8-11(40), gen-op 7-19(47), activate-tasks 4-18(44)}

Note: If \( \text{delete-event} \) is activated by \text{task-epilogue} (cf. \text{4-9(18)/4-10(24)}) the status value 1(\text{abnormal}) is assigned to the event variable generation; if it is activated by \text{compl-event} the correct status value has already been set and is left unchanged.
4.4 The Delay Statement

The execution of a delay statement results in a time delay for continuation of execution of the current task. The time, up to which the task has to wait, is noted in a special state component s-delay(TD) and the current task is set into the wait state until this time has been reached. The time updating function, cf. 2.1.6, inspects the component s-delay(TD) at each instruction execution andreactivates all waiting tasks automatically whenever the time crosses one of the entries in s-delay(TD).

For the waiting mechanism compare section 4.3.1.

(50) \text{int-delay-st}(t) =
    \begin{align*}
    \text{wait-delay}(\text{time}); \\
    \text{time} : \text{note-delay}(i); \\
    i : \text{eval-intq-expr}(s\text{-time}(t),E)
    \end{align*}

\text{for: is-delay-st}(t)
\text{Ref.: eval-intq-expr 7-21(52)}

(51) \text{note-delay}(i) =
    \begin{align*}
    \text{PASS: } \text{time}_{i} \\
    \text{s-td} ; \mu(\text{TD}; <\text{s-delay}; s\text{-delay(TD)} \cup \{\text{time}_{i}\} >) \\
    \end{align*}

\text{where: } \text{time}_{i} = i + s\text{-time(TD)}
\text{for: is-intg-val}(i)

(52) \text{wait-delay}(\text{time}) =
    \begin{align*}
    \text{time} > s\text{-time(TD)} & \rightarrow \text{null} \\
    T & \rightarrow \\
    \text{wait-delay}(\text{time}); \\
    \text{set-wait-state-2}(\text{time})
    \end{align*}

\text{for: is-intg-val}(\text{time})

(53) \text{set-wait-state-2}(\text{time}) =
    \begin{align*}
    \text{time} > s\text{-time(TD)} & \rightarrow \text{null} \\
    T & \rightarrow \\
    s\text{-te}; \mu(\text{TE}; <\text{s-wait}; T>)
    \end{align*}

\text{for: is-intg-val}(\text{time})
\text{Note: Cf. the instruction } \text{set-wait-state, 4-18(43).}
5. FLOW OF CONTROL WITHIN A SINGLE TASK

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 one task. Excepted are only the activation and termination of on-units which interrupt the normal flow on occurrence of exceptional conditions (cf.9.4). Included are all those actions which, though not directly concerned with the flow of control, are performed by the PL/I statements described here, especially the installation of declarations at block entry and argument passing at procedure call.

The first two sections define the basic mechanism of the flow of control: section 5.1 the dynamics of nested blocks, and section 5.2 the sequential interpretation of statement lists within one block. In this section control is passed to the interpretation of the single simple statements which, depending on their type, is defined in the different other chapters of this document.

The next two sections (5.3 and 5.4) define the modification of the sequential flow by if-statement and group.

Section 5.5 defines the goto-statement which interrupts the sequential flow of control and starts it again with a completely changed state of the machine.

Section 5.6 defines the activation, and termination, of a block level by means of a called procedure declaration instead of a begin block occurring in the sequential text.

Metavariables

| st     | is-p-st |
| expr   | is-expr |
| ref    | is-ref  |
| op     | is-op   |
| gen    | is-gen  |
| decl   | is-p-decl |
| attr   | is-attr |
| da     | is-da   |
| eda    | is-eda  |
| descr  | is-descr |
| edescr | is-edescr |
5.1 Dynamics of Block Structure

The block structure of a PL/I program leads on interpretation to a dynamic system of nested "block activations". Whenever a new block (begin block, procedure body or on-unit) is going to be interpreted a new block activation is established in the state of the PL/I-machine; when the interpretation of this block ends this block activation is terminated and the former block activation reestablished. No established block activation is terminated before all block activations established later are terminated. The first block activation of a task is established by creating the task and is terminated by terminating the task.

At each time, the block activation established last, the "current" block activation, is represented in the state of the PL/I machine by the following five "local" state components: the environment $E$, the epilogue information $EI$, the condition state $CS$, the control information $CI$ and the control $C$. The other "active" block activations, i.e. the established but not yet terminated ones, are represented by the sixth local state component, the dump $D$.

This section defines the interpretation of a begin block, i.e. it defines the normal actions performed on establishing and terminating a block activation.

The special actions performed on interpretation of a procedure body are defined in 5.6, those performed on interpretation of an on-unit in 9.4.

The interpretation of the statement list of a block within the block activation is defined in the next section (5.2).
Metavariables

t is-p-block the block to be interpreted as produced by the prepass from the program text
dp is-p-decl-part the declaration part of t
bl is-p-body-list the body list of t
cp is-cond-part the condition part of t

Abbreviation

\[
\text{id}_p = \{ \text{id} \mid \text{is-p-decl-id(dp)} \}, \text{ the set of all locally declared identifiers}
\]

(1) \text{int-block(t)} =

\[
\begin{align*}
&\text{s-ei:block-ei(EI)} \\
&\text{s-d:stack(x)} \\
&\text{s-ci: } Q \\
&\text{s-c:epilogue;} \\
&\text{int-st-list(st-list}_t) \\
&\text{update-dn(dp}_t,bl_t); \\
&\text{update-cs(cp}_t); \\
&\text{update-at(dp}_t); \\
&\text{update-env(dp}_t)
\end{align*}
\]

where: \(\text{dp}_t = \text{s-decl-part}(t)\)
\(\text{bl}_t = \text{s-body-list}(t)\)
\(\text{cp}_t = \text{s-cond-part}(t)\)
\(\text{st-list}_t = \text{s-st-list}(t)\)

Ref.: \text{int-st-list 5-16}(23)

5.1.1 The dump

The dump \(\mathcal{D}\) serves to keep track of the different levels of block activations. When establishing a new block activation all information necessary to reestablish the former one afterwards, i.e. the six local state components (including the former dump) \(\mathcal{E}, \mathcal{P}, \mathcal{C}, \mathcal{D}, \mathcal{C}, \mathcal{Q}\), are reserved in the dump. When the block activation is terminated they are reinstalled from the dump into the state of the machine.

Normally, the information in the dump is left unchanged during a block activation. So it is guaranteed that flow of control after termination can
continue with the state as left before establishing a block activation. Only in some cases of abnormal block termination the control component of the dump is replaced (cf. the goto statement 5.5, the return statement 5.6.4, the abnormal task exit 4.2).

(2) \( \text{stack}(\xi) = \)

\[
\mu_0(<s-e : s-e(\xi)>,
    <s-ei : s-ei(\xi)>,
    <s-cs : s-cs(\xi)>,
    <s-d : s-d(\xi)>,
    <s-ci : s-ci(\xi)>,
    <s-c : s-c(\xi)>)
\]

(3) \( \text{unstack} = \)

\[
\text{unstack} = \begin{align*}
    s_e : s-e(D) \\
    s_ei : s-ei(D) \\
    s_cs : s-cs(D) \\
    s_d : s-d(D) \\
    s_ci : s-ci(D) \\
    s_c : s-c(D)
\end{align*}
\]

5.1.2 The environment

The environment \( \xi \) is the link which connects each (qualified) name occurring in the program text (which is an identifier list, e.g. the \( s\)-id-list component of a reference) via a unique name \( n \) with that information in the different directories of the state which determine its current meaning.

On establishing a block activation all "local" names, i.e. those declared by the declaration part \( dp \), receive new unique names and thereby new meaning, while those names valid in the former block activation which are not redeclared by \( dp \) retain their unique names and thereby their meaning.

This is guaranteed by inheriting the environment \( \xi \) from the former block activation and updating it by entering a new (or overwriting the old) unique name for each local name.

The environment is updated as the very first action within a block activation. Then it is left unchanged as long as this block activation is the current one.
Each single declaration occurring in the declaration part dp of the block gets its own characteristic unique name n. This unique name is entered into E for all single qualified names belonging to the declaration. These qualified names are for non-variable declarations the one-element-lists \(<id>\) of the main identifier only, while for variable declarations all, terminal or non-terminal, qualified names are extracted from the data attributes.

**Metavariables**

- id  \(\text{is-id}\)  the main identifier of a single declaration
- da  \(\text{is-named-da} \lor \text{is-Q}\)  the data attribute component for a variable declaration, Q for another declaration
- idl  \(\text{id-l-list}\)  a qualified name referencing part of the declaration, it starts with id and continues with an identifier list extracted from da
- n  \(\text{is-n}\)  the unique name for the declaration entered into E

(4) \[\text{update-env}(dp) = \]
\[\text{null}; \{\text{update-decl-env}(\text{id}, \text{s-da}\circ \text{id}(dp), n) ; n: \text{un-name} | \text{id}\in \text{id}(dp)\}\]

Ref.: un-name 2-17(36)

(5) \[\text{update-decl-env}(\text{id}, \text{da}, n) = \]
\[\text{s-e}: \mu(E; \{\text{sel}(\text{id}) : n | \text{id}\in \text{collect-idl}(\text{id}, \text{da})\})\]

(6) \[\text{collect-idl}(\text{idl}, \text{da}) = \]
\[\text{is-array}(\text{da}) \rightarrow \text{collect-idl}(\text{idl}, \text{s-elem}(\text{da}))\]
\[\text{is-struct}(\text{da}) \lor \text{is-cell}(\text{da}) \rightarrow \]
\[\{\text{idl}\} \cup \bigcup_{i=1}^{\text{order}(\text{da})} \text{collect-idl}(\text{idl} \cap \text{s-id-elem}(i, \text{da}), \text{s-da-elem}(i, \text{da}))\]
\[T \rightarrow \{\text{idl}\}\]

Note: Especially \(\text{collect-idl}(\text{id}, Q) = \{\text{id}\}\), so that for all declarations without a s-da component \(\text{update-decl-env}\) results in:
\[\text{s-e}: \mu(E; \{\text{sel}(\text{id}) : n\})\]

5.1.2
5.1.3 The attribute directory

The attribute directory $\mathcal{AT}$ contains for each declaration of an established block activation an object consisting of its attribute and the environment of its block activation.

The attribute is the completely unchanged declaration as occurring in the program text, without the $s$-den component of variable, entry and file declarations, which occurred in the original program text or was inserted by the prepass and is used for constructing the entry into the denotation directory (cf. 5.1.5).

The environment component in the attribute directory is necessary since in different situations names occurring in an attribute have to be interpreted with the meaning determined by this environment (cf. 5.1.2) at a later time when the environment $E$ of the state has been changed by a new block activation (e.g. allocation and initialization of controlled variables cf. 6.1.1, 6.2.1, reference of defined or based variables cf. 7.3.3, 7.3.4, generic selection cf. 5.6.5, goto statement cf. 5.5).

The entries in the attribute directory are accessible by means of the unique names $n$ of the environment $E$. They are never deleted; but by means of the environment handling (cf. 5.1.2) at each time only the entries belonging to declarations of active block activations are accessible.

The entries in the attribute directory are normally never changed; only the attributes of parameters are completed in some cases by attributes coming from the passed argument (cf. 5.6.2.4).

(7) $\text{update-at}(dp) =$

$$s\text{-at}: \mu(\mathcal{AT}; \{ \langle \text{sel}(\langle id \rangle)(E); \mu_0(\langle s\text{-attr:attr}_id \rangle, \langle s\text{-e:E} \rangle) \rightarrow \{ id \in \text{ids}_{dp} \} \})$$

where: $\text{attr}_id = (is\cap s\text{-den}\cdot id(dp) \rightarrow id(dp), T \rightarrow \delta(id(dp); s\text{-den}))$

5.1.4 The condition state

The condition state $\mathcal{CS}$ contains all information necessary for raising of condition actions (cf. 2.5). Since all of this information when entered into $\mathcal{CS}$ is valid only until termination of the current block activation, the condition state is reserved in the dump like the other local state components (cf. 5.1.1).
Two components of CS, the block prefix part s-bpp(CS) and the statement prefix part s-spp(CS), are to be updated at the begin of each block activation and each statement respectively. This is done by inheriting the former block prefix part and merging it with the information received from the condition part occurring in the text of the block or statement.

Metavariabes

- **cp** is-cond-part a (static) condition part as occurring in the program text
- **bpp** is-dyn-pref-part the (dynamic) block prefix part of the surrounding block

\[ \text{update-cs}(cp) = \]

\[
(\forall idl) \,(\neg is-\emptyset \cdot (sel(idl)) \cdot s-check(cp) \supset \neg is-PARAM \cdot s-scope(\text{attr}_{idl}) \land \\
(is-prop-var(\text{attr}_{idl}) \lor is-entry(\text{attr}_{idl}) \lor is-LABEL(\text{attr}_{idl}))) \rightarrow \\
\text{s-cs: } \mu(CS; <s-bpp:merge-pref(cp,s-bpp(CS),E,AT)>, \\
<s-spp:merge-pref(cp,s-bpp(CS),E,AT)>)
\]

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

where: \( attr_{idl} = s-\text{attr}(sel(idl))(E)(AT) \)

\[ \text{merge-pref}(cp,bpp,env,at) = \]

\[
\mu(bpp; \{ <\text{cond-dyn-sel}(\text{cond},\text{env},\emptyset,at);T> \mid is-ON(\text{cond-sel}(\text{cond})(cp)) \} \cup \\
\{ <\text{cond-dyn-sel}(\text{cond},\text{env},\emptyset,at);\emptyset> \mid is-NO(\text{cond-sel}(\text{cond})(cp)) \})
\]

Ref.: cond-dyn-sel 2-30(100), cond-sel 2-29(99)

5.1.5 The denotation directory

The denotation directory DN contains for most declarations of the established block activations their denotations. These entries in the denotation directory are accessible by means of the unique names \( n \) of the environment \( E \). They are never deleted, but by means of the environment handling (cf. 5.1.2) accessible only for the duration of their block activations.

The denotation of a declaration contains the information necessary for interpretation of the individual declared name, if not contained in the attribute directory (cf. 5.1.3). The denotation differs very much for the different types of declarations. It is the following:
1) For a proper variable a unique name b, the "aggregate name", serving as link via the aggregate directory \( AG \) (cf. 2.4.2) to the storage allocated for it (cf. 2.4.1). For a static or controlled variable this unique name \( b \) is inserted by the prepass as s-den component into the declaration in the program text (cf. 3.3), for an automatic variable it is newly created at each block activation.

2) For a file declaration a unique name \( f \) serving as link via the file directory \( FD \) (cf. 2.6.1) and the file union directory \( FU \) (cf. 2.6.2) to the file in the external storage. Also this unique name \( f \) is inserted by the prepass as s-den component into the declaration text.

3) For an entry declaration an object composed from: the identifier \( id \) of the declaration, the procedure body, the current environment and condition state, the evaluated parameter description list and return type. All this information is necessary for correct activation of the declared procedure (cf. 5.6). The body is received for an internal declaration by means of the integer value contained as the s-den component in the declaration from the body list \( bl \) of the block, for an external declaration by means of the unique name inserted as s-den component by the prepass from the denotation directory itself. The parameter descriptor list and return type are taken in any case from the declaration itself (so different declarations of the same external entry name may have different parameter descriptor lists and return types).

4) For a defined or based variable its evaluated data attributes (cf. 5.1.6).

5) For a builtin function an object consisting of its identifier only (however for builtin functions passed to entry parameters, cf. 5.6.2.1).

6) For a label an object consisting of its identifier and unique name.

7) For a format label an object consisting of the format list, the current environment and the statement prefix part merged from the condition part of the format declaration and the current block prefix part as described in 5.1.4.

8) For a generic or condition name no denotation is necessary, i.e. \( \emptyset \) is entered as denotation.

For automatic variables besides entering aggregate names into the denotation directory their allocation and initialization is performed.
Metavariables

<table>
<thead>
<tr>
<th>decl</th>
<th>is-p-decl</th>
<th>a declaration as modified by the prepass from the program text</th>
</tr>
</thead>
<tbody>
<tr>
<td>n</td>
<td>is-n</td>
<td>the unique name entered for the declaration into the environment</td>
</tr>
<tr>
<td>den</td>
<td>is-den</td>
<td>the denotation produced for the declaration</td>
</tr>
<tr>
<td>id</td>
<td>is-id</td>
<td>the main identifier of the declaration</td>
</tr>
<tr>
<td>dl</td>
<td>is-descr-list</td>
<td>the parameter descriptor list of an entry declaration</td>
</tr>
<tr>
<td>descr</td>
<td>is-descr</td>
<td>a single parameter descriptor</td>
</tr>
<tr>
<td>edl</td>
<td>is-edescr-list</td>
<td>the parameter descriptor list of an entry declaration with evaluated data attributes of variable descriptors</td>
</tr>
<tr>
<td>ert</td>
<td>is-e-ret-type</td>
<td>the return type of an entry declaration, with evaluated length in case of string</td>
</tr>
</tbody>
</table>

(10) update-dn(dp,bl) =

(∀id)(id,ids dp) → ¬is-PARAM*scope*id(dp) →
null;
{ update-ecl-dn(sel(id))(E),den; den:eval-den(id,id(dp),bl) | id ∈ ids dp }

T → error;

(11) update-decl-dn(n,den) =

s-dn: μ(DN;<n:den>)

(12) eval-den(id,decl,bl) =

is-STATIC*s-stg-cl(decl) v is-CTL*s-stg-cl(decl) v is-p-file(decl) →
PASS:s-den(decl)

is-AUTO*s-stg-cl(decl) & is-INT*s-scopc(decl) →
pass(b);
initialize(id,s-init-set(decl),s-da(decl),gen,E);
gen:auto-allocate(b,eda,s-dens(decl));
b : un-name,
eda:eval-da(s-da(decl),E)

cont'd
is-p-entry(decl) →

\[ \text{eval-entry-den}(id, decl, b1, edl, ert); \]
\[ \text{edl: eval-descr-list}(s-param-list(decl)), \]
\[ \text{ert: eval-da}(s-ret-type(decl), E) \]

is-defined(decl) →

\[ \text{eval-da}(s-da(decl), E) \]

is-based(decl) &

\[ (\forall \text{extent}(\text{extent} \in \text{extents-set} \ast s-da(decl) \supset \text{is-intg-const}(\text{extent}))) \rightarrow \]
\[ \text{eval-da}(s-da(decl), E) \]

is-BUILTIN(decl) →

\[ \text{PASS}: \mu_0(<s-id:id>) \]

is-LABEL(decl) →

\[ \text{pass-label-den}(id, E, AT) \]

is-format-attr(decl) & is-Q*s-check*s-cond-part(decl) →

\[ \text{PASS}: \mu_0(<s-format-list:s-format-list(decl)>, \]
\[ <s-e:E>, \]
\[ <s-spp:merge-prefix(s-cond-part(decl), s-bpp(CS), E, AT)>) \]

is-generic(decl) v is-COND(decl) → null

T → error

Ref.: eval-da 5-13(18), initialize 6-16(34), is-p-file 3-10(17),
is-p-entry 3-10(17), is-intg-const 5-41(69), un-name 2-17(36)

Note: The predicates is-p-file and is-p-entry characterize declarations as
modified by the prepass (cf. 3.3).
(13) `auto-allocate(b,eda,dens) =`

\[
\begin{align*}
\text{is-correct-eda}(eda) & \rightarrow \text{error} \\
\text{is-mstg-applic}(p_1,S) & \rightarrow \\
\text{PASS:mk-gen}(eda,dens,I,S) \\
s_s & : \text{el-alloc}(p_1,I,S) \\
s-ag & : \mu(AG; b: \text{mk-gen}(eda,dens,I,S),\text{NULL}) \\
s-ei & : \mu(EI; s-\text{free-set}; \text{free-set}(EI) \cup \{b\})
\end{align*}
\]

\[T \rightarrow \text{mstg-overflow}\]

where: \(p_1 = \text{alloc-space}(eda,dens,S)\)

Ref.: `is-mstg-applic` 6-5(5), `alloc-space` 6-11(19), `is-correct-eda` 6-5(7), `el-alloc` 2-23(75), `mk-gen` 6-5(8), `mstg-overflow` 6-5(6).

Note: This instruction differs from the instruction `allocate-1` (cf. 5-44(80)) only in the fact, that \(b\) is entered into the free set of \(EI\) (cf. 5.1.7) in order to ensure that the storage is freed at the end of the current block. (cf. also the instruction `allocate` 6-4(4)).

(14) `eval-descr-list(dl) =`

\[
\begin{align*}
is-\ast(dl) & \rightarrow \text{PASS:*} \\
is-\text{<>}(dl) & \rightarrow \text{error} \\
T & \rightarrow \\
\text{pass}(edl); \\
\{\text{elem}(i)(dl): \text{eval-descr}(\text{elem}(i,dl)) \mid 1 \leq i \leq \text{length}(dl)\}
\end{align*}
\]

(15) `eval-descr(descr) =`

\[
\begin{align*}
is-\text{var-descr}(descr) & \rightarrow \\
\text{pass}(edescr); \\
s-da(edescr): \text{eval-da}(s-da(descr),E); \\
edescr: \text{pass}(descr)
\end{align*}
\]

\[T \rightarrow \text{PASS:descr}\]

Ref.: `eval-da` 5-13(18).

Note: Since only evaluated variable descriptors are needed for argument testing and passing, the entry descriptors are not evaluated. Their "nested" parameter descriptors are needed only in not evaluated form for generic selection.
(16) \texttt{eval-entry-den}(id,decl,bl,edl,ert) =
\begin{align*}
is-\text{INT}\cdot \text{s-scope}(\text{decl}) & \land \text{lss-den}(\text{decl})\cdot \text{s-length}(\text{bl}) \\
\text{PASS: } & \mu_0(<s\cdot \text{id:id}>, \\
& <s\cdot \text{body:elem}(\text{den}(\text{decl}),\text{bl})>, \\
& <s\cdot \text{e:E}>, \\
& <s\cdot \text{bpp:s-bpp(CS)}>, \\
& <s\cdot \text{param-list:edl}>, \\
& <s\cdot \text{ret-type:ert}>)
\end{align*}
\begin{align*}
is-\text{EXT}\cdot \text{s-scope}(\text{decl}) \\
\text{PASS: } & \mu(<\text{den}(\text{decl})(\text{DN}); <\text{param-list:edl}>, \\
& <\text{ret-type:ert}>)
\end{align*}
\end{align*}
T \rightarrow \text{error}
\text{for: } \text{is-p-entry}(\text{decl})

(17) \texttt{label-den}(id,env,at) =
\begin{align*}
is-\text{LABEL}\cdot \text{s-attr}(\text{sel}(id)) (env)(at)) \\
\mu_0(<s\cdot \text{id:id}, <s\cdot n:sel(id))(env)> \\
is-\text{format-attr}\cdot \text{s-attr}(\text{sel}(id)) (env)(at)) \\
\end{align*}
\begin{align*}
T \rightarrow \text{error}
\end{align*}

\text{Note: Since this function is also used in the interpretation of the goto statement (cf. 5.5) the case distinction and error testing, which is not needed for eval-den, is included.}

5.1.6 Evaluation of data attributes

The data attributes occurring in the program text in declarations of variables, parameter descriptors and return types contain as "extents" (i.e., as array bounds, string lengths and area sizes) expressions or asterisks. For the allocation of storage however these expressions have to be evaluated (cf. 6.1), and also for argument passing data attributes with evaluated extents are needed (cf. 5.6.2). This evaluation of extent expressions is performed by the instruction eval-da which transforms an object satisfying is-da into an object satisfying is-edda, which has integer values instead of expressions as extents.
Besides, the qualifying substructure names and the label constant sets of label variables, which are irrelevant for storage allocation are deleted.

Since in some situations (e.g. storage allocation of controlled variables, cf. 6.1) this evaluation has to be performed after a new environment has been installed into the state by a new block activation, the environment of the declaration is passed as second argument to this instruction.

\[(18) \text{eval-da}(da, env) = \]
\[
is-\text{array}(da) \rightarrow \]
\[
\text{pass}(eda);
\quad s-lbd(eda): eval-\text{intg-expr}(s-lbd(da), env),
\quad s-\text{ubd}(eda): eval-\text{intg-expr}(s-\text{ubd}(da), env),
\quad s-\text{elem}(eda): eval-da(s-\text{elem}(da), env)
\]
\[
is-\text{struct}(da) \lor is-\text{cell}(da) \rightarrow \]
\[
\text{pass}(eda);
\quad s-\text{cell}(eda): \text{pass}(s-\text{cell}(da));
\quad \{ s-\text{da}*(elem(i))(eda): eval-da(s-\text{da}*elem(i, da), env) \mid \text{is-sorder}(da) \}
\]
\[
is-\text{string}(da) \rightarrow \]
\[
\text{pass}(eda);
\quad s-\text{length}(eda): eval-\text{intg-expr}(s-\text{length}(da), env);
\quad eda: \text{pass}(da)
\]
\[
is-\text{spec-area}(da) \rightarrow \]
\[
\text{pass}(eda);
\quad s-\text{size}(eda): eval-\text{intg-expr}(s-\text{size}(da), env)
\]
\[
is-\text{label}(da) \rightarrow \text{PASS: LABEL}
\]

\[T \rightarrow \text{PASS: da} \]

\[\text{for: is-da}(da) \lor is-\text{al}(da)\]

Ref.: \text{eval-intg-expr} 7-21(52)

Note: This instruction is used also for the partial data attributes occurring in the allocate statement (cf. 6.1). It is even necessary to transform integer constant extents into integer value extents, e.g. for static variables (cf. 3.3).
5.1.7 The block epilogue

A block activation is terminated by the block epilogue. It terminates all tasks activated during the activation if they are yet active (cf. 4.2), frees all allocated storage of local automatic variables (and dummy arguments, cf. 5.6.2.3) and reinstalls the former block activation (cf. 5.1.1).

For these actions the epilogue information $\text{EI}$ is necessary. It contains the set of aggregate names of all automatic variables (cf. 5-11(13)) and the set of unique task names of all attached tasks (cf. 4-6(10)); both sets are initially empty. Since the epilogue information contains further information needed by return from a procedure body (cf. 5.6.4), which is installed at procedure call (cf. 5-33(53)) and program initialization (cf. 3-2(2)) and may be needed also in nested blocks, these parts of $\text{EI}$ are inherited from the former block activation.

Metavariable

ei is $\text{ei}$ the epilogue information of the surrounding block

(19) block-ei(ei) = 

\[
\mu(ei; \langle s\text{-block-act:BLOCK} \rangle,
\langle s\text{-free-set :\{} \rangle,
\langle s\text{-task-set :\{} \rangle)
\]

(20) epilogue =

unstack;
free-block;
term-tasks-wait;
term-tasks

Ref.: unstack 5-4(3), term-tasks 4-10(19), term-tasks-wait 4-10(21)

(21) free-block =

null:
\{ auto-free(b) \mid b \in s\text{-free-set(EI)} \}
5.2 Sequential Interpretation of Statements

This section defines the flow of control within one block activation. The main part of a block activation is the execution of its statement list.

Section 5.2.1 defines the interpretation of a statement list and the sequential activation of its single statements, section 5.2.2 the common actions to be performed for each single statement, and section 5.2.3 the case distinction for the different statement types.

5.2.1 Statement list

The interpretation of a statement list is controlled by the control information $C_I$. The text component $s-tx(CI)$ contains the complete text of the statement list as occurring in the original program text and modified by the prepass (cf. 3.3). The statement counter $s-sc(CI)$ is an integer value counting which statement of the list is just being executed. The control contains only the instructions necessary for execution of this single statement and the instruction int-next-st which finally updates the statement counter and takes the next statement from the text component for execution into the control.

Now even within one block activation, there may be nested statement lists, a proper statement may itself be a statement list (cf. 5.2.3) or a group containing a statement list (cf. 5.4). Therefore the above mechanism with text and statement counter is not sufficient. Therefore the control information $C_I$ contains a third component the control dump $s-cd(CI)$, which acts for nested statement lists similar as the dump $D$ for nested block activations (cf. 5.1.1): Text, statement counter and control contain the information about the innermost nested statement list being executed while the control dump keeps track of the other nested statement lists being executed.
Whenever a new statement list is to be executed, the complete control information $CI$ and control $C$ are stacked in the control dump, and text, statement counter and control are initialized new. When the execution of a statement list is finished, $CI$ and $C$ are reinstalled from the control dump.

On establishing a new block activation the control information $C_4$ and $C$ are stacked in the dump $D$ like the other four state components (cf. 5.1.1(1) and 5.1.1(2)), and the control information $CI$ is initialized to $Q$. So $CI = Q$ is characteristic for the status when the prologue and epilogue actions of a block activation described in 5.1 are performed.

Metavariable

$t$ is-p-proc-st-list the text of the statement list to be executed as produced by the prepass.

(23) int-st-list$(t) =$

$$s-ci: \mu_{o}(\langle s-tx:t, \langle s-sc:0, \langle s-cd: \mu_{o}(\langle s-ci:CI, \langle s-c:C >) \rangle, s-c: \text{int-next-st}$$

(24) int-next-st =

$$s-sc(CI) \langle \text{length}\cdot s-tx(CI) \rangle \rightarrow$$

$$s-ci: \mu(CI, \langle s-sc:s-sc(CI) + 1 \rangle)$$

$$s-c: \text{int-next-st};$$

$$\text{int-st}(\text{elem}(s-sc(CI) + 1, s-tx(CI)))$$

$T \rightarrow$

$$s-ci:s-ci\cdot s-cd(CI)$$

$$s-c:s-c\cdot s-cd(CI)$$

Ref.: int-st 5-17(25)
5.2.2 Interpretation of a single statement

A single element of a statement list may be either a statement or an entry point of a procedure body (cf. 5.6.3). The latter is to be skipped on sequential execution of a statement list.

For a statement, first the condition status CS is updated by merging the condition part of the statement with the block prefix part of CS (cf. 5.1.4), entering the statement type and in case of an input/output statement the file title into CS. Then the statement labels are checked (cf. 9.4). Finally the proper statement itself is executed.

Metavariables

- t is-p-proc-st the text of the statement executed
- cp is-cond-part the condition part s-cond-part(t)
- st-type is-st-type v
- is- the type of the statement; an elementary object for "simple" statements, for block, statement list, group
- id is-id v is- the file identifier for input/output statements, otherwise
- id-1 is-id an identifier

(25) \[ \text{int-st}(t) = \]
\[ \text{is-p-st}(t) \rightarrow \]
\[ \text{int-proper-st}(s-prop-st(t)); \]
\[ \text{call-check-cond}(\text{lab-ref-list}_t, E, 301); \]
\[ \text{update-st-cs}(s-cond-part(t), s-st \cdot s-prop-st(t), s-file \cdot s-prop-st(t)) \]
\[ \text{is-entry-point}(t) \rightarrow \text{null} \]

where: \[ \text{lab-ref-list}_t = \left\{ \text{id-ref\_elem}(i, s-label-list(t)) \right\}_{i=1}^{\text{len_{lab}}} \]
\[ \text{len_{lab}} = \text{length}\cdot s-label-list(t) \]

Ref.: call-check-cond 9-15(27), int-proper-st 5-18(28), is-p-st 3-10(17)

(26) \[ \text{id-ref}(id-1) = \]
\[ \mu_0(<s-id-list : <id-1>>, \]
\[ <s-arg-list:>> \]
cont'd

5.2.2
Note: This function makes a simple reference from an identifier. It is needed whenever an instruction expects to receive a reference where an identifier is available.

(27) \[ \text{update-st-cs}(cp, st-type, id) = \]

\[ \text{is-}\Omega \cdot \text{s-check}(cp) \]

\[ \text{s-cs: } \mu (CS; <s\cdot\text{spp:merge-pref}(cp, s\cdot\text{bpp}(CS), E, AT)>, <s\cdot\text{s-cbis:st-type}>, <s\cdot\text{curr-file}s\cdot\text{cbif:id}>)
\]

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

where: \( id_1 = (\text{is-id}(id) \rightarrow s\cdot\text{title}\cdot s\cdot\text{attr}(\text{sel}(id))(E)(AT)), \)

\[ \text{is-}\Omega (id) \rightarrow \Omega \}

Ref.: merge-pref 5-7(9)

5.2.3 Case distinction of proper statements

The interpretation of a proper statement determines the type of the statement and installs in the control the special instruction for interpretation of this type of statement.

The definition of these different instructions is, depending on their type, given in the different chapters of this document (cf. chapters 4,5,6, 7,9,10). The references to the single instructions are given to the right of the formula.

Metavariable

\( t \quad \text{is-p-prop-st} \quad \) the text of the proper statement to be executed as produced by the prepass.

(28) \[ \text{int-proper-st}(t) = \]

\[ \text{is-p-block}(t) \rightarrow \text{int-block}(t) \]

(5-3(1))

\[ \text{is-p-group}(t) \rightarrow \text{int-group}(t) \]

(5-21(33))

\[ \text{is-p-proc-st-list}(t) \rightarrow \text{int-st-list}(t) \]

(5-16(23))

\[ \text{is-p-if-st}(t) \rightarrow \text{int-if-st}(t) \]

(5-20(29))

\[ \text{is-goto-st}(t) \rightarrow \text{int-goto-st}(t) \]

(5-26(45))

cont'd
<table>
<thead>
<tr>
<th>Predicate</th>
<th>Replacement</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>is-call-st(t)</code></td>
<td><code>int-call-st(t, E)</code></td>
<td>5-32</td>
</tr>
<tr>
<td><code>is-return-st(t)</code></td>
<td><code>int-return-st(t)</code></td>
<td>5-53</td>
</tr>
<tr>
<td><code>is-wait-st(t)</code></td>
<td><code>int-wait-st(t)</code></td>
<td>4-15</td>
</tr>
<tr>
<td><code>is-delay-st(t)</code></td>
<td><code>int-delay-st(t)</code></td>
<td>4-21</td>
</tr>
<tr>
<td><code>is-exit-st(t)</code></td>
<td><code>int-exit-st</code></td>
<td>4-9</td>
</tr>
<tr>
<td><code>is-stop-st(t)</code></td>
<td><code>int-stop-st</code></td>
<td>4-9</td>
</tr>
<tr>
<td><code>is-assign-st(t)</code></td>
<td><code>int-assign-st(t)</code></td>
<td>7-3</td>
</tr>
<tr>
<td><code>is-allocate-st(t)</code></td>
<td><code>int-allocate-st(t)</code></td>
<td>6-3</td>
</tr>
<tr>
<td><code>is-free-st(t)</code></td>
<td><code>int-free-st(t)</code></td>
<td>6-18</td>
</tr>
<tr>
<td><code>is-p-on-st(t)</code></td>
<td><code>int-on-st(t)</code></td>
<td>9-5</td>
</tr>
<tr>
<td><code>is-revert-st(t)</code></td>
<td><code>int-revert-st(t)</code></td>
<td>9-7</td>
</tr>
<tr>
<td><code>is-signal-st(t)</code></td>
<td><code>int-signal-st(t)</code></td>
<td>9-14</td>
</tr>
<tr>
<td><code>is-open-st(t)</code></td>
<td><code>int-open-st(t)</code></td>
<td>10-11</td>
</tr>
<tr>
<td><code>is-close-st(t)</code></td>
<td><code>int-close-st(t)</code></td>
<td>10-21</td>
</tr>
<tr>
<td><code>is-get-st(t)</code></td>
<td><code>int-get-st(t)</code></td>
<td>10-64</td>
</tr>
<tr>
<td><code>is-put-st(t)</code></td>
<td><code>int-put-st(t)</code></td>
<td>10-88</td>
</tr>
<tr>
<td><code>is-read-st(t)</code></td>
<td><code>int-read-st(t)</code></td>
<td>10-37</td>
</tr>
<tr>
<td><code>is-write-st(t)</code></td>
<td><code>int-write-st(t)</code></td>
<td>10-38</td>
</tr>
<tr>
<td><code>is-rewrite-st(t)</code></td>
<td><code>int-rewrite-st(t)</code></td>
<td>10-40</td>
</tr>
<tr>
<td><code>is-locate-st(t)</code></td>
<td><code>int-locate-st(t)</code></td>
<td>10-42</td>
</tr>
<tr>
<td><code>is-delete-st(t)</code></td>
<td><code>int-delete-st(t)</code></td>
<td>10-48</td>
</tr>
<tr>
<td><code>is-unlock-st(t)</code></td>
<td><code>int-unlock-st(t)</code></td>
<td>10-34</td>
</tr>
<tr>
<td><code>is-display-st(t)</code></td>
<td><code>int-display-st(t)</code></td>
<td>10-106</td>
</tr>
<tr>
<td><code>is-null-st(t)</code></td>
<td><code>null</code></td>
<td>1-21</td>
</tr>
</tbody>
</table>

Note: The predicate names starting with `is-p-`... define classes of objects as produced by the prepass (cf. 3-10(17)).
5.3 If-Statement

The if-statement selects, depending on the value of an expression, one out of two statements for execution. If the statement selected is missing, it is skipped.

Metavariables

\[
t \quad \text{is-p-if-st} \quad \text{the if-statement to be executed as produced by the prepass}
\]

\[
st-1 \quad \text{is-st} \quad \text{the then-alternative of t}
\]

\[
st-2 \quad \text{is-st v is-Q} \quad \text{the else-alternative of t, may be missing}
\]

(29) \[\text{int-if-st}(t) = \]

\[\text{int-if-st-1}(\text{truth}, s\text{-then-st}(t), s\text{-else-st}(t));\]

\[\text{truth}: \text{eval-truth}(s\text{-expr}(t))\]

(30) \[\text{int-if-st-1}(\text{truth}, st-1, st-2) = \]

\[
\begin{align*}
\text{truth} & \rightarrow \text{int-st}(st-1) \\
\text{is-p-st}(st-2) & \rightarrow \text{int-st}(st-2) \\
\text{is-Q(st-2)} & \rightarrow \text{null}
\end{align*}
\]

Ref.: int-st 5-17(25), is-p-st 3-10(17)

(31) \[\text{eval-truth(expr)} = \]

\[\text{pass-truth-val(op-1)};\]

\[\text{op-1}: \text{convert}(\text{BIT-DA}, op);\]

\[\text{op}: \text{eval-expr(expr, op)}\]

Ref.: eval-expr 7-18(43), convert 8-30(118), BIT-DA 8-13(59)

(32) \[\text{truth-val(op)} = \]

\[\oplus i)(\text{is-1-BIT}(\text{elem}(i)) \circ \text{op-val}(op))\]

for: \(\text{is-op}(op) \& \text{is-BIT}s\text{-base}s\text{-da}(op)\)

Ref.: op-val 8-11(40)
5.4 Group

In PL/I, there are two possibilities to specify iterated execution of a statement list: the while-group, described in section 5.4.1, and the controlled group, described in section 5.4.2.

The so-called "simple groups" are statement lists to be executed once (cf. 5.2.1); they are not denoted as "groups" in this document.

Metavariable

t is-p-group the text of the group to be executed as produced by the prepass.

(33) \begin{align*}
\text{int-group}(t) &= \\
\text{is-p-while-group}(t) &\rightarrow \text{int-while-group}(t) \\
\text{is-p-contr-group}(t) &\rightarrow \text{int-contr-do}(t, \emptyset )
\end{align*}

Ref.: is-p-while-group 3-10(17), is-p-contr-group 3-10(17)

5.4.1 While-group

A while-group specifies that its statement list is executed repeatedly as long as the evaluation of the while-expression yields "true".

(34) \begin{align*}
\text{int-while-group}(t) &= \\
\text{int-while-group}(t); \\
\text{int-st-list}(s\text{-st-list}(t)); \\
\text{while-test}(\text{truth}); \\
\text{eval-truth}(s\text{-while-exp}(t))
\end{align*}

for: is-p-while-group(t)

Ref.: eval-truth 5-20(31), \textit{int-st-list} 5-16(23)
(35) \[ \text{while-test(truth)} = \]

\[
\begin{align*}
\text{truth} & \rightarrow \text{null} \\
T & \rightarrow \text{int-next-st}
\end{align*}
\]

Ref.: \text{int-next-st} 5-16(24)

Note: The instruction \text{int-next-st}, if inserted into the control, causes all other instructions left in the control to be neglected, since by its definition (cf. 5-16(24)) it skips to the execution of the next statement in \text{s-tx(CI)}.

5.4.2 Controlled group

A controlled group is executed by interpreting a list of do-specifications. Each single do-specification controls the iterated execution of the statement list of the group.

The interpretation of a single do-specification is mainly performed by the instruction \text{iterate-do}: First, the controlling variable is compared with the value of the to-expression. Second, if the result of the compare was \( T \), the while-expression is evaluated. If both the compare and the while-expression evaluation yielded \( T \), 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 \( T \). If both the by-expression and the to-expression are missing, the iteration is never continued.

Since the same mechanism is used for the iteration of data item lists in get and put statements, the instruction \text{int-contr-do} is not only applied to controlled groups, but also to controlled data items. In the latter case, it has as second argument the input-output link which is needed for the interpretation of the iterated data item list (cf. 10.5.1.1).
Metavariables

\[ t \] is \( p \)-contr-group \( v \) the text of the controlled group or controlled data item list to be interpreted

\[ \text{is-contr-item} \]

\[ \text{link} \] is \( i-o \)-link \( v \) \( \Omega \) if \( t \) is a controlled group; the input/output-link necessary for the interpretation of the data item list if \( t \) is a controlled data item list

\[ \text{spec-list} \] is \( p \)-proc-st-list \( v \) the statement list or data item list to be iterated

\[ \text{list} \] is \( i-item \)-list

\[ \text{gen} \] is \( g-e n \) \( v \) \( \text{is-ps-gen} \) the generation (or pseudo generation) of the controlling variable

\[ \text{ref} \] is \( \text{ref} \) the reference of the controlling variable (used only for checking)

\[ \text{by-to} \]

\[ (<s-by:is-op \ v \ \Omega>, \text{an auxiliary object consisting of the optional operands resulting from evaluation of the optional by- and to-expressions}) \]

\[ (\text{by-to:eval-by-to}(s-by-expr(spec_1), s-to-expr(spec_1))) \]

\[ \text{expr} \] is \( \text{expr} \) the while-expression

(36) \( \text{int-contr-do}(t, link) = \)

\[ \text{iterate-spec-list}(s-spec-list(t), gen, s-contr-var(t), s-do-list(t), link); \]

\[ \text{gen:eval-lp}(s-contr-var(t)) \]

Ref.: \( \text{eval-lp} \ 7-4(5) \)

(37) \( \text{iterate-spec-list}(\text{spec-list}, gen, ref, list, link) = \)

\[ \text{is-\text{\textless}>(spec-list) \rightarrow \text{null} \}

\[ \neg (\text{is-arithm-s-da}(gen) \ v \ \text{is-string-s-da}(gen) \ v \ \text{is-pic-s-da}(gen)) \& \]

\[ \neg (\text{is-\Omega \ s-by-expr}(spec_1) \ & \ \text{is-\Omega \ s-to-expr}(spec_1)) \rightarrow \text{error} \]

\[ T \rightarrow \]

\[ \text{iterate-spec-list}(\text{tail(spec-list), gen, ref, list, link}); \]

\[ \text{iterate-do}(\text{gen, by-to, s-while-expr}(spec_1), \text{ref, list, link}); \]

\[ \text{call-check-cond}(<\text{ref}>, E, 302); \]

\[ \text{convert-assign}(\text{gen, op-1}); \]

\[ \text{by-to:eval-by-to}(s-by-expr(spec_1), s-to-expr(spec_1)); \]

\[ \text{op-1:eval-expr}(s-init-expr(spec_1), E) \]

cont'd
where: \texttt{spec}_1 = \text{head} (\text{spec-list})

Ref.: eval-expr 7-18(43), convert-assign 7-9(19),
call-check-cond 9-15(27)

(38) \texttt{eval-by-to(expr-1,expr-2)} =

\begin{align*}
\text{pass(by-to);} \\
\text{s-by(by-to)}: \text{eval-opt-expr(expr-1,E)}, \\
\text{s-to(by-to)}: \text{eval-opt-expr(expr-2,E)}
\end{align*}

Ref.: eval-opt-expr 7-21(53)

(39) \texttt{iterate-do(gen,by-to,expr,ref,list,link)} =

\begin{align*}
\text{do-continue(truth,gen,by-to,expr,ref,list,link);} \\
\text{int-do-list(truth,list,link);} \\
\text{truth: eval-while(truth-1,expr);} \\
\text{truth-1: eval-comp(gen,by-to)}
\end{align*}

(40) \texttt{eval-comp(gen,by-to)} =

\begin{align*}
\text{is-}_Q \neg \text{s-to(by-to)} \rightarrow \text{PASS:T} \\
\neg \text{is-}_Q \text{s-by(by-to)} \rightarrow \\
\text{pass-truth-val(op);} \\
\text{op: eval-infix-expr (gen-op (gen, S), s-to(by-to), opor)} \\
\text{opor: comp-operator (op-0);} \\
\text{op-0: eval-infix-expr (s-by(by-to), intg-op (O), GE)}
\end{align*}

\begin{align*}
T \rightarrow \text{error}
\end{align*}

Ref.: truth-val 5-20(32), eval-infix-expr 8-14(65), gen-op 7-19(47),
intg-op 8-11(42)

(41) \texttt{comp-operator(op-0)} =

\begin{align*}
\text{truth-val(op-0)} \rightarrow \text{PASS:LE} \\
T \rightarrow \text{PASS:GE}
\end{align*}

Ref.: truth-val 5-20(32)

(42) \texttt{eval-while(truth,expr)} =

\begin{align*}
\text{truth} \rightarrow \text{eval-truth(expr)} \\
T \rightarrow \text{PASS:F}
\end{align*}

Ref.: eval-truth 5-20(31)
(43) \texttt{int-do-list}(\texttt{truth}, \texttt{list}, \texttt{link}) =

\texttt{null}

\texttt{is-\penalty0 \theta (link)} \rightarrow \texttt{int-st-list}(\texttt{list})

\textbf{T} \rightarrow \texttt{int-item-list}(\texttt{list}, \texttt{link})

Ref.: \texttt{int-st-list} 5-16(22), \texttt{int-item-list} 10-52(132)

(44) \texttt{do-continue}(\texttt{truth}, \texttt{gen}, \texttt{by-to}, \texttt{expr}, \texttt{ref}, \texttt{list}, \texttt{link}) =

\texttt{null}

\texttt{T} \rightarrow

\texttt{iterate-do}(\texttt{gen}, \texttt{by-to}, \texttt{expr}, \texttt{ref}, \texttt{list}, \texttt{link});

\texttt{call-check-cond(<ref>,E,302)};

\texttt{convert-assign(\texttt{gen},\texttt{op-l})};

\texttt{op-l: eval-infix-expr(\texttt{gen-op(\texttt{gen},S)},s-by(by-to),ADD)}

Ref.: \texttt{eval-infix-expr} 8-14(65), \texttt{convert-assign} 7-9(19), \texttt{gen-op} 7-19(47),
\texttt{call-check-cond} 9-15(27)

5.5 \textbf{Goto Statement}

A goto statement interrupts the sequential flow of control and transfers it to a location denoted by a label constant, where it starts new. This process is done in two steps: first, the searching for the specified label, described in section 5.5.1, and second, the transfer of control to the state denoted by the label.

If the reference in the goto statement specifies a restricted label variable, the program is in error if the value of this variable is not contained in the label set of this variable.

\textbf{Metavarijables}

\begin{itemize}
  \item \texttt{t} is-goto-st the goto statement to be executed
  \item \texttt{ref} is-ref the label reference of the goto statement
  \item \texttt{op} is-op the operand resulting from the evaluation of ref
\end{itemize}
(45) \( \text{int-goto-st}(t) = \)

\[
\begin{align*}
\text{int-goto-st-1}(op, s-ref(t)); \\
op: \text{eval-ref}(s-ref(t), E)
\end{align*}
\]

Ref.: \text{eval-ref} 7-19 (46)

(46) \( \text{int-goto-st-1}(op, ref) = \)

\[
\begin{align*}
\text{is-label-den}(v_{op}) \land \\
(is\text{-rest}\text{-label}(dar_{gt}) \Rightarrow v_{op} \in \{\text{label-den}(id, env_{qt}, AT) | id\text{-set}(dar_{gt})\}) \Rightarrow \\
\text{goto-search}(s-id(v_{op}), s-n(v_{op}))
\end{align*}
\]

\( T \rightarrow \text{error} \)

where: \( v_{op} = \text{op-val}(op) \)
\( n_{gt} = \text{sel} \cdot s\text{-id-list}(ref)(E) \)
\( env_{gt} = s\cdot n_{gt}(AT) \)
\( attr_{gt} = s\cdot attr\cdot n_{gt}(AT) \)
\( dagt = s\cdot da(attr_{gt}) \)
\( sl_{gt} = s\cdot arg\cdot list(ref) \)
\( eql_{gt} = \text{eval-ql}(tail\cdot s\text{-id-list}(ref), dagt_{gt}) \)
\( dar_{gt} = (is\text{-LABEL}(attr_{gt}) \rightarrow \text{LABEL}, \\
is\text{-var}(attr_{gt}) \rightarrow \text{sub-da}(dagt_{gt}, sl_{gt}, eql_{gt})) \),

i.e. \( dagt_{gt} \) is the data attribute referenced by ref.

Ref.: \text{label-den} 5-12 (17), \text{is-label-den} 2-20 (49), \text{eval-ql} 7-27 (66), \text{sub-da} 7-34 (89), \text{op-val} 8-11 (40)

5.5.1 Label searching

A label denotation consists of the identifier \( id \) of a declared label constant and its unique name \( n \) (cf. 5-12 (17)). By such a denotation a location in the program text (namely a statement with \( id \) in its label list) and a special block activation of this text (namely a block activation in whose environment \( id \) leads to the unique name \( n \)) is denoted. The aim of the label searching process of a goto statement is to find the location in text and the block activation denoted by a label denotation.

A goto statement may lead only into established block activations and within them only into groups already under execution, including nested statement lists and if-statements contained in them, but not into contained blocks,
groups or on-statements. Each block activation or statement list left by a goto statement has to be terminated in the normal way as described in 5.1.7, 5.2.1 respectively.

The searching for the correct location in the PL/I machine state (i.e. text location and block activation) and termination of block activations and statement lists is performed level by level by inspecting the current text component of CI and reinstalling of the former statement list or block activation respectively until the correct location is found.

The inspection of the statement list in s-tx(CI) is performed by the function search, which systematically scans through all statements of this list and all statements in contained statement lists and if statements, but not the statements in contained blocks, groups and on-statements. This search function yields \( \emptyset \) as its value, if it has not found a statement with the label id. If it has found a label, it yields the list of those integer values, which one after the other would have been installed as statement counters in the control information CI (cf. 5.2.1) for the nested statement lists, if the flow of control had reached the labeled statement in the normal sequential way described in 5.2. To distinguish an if-statement and its two alternative statements, T is inserted instead of an integer value to denote the then-statement and F the else-statement. So the value of the function search is a list of integer and truth values uniquely denoting the location of the labeled statement within the inspected statement list.

Since nearly the same searching mechanism is used for searching the entry point in a procedure body denoted by an identifier id, the search function is constructed for both cases.

Metavariables

- \( \text{id} \) is-id the identifier of the label
- \( \text{n} \) is-n the unique name of the label
- \( \text{t} \) is-p-prop-st \( \forall \) the text to be scanned for the label id
- \( \text{is-p-prop-st} \)
- \( \text{le} \) is-LABEL \( \forall \text{is-ENTRY} \)

5.5.1
(47) \( \text{goto-search}(id,n) = \)
\[ \neg \text{is-}\varnothing \circ \text{search}(id,s-\text{tx}(CI),\text{LABEL}) \& \text{sel}(<id>)(E) = n \rightarrow \]
\[ \text{goto-jump}(\text{search}(id,s-\text{tx}(CI),\text{LABEL})) \]
\[ \neg \text{is-}\varnothing (CI) \rightarrow \]
\[ s-ci:s-ci \circ s-cd(CI) \]
\[ s-c : \text{goto-search}(id,n) \]
\[ \text{is-ON} \circ s-\text{block-act}(EI) \& \neg \text{is-}\varnothing \circ s-\text{ten} \circ s-\text{cbif}(CS) \rightarrow \]
\[ s-d: \mu(D;<s-c: \text{goto-search}(id,n)> ) \]
\[ s-c: \text{epilogue}; \]
\[ \text{compl-event-l}(s-\text{ten} \circ s-\text{cbif}(CS)) \]
\[ \neg \text{is-}\varnothing \circ s-c(D) \rightarrow \]
\[ s-d: \mu(D;<s-c: \text{goto-search}(id,n)> ) \]
\[ s-c: \text{epilogue} \]
\[ T \rightarrow \text{error} \]

Ref.: \text{epilogue} 5-14(20), \text{goto-jump} 5-29(50), \text{compl-event-l} 4-20(48)

Note: The third and fourth alternative of this definition have the effect that the current block activation is terminated as described in 5.1.7 and after this termination the reinstalled block activation finds in the control the instruction \text{goto-search}(id,n) instead of the control previously reserved in the dump D. The third alternative completes an I/O event after leaving an I/O on-unit.

(48) \( \text{search}(id,t,le) = \)
\[ \text{is-LABEL}(le) \& \text{is-p-st}(t) \& (\exists i)(i = \text{elem}(i,s\text{-label-list}(t))) \rightarrow < \]
\[ \text{is-ENTRY}(le) \& \text{is-entry-point}(t) \& id = s-id(t) \rightarrow < \]
\[ \text{is-p-st}(t) \rightarrow \text{search}(id,s\text{-prop-st}(t),le) \]
\[ \text{is-p-proc-st-list}(t) \rightarrow \text{search-1}(id,t,le,1) \]
\[ \text{is-p-if-st}(t) \& \neg \text{is-}\varnothing \circ \text{search}(id,s\text{-then-st}(t),le) \rightarrow \]
\[ <T> \neg \text{search}(id,s\text{-then-st}(t),le) \]
\[ \text{is-p-if-st}(t) \& \neg \text{is-}\varnothing \circ \text{search}(id,s\text{-else-st}(t),le) \rightarrow \]
\[ <F> \neg \text{search}(id,s\text{-else-st}(t),le) \]
\[ T \rightarrow \varnothing \]

Ref.: \text{is-p-st} 3-10(17), \text{is-p-proc-st} 3-10(17), \text{is-p-if-st} 3-10(17)

5.5.1
5.5.2 Transfer of control

After the location denoted by the label denotation has been found and all block activations and statement lists to be left have already been terminated by the searching process, the transfer of control has to simulate the situation which had been, if the labeled statement had been reached in the normal sequential flow of control.

This is done step by step, using the statement counter list produced by the search function in 5.5.1, by building up the control information CI, especially the control dump s-cd(CI). During this process intermediately an if-statement instead of a statement list may be the text component s-tx(CI) and Q instead of an integer value the statement counter s-sc(CI).

Metavariable

\( \text{scl} \) is-sc-list, where is-sc = is-intg-val \( \lor \) is-T \( \lor \) is-F, the statement counter list produced by the search function.

\[ \text{(50) } \text{goto-jump}(\text{scl}) = \]

\[ \text{is-intg-val} \cdot \text{head}(\text{scl}) \land \text{length}(\text{scl}) = 1 \rightarrow \]

\[ s-ci:ci_{scl} \]

\[ s-c : \text{int}- \text{next-st}; \]

\[ \text{int-st}(st_{scl}) \]

\[ \text{is-intg-val} \cdot \text{head}(\text{scl}) \rightarrow \]

\[ s-ci: \mu_0(\langle s-tx:**prop-st(st_{scl})** \rangle, \]

\[ s-cd: \mu_0(\langle s-ci:ci_{scl} \rangle, \]

\[ \langle s-c: \text{int}- \text{next-st} \rangle )) \]

\[ s-c : \text{goto-jump}(\text{tail}(\text{scl})) \]

cont'd
5.6 Procedure Call

A procedure is activated either by a call statement on sequential execution of a statement list (cf. 5.2.3), or on initialization of allocated storage (cf. 6.2), or by a function reference on expression evaluation (cf. 7.2.1).

In the case of a function reference, a dummy variable is allocated for the function value and freed afterwards (cf. 7.2.1) and the generation gen of this dummy variable passed through the procedure call to the return statement (cf. 5.6.4). In all other respects the procedure activation by a call statement and a function reference is performed completely in the same way.

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 as described in 5.1.

Section 5.6.1 describes all actions performed in the calling block activation to prepare the procedure call and after termination of the called procedure, except the actions concerned with argument passing. Section 5.6.2 describes the passing of arguments to the parameters of the called procedure. Section 5.6.3 describes the procedure call itself. Section 5.6.4 describes the termination of a procedure, especially the return statement.
It is possible in PL/I to declare a family of entry names by a generic declaration and to use in a call statement the family name instead of an individual entry name. In this case the selection of the individual entry name is performed before all actions performed usually. This generic selection is defined in section 5.6.5.

The attaching of a new task instead of a normal block activation by a call statement is described in chapter 4.

Metavariables

- **st** \( \Rightarrow \) **is-call-st**
  - the call statement to be interpreted

- **n** \( \Rightarrow \) **is-n**
  - the unique name of the entry to be called; in the case of generic this unique name is produced by the generic selection

- **expr-list** \( \Rightarrow \) **is-arg-expr-list**
  - the expression list occurring in the text as argument list of the call

- **t** \( \Rightarrow \) **is-arg-exp**
  - an element of expr-list

- **arg-list** \( \Rightarrow \) **is-arg-list**
  - the argument list produced from expr-list and passed to the called procedure

- **id-1** \( \Rightarrow \) **is-id**
  - the identifier occurring in the text as entry name; used only for checking the actual entry name (received from the denotation of the procedure);

- **id** \( \Rightarrow \) **is-id**
  - \( id = id-1 \) if it is not parameter and not generic

- **gen** \( \Rightarrow \) **is-gen \( v \) is-\( \emptyset \)**
  - the generation of the dummy for the function value in case of a function reference; \( \emptyset \) in case of a call-statement

- **pa-opt** \( \Rightarrow \) **is-pa-opt \( v \) is-\( \emptyset \)**
  - the parallel-action option in the text in case of a task call; \( \emptyset \) else

- **dl** \( \Rightarrow \) **is-descr-list \( v \) is-\( \ast \)**
  - the parameter descriptor list of the entry attribute

- **edl** \( \Rightarrow \) **is-edescri-list \( v \) is-\( \ast \)**
  - the evaluated parameter descriptor list dl
5.6.1 Call statement, actions before the call

The interpretation of a call statement consists of: Possibly generic selection (cf. 5.6.5), evaluation of the argument list to be passed to the parameter list of the called procedure (cf. 5.6.2), preparation of the condition state and epilogue information to be installed in the called procedure, checking of the entry name occurring in the program text (cf. 9.4.), the proper procedure call itself (cf. 5.6.3), and finally checking of those of the passed arguments which are no dummy variables (cf. 9.4).

The following local state components are passed to the proper call in order to be installed initially in the called procedure: The environment valid at the time of the entry declaration (stored in the entry denotation); the epilogue information, containing the set of dummy arguments to be freed at the end of the called procedure, the generation of the dummy variable for the function value if it is a function, and the information whether it is a normal procedure call or task call; the current condition state with the block prefix part of the block of the entry declaration (stored in the entry denotation).

The call may in some instances be executed in an environment env which is not that of the current state (e.g., the initialization of a controlled variable, cf. 6.2.1, function references in expressions which are evaluated not in the current environment, cf. 7.2.1). Therefore the environment env is passed explicitly to most of the involved instructions.

(51) \[ \text{int-call-st}(st, \text{env}) = \]

\[
\begin{align*}
\text{is-entry(\text{attr}_{st})} & \quad \longrightarrow \\
\text{int-call-l}(n_{st}, \text{s-arg-list}(st), \text{env}, \emptyset, \text{s-id}(st), \text{s-pa-opt}(st)) & \\
\text{is-generic(\text{attr}_{st})} & \quad \longrightarrow \\
\text{int-call-l}(n, \text{s-arg-list}(st), \text{env}, \emptyset, \text{s-id}(st), \text{s-pa-opt}(st)) & \\
\text{n:generic-sel}(\text{s-id}(st), \text{s-arg-list}(st), \text{env}) & \\
T & \quad \longrightarrow \quad \text{error}
\end{align*}
\]

where: \( n_{st} = \text{sel}(\langle\text{id}(st)\rangle)(\text{env}) \)

\( \text{attr}_{st} = \text{s-attr}_{st}(\text{AT}) \)

Ref.: generic-sel 5-54(99)
(52) \text{int-call-1}(n, \text{expr-list}, \text{env}, \text{gen}, \text{id-l}, \text{pa-opt}) =

\begin{align*}
\text{check-arg-list}(\text{expr-list}, \text{arg-list}, \text{env}, \text{pa-opt}) ; \\
\text{int-call}(\text{s-id}(\text{den}_n), \text{s-body}(\text{den}_n), \text{arg-list}, \text{s-e}(\text{den}_n), \text{ei}, \text{cs}, \text{pa-opt}) ; \\
\text{call-check-cond}(\langle \text{id-ref}(\text{id-l}) \rangle, \text{env}, 303) ; \\
\text{cs:proc-cs}(\text{s-bpp}(\text{den}_n), \text{s-id}(\text{den}_n)), \\
\text{ei:proc-ei}(\text{gen}, \text{arg-list}, \text{pa-opt}) ; \\
\text{arg-list:eval-arg-list}(\text{expr-list}, \text{s-param-list}(\text{attr}_n), \\
\text{s-param-list}(\text{den}_n), \text{env})
\end{align*}

where: \text{den}_n = n(\text{DN}) \text{ (the denotation of the called entry)}
\text{attr}_n = s-\text{attr}n(\text{AT})

Ref.: \text{eval-arg-list} 5-36(58), \text{int-call} 5-50(89), \text{call-check-cond} 9-15(27)
\text{id-ref} 5-17(26)

(53) \text{proc-ei}(\text{gen}, \text{arg-list}, \text{pa-opt}) =

\begin{align*}
\text{PASS: \mu}_o(\langle s-\text{block-act}: \text{block-act}_\text{pa} \rangle, \\
\langle s-\text{free-set}: \text{free-set}(\text{arg-list}) \rangle, \\
\langle s-\text{task-set}: \{} \rangle, \\
\langle s-\text{funct-gen}: \text{gen} \rangle)
\end{align*}

where: \text{block-act}_\text{pa} = (\text{is-\bigodot}(\text{pa-opt}) \longrightarrow \text{PROC}, \\
\text{T} \longrightarrow \text{TASK})

(54) \text{free-set}(\text{arg-list}) =

\{ b \mid \exists i \text{ (b = s-den*elem(i, arg-list) \& is-DUMMY*s-type*elem(i, arg-list))} \}

(55) \text{proc-cs}(\text{bpp}, \text{id}) =

\begin{align*}
\text{PASS: \mu}(\text{CS}; \langle s-\text{bpp}: \text{bpp} \rangle, \\
\langle s-\text{entry}: s-\text{chif}: \text{id} \rangle)
\end{align*}

for: \text{is-dyn-pref-part}(\text{bpp})

(56) \text{check-arg-list}(\text{expr-list}, \text{arg-list}, \text{env}, \text{pa-opt}) =

\begin{align*}
\text{is-\bigodot}(\text{pa-opt}) \longrightarrow \\
\text{call-check-cond}(\text{ref-list}, \text{env}, 304) ; \\
\text{ref-list:pass-arg-check-list}(\text{expr-list}, \text{arg-list}) \\
\text{T} \longrightarrow \text{null}
\end{align*}

Ref.: \text{call-check-cond} 9-15(27)

5.6.1
5.6.2 Argument passing

The expressions occurring as arguments in the text of the call statement or function reference are to be tested against the corresponding parameter descriptors in the entry attribute of the called procedure declaration. Depending on this test one of the following three ways of argument passing is applied:

1. direct passing
2. passing of a generation
3. passing of a dummy variable.

The relation to be satisfied between the attribute attr referred to by the argument expression and the parameter descriptor descr, and the evaluation of the passed argument are described for these three cases in sections 5.6.2.1, 5.6.2.2, 5.6.2.3 respectively.

In any of these three cases the instruction `eval-arg` constructs an argument arg, which satisfies the predicate `is-arg`:

```
is-arg = (is-den is-n v is-entry-den v is-builtin-den),
       (is-type is-ctl v is-gen v is-dummy v is-entry v
        is-builtin v is-file)
```

The s-den component contains a denotation passed to the parameter, the s-type component denotes the argument type.

In the called procedure, the instruction `install-arg` tests arguments of this form against the declarations of the corresponding parameters and enters, if the test is successful, the passed denotations into the denotation directory as denotations of the parameter. This is described in section 5.6.2.4.
Metavariables

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>t</td>
<td>is-arg-expr</td>
</tr>
<tr>
<td>arg</td>
<td>is-arg</td>
</tr>
<tr>
<td>descr</td>
<td>is-descr ∨ is-decl</td>
</tr>
<tr>
<td>edescr</td>
<td>is-edescr</td>
</tr>
<tr>
<td>attr</td>
<td>is-attr (∨)</td>
</tr>
<tr>
<td>decl</td>
<td>is-decl</td>
</tr>
<tr>
<td>da-1</td>
<td>is-da</td>
</tr>
<tr>
<td>da-2</td>
<td>is-da</td>
</tr>
<tr>
<td>eda-1</td>
<td>is-eda</td>
</tr>
<tr>
<td>eda-2</td>
<td>is-eda</td>
</tr>
<tr>
<td>n</td>
<td>is-n</td>
</tr>
<tr>
<td>idl</td>
<td>is-id-list</td>
</tr>
<tr>
<td>dp</td>
<td>is-decl-part</td>
</tr>
</tbody>
</table>

Abbreviations

If the expression to be passed as argument is a reference t the following abbreviations are used:

\[
\begin{align*}
\text{id}_t &= \text{head}\ast s\text{-id-list}(t) \\
\text{nt}_t &= \text{sel}\ast s\text{-id-list}(t)(\text{env}) \\
\text{den}_t &= \text{n}_t(DN) \\
\text{env}_t &= s\ast e\ast n_t(\text{AT}) \\
\text{attr}_t &= s\ast\text{attr}\ast n_t(\text{AT}) \\
\text{eql}_t &= \text{eval-ql}\text{(tail}\ast s\text{-id-list}(t),s\text{-da(attr}_t)) \\
\text{dar}_t &= \text{sub-da}(s\text{-da(attr}_t),s\text{-arg-list}(t),\text{eql}_t) \\
\text{ref-attr}_t &= \mu_0(<s\text{-da:dar}_t>, \\
&<s\text{-dens:s-dens(attr}_t)>)
\end{align*}
\]

Ref.: eval-ql 7-27(66), sub-da 7-34(89)
5.6.2.1 Directly passed arguments

An argument expression is passed directly, if it refers to a complete declaration, i.e. it is a simple reference, if argument attribute and parameter descriptor are controlled variable, entry type (entry, generic, builtin) or file, and if argument attribute and parameter descriptor match. In this case usually the denotation of the argument is passed unchanged to the parameter.

There are two exceptions from this unchanged passing of denotation:

If the argument refers to a generic, the generic selection is performed
according to the parameter descriptor list of the entry parameter descriptor (cf. 5.6.5) and the denotation of the selected entry declaration is passed.

If the argument refers to a float generic builtin function the generic selection is also made here and instead of the denotation of the complete builtin function family (which is of the form \((<s-id:is-id>)\), cf. 5.1.5) the denotation of the selected member of this builtin function family is passed. This denotation is of the form:

\[
\text{is-param-built-in-den} = (\langle \text{s-id:is-id}\rangle, \\
\langle \text{s-param-list:is-descr-list}\rangle)
\]

where the parameter descriptor list denotes the selected member. If a parameter with such a builtin argument is again passed as argument, its denotation is passed unchanged.

\[(60) \text{dir-match} (\text{attr}, \text{descr}) =
\]

\[
\text{is-} \ast (\text{descr}) \land
\text{(is-CTL} \ast \text{s-stg-cl} (\text{attr}) \lor \text{is-entry} (\text{attr}) \lor \text{is-param-built-in} (\text{attr}) \lor
\text{is-file} (\text{attr})) \lor
\]

\[
\text{is-var-descr} (\text{descr}) \land \text{is-CTL} \ast \text{s-stg-cl} (\text{descr}) \land
\text{is-CTL} \ast \text{s-stg-cl} (\text{attr}) \land \text{ctl-match} (\text{attr}, \text{descr}) \lor
\]

\[
\text{is-entry-descr} (\text{descr}) \land
\text{(is-entry} (\text{attr}) \lor \text{is-generic} (\text{attr}) \lor \text{is-built-in} (\text{attr})) \lor
\]

\[
\text{is-file-descr} (\text{descr}) \land
\text{is-file} (\text{attr}) \land \text{s-file-attr} (\text{descr}) \subseteq \text{s-file-attr} (\text{attr})
\]

Ref.: \text{is-built-in} 2-18(40), \text{is-param-built-in} 2-19(41)

Note: Builtin functions passed to entry parameters are for further argument passing to be handled like entry names.

\[(61) \text{ctl-match} (\text{attr}, \text{descr}) =
\]

\[
\text{ctl-da-match} (\text{s-da} (\text{attr}), \text{s-da} (\text{descr})) \land \text{dens-match} (\text{attr}, \text{s-dens} (\text{descr}))
\]

5.6.2.1
(62) \[\text{ctl-da-match}(da-1, da-2) =\]
\[\text{is-array}(da-1) \& \text{is-array}(da-2) \rightarrow \]
\[\text{ctl-da-match}(s\text{-elem}(da-1), s\text{-elem}(da-2))\]
\[(\text{is-struct}(da-1) \& \text{is-struct}(da-2) \lor \text{is-cell}(da-1) \& \text{is-cell}(da-2)) \& \]
\[\text{order}(da-1) = \text{order}(da-2) \rightarrow \]
\[\text{ctl-da-match}(s-da\text{-elem}(i, da-1), s-da\text{-elem}(i, da-2))\]
\[\text{is-string}(da-1) \& \text{is-string}(da-2) \& \]
\[\delta(da-1; s\text{-length}) = \delta(da-2; s\text{-length}) \rightarrow T\]
\[\text{is-area}(da-1) \& \text{is-area}(da-2) \rightarrow T\]
\[\text{is-label}(da-1) \& \text{is-label}(da-2) \rightarrow T\]
\[\text{is-scalar}(da-1) \& \text{is-scalar}(da-2) \& da-1 = da-2 \rightarrow T\]
\[T \rightarrow F\]

(63) \[\text{dens-match}(attr, dens) =\]
\[\text{s-dens}(attr) = dens \lor\]
\[\text{is-scalar} \cdot s\text{-da}(attr) \lor\]
\[\neg(\exists \chi)(\text{is-string} \cdot \chi \cdot s\text{-da}(attr) \lor \text{is-pico} \cdot \chi \cdot s\text{-da}(attr))\]

(64) \[\text{eval-dir-arg}(t, descr, env) =\]
\[\text{is-CTL} \cdot s\text{-stg-cl}(attr_t) \rightarrow\]
\[\text{PASS}: \mu_o(\langle s\text{-den:den}_t,\rangle,\]
\[\langle s\text{-type:CTL} \rangle)\]
\[\text{is-entry}(attr_t) \rightarrow\]
\[\text{PASS}: \mu_o(\langle s\text{-den:den}_t,\rangle,\]
\[\langle s\text{-type:ENTRY} \rangle)\]
\[\text{is-generic}(attr_t) \rightarrow\]
\[\text{PASS}: \mu_o(\langle s\text{-den:generic-sel-1}(attr_t, s\text{-param-list}(descr), env_t)(DN)\rangle,\]
\[\langle s\text{-type:ENTRY} \rangle)\]
\[\text{is-param-builtin}(attr_t) \rightarrow\]
\[\text{PASS}: \mu_o(\langle s\text{-den:den}_t,\rangle,\]
\[\langle s\text{-type:BUILTIN} \rangle)\]

cont'd
is-BUILTIN(attr_t) & is-float-generic-built-in(id_t) &
builtin-match(id_t, descr) →

\[
\begin{align*}
\text{PASS: } & \mu_{t}(<s\text{-den}: \mu_{o}(<s\text{-id}: id_{t}>, \text={<s\text{-param-list}: s\text{-param-list}(descr)>}), \text{<s\text{-type}: BUILTIN}>), \\
\text{is-file(attr_t)} → \quad \text{PASS: } & \mu_{o}(<s\text{-den}: \text{den}_{t}>, \text{<s\text{-type}: FILE}>)
\end{align*}
\]

T → error

for: is-simple-ref(t)
Ref.: generic-sel-1 5-54(100), is-float-generic-built-in 11-12(25),
is-param-built-in 2-19(41)

(65) builtin-match(id, descr) =

\[
\begin{align*}
\text{(length}_{p} & = 1 \lor (\text{length}_{p} = 2 \land \text{id} \in \{\text{mk-id( } \text{ATAN}) , \text{mk-id( } \text{ATAND}) \}) ) \land \\
\text{\bigwedge_{p}} & \text{is-FLT\text{*s-scale\text{*s-da*elem}(i, s\text{-param-list}(descr))}
\end{align*}
\]

where: \text{length}_{p} = \text{length\text{*s-param-list}(descr)}
for: is-float-generic-built-in(id),
is-entry-descr(descr)
Ref.: mk-id 10-76(214)
Note: The condition is-FLT\text{*s-scale(da) guarantees especially that
is-scalar(da) and is-arithm(da).

5.6.2.2 Passed argument generations

The generation of an argument is passed immediately, if the argument
expression in the text is a reference to a variable declaration (or a sub-
part of it) and if this declaration matches the parameter descriptor. Fur-
thermore it is necessary that string lengths and area sizes in both the text
of the argument's declaration and the parameter descriptor are integer con-
stants (if not stars in the parameter descriptor) so that the matching may
be recognized without evaluation of data attributes.

Nevertheless, after the decision is made, the argument's attributes
are tested against the evaluated parameter descriptor to confirm that also
array bounds are correct.

In this case, the referenced generation is entered into AG under a newly created unique aggregate name b, which is passed as denotation of the argument.

(66) gen-match(attr,descr) =
   is-\ast (descr) \lor
   is-var-descr(descr) \land \neg is-CTL\circ s-stg-cl(descr) \land
   gen-da-match\left(s-da(attr),s-da(descr),s-stg-cl(attr)\right) \land
   dens-match\left(attr,s-dens(descr)\right)

Ref.: dens-match 5-38(63), is-intg-op 8-11(44)

(67) gen-da-match\left(da-1,da-2,stg-cl\right) =
   is-array(da-1) \land is-array(da-2) \rightarrow
   gen-da-match\left(s-elem(da-1),s-elem(da-2),stg-cl\right)

(is-struct(da-1) \land is-struct(da-2) \lor is-cell(da-1) \land is-cell(da-2)) \land
order(da-1) = order(da-2) \rightarrow

\begin{align*}
\forall i \in \mathbb{N} & \quad gen-da-match\left(s-da\ast elem(i,da-1),s-da\ast elem(i,da-2),stg-cl\right) \\
\end{align*}

is-string(da-1) \land is-string(da-2) \land \delta(da-1; s-length) = \delta(da-2; s-length) \rightarrow
is-\ast \circ s-length(da-2) \lor
\neg is-CTL(stg-cl) \land equal-intg\left(s-length(da-1),s-length(da-2)\right)

is-area(da-1) \land is-area(da-2) \rightarrow
is-\ast \circ s-size(da-2) \lor
\neg is-CTL(stg-cl) \land (equal-intg\left(s-size(da-1),s-size(da-2)\right) \lor
is-AREA(da-1) \land is-AREA(da-2))

is-label(da-1) \land is-label(da-2) \rightarrow T

is-scalar(da-1) \land is-scalar(da-2) \land da-1 = da-2 \rightarrow T

T \rightarrow F

for: is-stg-cl(stg-cl) \lor is-\mathbb{Q}(stg-cl)

(68) equal-intg\left(expr-1,expr-2\right) =
   is-intg-const(expr-1) \land is-intg-const(expr-2) \land
   intg-const-val(expr-1) = intg-const-val(expr-2)

for: is-extent(expr-1), is-extent(expr-2)
(69) \( \text{is-intg-const}(\text{expr}) = \)

\( \text{is-intg-op}(\text{expr}) \lor \)

\( \text{is-prefix-expr}(\text{expr}) \land \text{is-intg-op's-oper}(\text{expr}) \land \)

\( (\text{is-PLUS's-operator}(\text{expr}) \lor \text{is-MINUS's-operator}(\text{expr})) \)

for: \( \text{is-extent}(\text{expr}) \)
Ref.: \( \text{is-intg-op} \ 8-11(44) \)

(70) \( \text{intg-const-val}(\text{expr}) = \)

\( \text{is-intg-op}(\text{expr}) \rightarrow \text{op-val}(\text{expr}) \)

\( \text{is-prefix-expr}(\text{expr}) \land \text{is-PLUS's-operator}(\text{expr}) \rightarrow \)

\( \text{op-val's-op}(\text{expr}) \)

\( \text{is-prefix-expr}(\text{expr}) \land \text{is-MINUS's-operator}(\text{expr}) \rightarrow \)

\( -\text{op-val's-op}(\text{expr}) \)

for: \( \text{is-intg-const}(\text{expr}) \)
Ref.: \( \text{op-val} \ 8-11(40), \ \text{is-intg-op} \ 8-11(44) \)

(71) \( \text{eval-gen-arg}(t, \text{edescri}, \text{env}) = \)

\( \text{eval-gen-arg-1}(b, \text{gen}, \text{edescri}); \)

\( \text{install-gen}(b, \text{gen}); \)

\( \text{gen: eval-ref-gen}(t, \text{env}), \)

\( b : \text{un-name} \)

for: \( \text{is-ref}(t), \ \text{is-var-edescri}(\text{edescri}) \)
Ref.: \( \text{eval-ref-gen} \ 7-25(59), \ \text{un-name} \ 2-17(36) \)

(72) \( \text{install-gen}(b, \text{gen}) = \)

\( \text{s-ag; } \mu(\text{AG}; <b: <\text{gen}>>) \)

Note: Since an entry into AG is made without allocation of storage, the
generation stack consists of only one generation and does not contain
the NULL generation. This fact guarantees that no freeing is performed
at task end (cf. 4.2).
5.6.2.3 Dummy arguments

If neither of the other two cases applies and the parameter descriptor is an asterisk or a variable descriptor, then a dummy variable is created and its aggregate name is passed as argument denotation.
First, the attributes of the dummy variable are determined from the parameter descriptor and, as far as the parameter descriptor leaves them open, from the argument expression. For this purpose the instruction \texttt{eda-expr} (cf. 7.2.3) is used which determines the data attributes of the result of an expression itself.

Second, the dummy variable is allocated.

Third, the argument expression is evaluated and assigned to the dummy variable. Hereby the same mechanism is used as in the assignment statement (cf. 7.1.1), which expands the argument expression and the generation of the dummy variable to scalar expressions and generations, which then are evaluated and assigned. This mechanism, by the way, does the job which in the other two cases the matching functions have to do, namely testing whether the argument fits to the parameter descriptor.

Finally the dummy variable's aggregate name is returned.

(76) \texttt{is-dummy-descr(descr) =}
    \texttt{is--*(descr) \lor is-var-descr(descr) \& \neg is-CTL:s-stg-cl(descr)}

(77) \texttt{eval-dummy-arg(t,edescr,env) =}
    \texttt{pass-dummy-arg(b);} \\
    \texttt{dummy-assign(gen,t,env);} \\
    \texttt{gen:allocate-1(b,eda,dens);} \\
    \texttt{b ::un-name,} \\
    \texttt{dens:pass-dummy-dens(eda,edescr);} \\
    \texttt{eda:pass-dummy-eda(eda-1,s-da(edescr));} \\
    \texttt{eda-1:eda-expr(t,env)}

for: \texttt{is-expr(t), is-var-edescr(edescr) \lor is--*(edescr)}

Ref.: \texttt{eda-expr 7-22(54), un-name 2-17(36)}

(78) \texttt{dummy-eda(eda-1,eda-2) =}
    \texttt{is-array(eda-1) \& is-array(eda-2) \& is--*s-lbd(eda-2) \& is--*s-ubd(edab-2) \rightarrow}
    \texttt{\mu_o(<s-lbd:s-lbd(eda-1)>,} \\
    \texttt{<s-ubd:s-ubd(eda-1)>,} \\
    \texttt{<s-elem:dummy-eda(s-elem(eda-1),s-elem(eda-2))>)}

cont'd
is-array(eda-1) & is-array(eda-2) →
\[ \mu_0(\langle \text{s-lbd:s-lbd(eda-2)}\rangle, \langle \text{s-ubd:s-ubd(eda-2)}\rangle, \langle \text{s-elem:dummy-eda(s-elem(eda-1),s-elem(eda-2))}\rangle) \]

is-struct(eda-1) & is-struct(eda-2) & order(eda-1) = order(eda-2) →
\[ \mu_0(\langle \text{s-da.elem(i):dummy-eda(s-da.elem(i,eda-1),s-da.elem(i,eda-2))} | \text{isorder(eda-1)} \rangle) \]

is-area(eda-1) & is-area(eda-2) & is-size(eda-2) → eda-1

is-area(eda-1) & is-area(eda-2) → eda-2

is-scalar(eda-1) & is-scalar(eda-2) → complete-tg(eda-2,eda-1)

is-Ω(eda-2) → eda-1

T → error

for: is-edd(eda-2) v is-Ω(eda-2)
Ref.: complete-tg 8-37(143)

(79) dummy-dens(eda,edescr) =
- is- (edescr) → s-dens(edescr)
is-array(eda) → ALIGNED
T → PACKED

for: edescr as in (77)

(80) allocate-l(b,eda,dens) =
- is-correct-eda(eda) → error
is- mstg-applic(p_1,δ) →
PASS:mk-gen(eda,dens,I,δ)
s-s:el-alloc(p_1,I,δ)
s-ag:μ(AC;<b:<mk-gen(eda,dens,I,δ),NULL>>)

T → mstg-overflow

where: p_1 = alloc-space(eda,dens,δ)
Ref.: mstg-overflow 6-5(6), is-mstg-applic 6-5(5), alloc-space 6-11(19), mk-gen 6-5(8), is-correct-eda 6-5(7), el-alloc 2-23(75)

Note: This instruction differs from the instruction allocate used for controlled allocation (cf. 6.1.1) in the fact that <NULL> instead of the already existing generation list is the tail of b(AC).

5.6.2.3
(81) \texttt{dummy-assign}(\text{gen}, t, \text{env}) =
\begin{align*}
& \text{is-scalar\_s-da}(\text{gen}) \rightarrow \\
& \text{convert-assign}(\text{gen, op}); \\
& \text{op: eval-expr}(t, \text{env}) \\
& T \rightarrow \text{iterate-dummy-assign}(\text{gen}, t, \text{lbd\_s-da}(\text{gen}), \text{env}) \\
& \text{for: is-expr}(t) \\
& \text{Ref.: eval-expr 7-18(43), convert-assign 7-9(19)}$
\end{align*}

(82) \texttt{iterate-dummy-assign}(\text{gen}, t, i, \text{env}) =
\begin{align*}
& i>\text{lbd\_s-da}(\text{gen}) \rightarrow \text{null} \\
& T \rightarrow \\
& \text{iterate-dummy-assign}(\text{gen}, t, i+1, \text{env}); \\
& \text{dummy-assign}(\text{sub-gen}(\text{gen, <i>}), \text{expr, env}); \\
& \text{expr: mod}(t, \text{s-da}(\text{gen}), i, \text{env}) \\
& \text{for: is-expr}(t), i>0 \\
& \text{Ref.: mod 7-7(13), sub-gen 7-32(84)}$
\end{align*}

(83) \texttt{dummy-arg}(b) =
\begin{align*}
\mu_0(<s\text{-den}: b>, \\
<s\text{-type}: \text{DUMMY}>)$
\end{align*}

5.6.2.4 Installation of arguments in the called procedure

Inside the called procedure the instruction \texttt{install-arg} tests the passed arguments against the corresponding parameter declarations and enters their denotations under the unique names \text{n} of the parameters into the denotation directory \text{DN}. In the case of controlled and file arguments the attributes of the argument needed for testing are taken from the attribute directory \text{AT}, where they have been entered under the denotation unique name by the prepass (cf. 3.3). In the case of generation and dummy arguments the evaluat-
ed attributes needed for testing are taken from the generation of the passed argument.

If the passed argument is controlled and the parameter is a non controlled variable, only the last generation is taken and handled like a passed argument generation (cf. 5.6.2.2).

In the case of a file argument, the attributes of the argument replace those of the parameter in AT.

In the case of a builtin function passed to an entry parameter, a flag is set into the parameter's attribute in AT to distinguish builtin from entry. The resulting entry in AT satisfies the predicate is-param-builtin (cf. 2-19(41)) while a non-parameter builtin function has in AT just the attribute BUILTIN.

(84) \text{install-arg-list}(\text{arg-list}, \text{idl}, \text{dp}) = \\
\text{length}(\text{arg-list}) = \text{length}(\text{idl}) & \\
\begin{array}{l}
\forall i \in \text{arg-list} \text{ is-PARAM-s-scope}(\text{decl}_i) & \text{is-GEN-s-den}(\text{decl}_i) \\
\text{null}; \\
\{\text{install-arg}(\text{arg}_i, \text{decl}_i, \text{id}_i) | \text{is-length}(\text{idl})\}
\end{array} \\
T \rightarrow \text{error}

where: \text{arg}_i = \text{elem}(i, \text{arg-list}) \\
\text{id}_i = \text{elem}(i, \text{idl}) \\
\text{decl}_i = \text{id}_i(\text{dp}) \\
\text{n}_i = \text{sel}(\text{id}_i)(\text{E})

(85) \text{install-arg}(\text{arg}, \text{decl}, n) = \\
\text{is-CTL-s-type}(\text{arg}) & \text{is-CTL-s-stg-cl}(\text{decl}) & \text{ctl-match(\text{attr}_\text{arg}, \text{decl})} \\
\text{s-den}(\text{DN}; \text{n}; \text{den}(\text{arg})); \\
\text{is-GEN-s-type}(\text{arg}) \lor \text{is-DUMMY-s-type}(\text{arg})) & \text{is-gen-param}(\text{decl}) \\
\text{install-gen-arg}(\text{s-den}(\text{arg}), \text{eda-2}, \text{s-dens}(\text{decl}), n), \\
\text{eda-2:eval-da}(\text{s-da}(\text{decl}), \text{E}) \\
\text{cont'd}
is-CTL s-type(arg) & is-gen-param(decl)

\[ \text{install-gen-arg(b,eda-2,s-dens(decl),n);} \]
\[ \text{eda-2:eval-da(s-da(decl),E),} \]
\[ \text{install-gen(b,head(s-den(arg)(AG))),} \]
\[ b:\text{un-name} \]

is-ENTRY s-type(arg) & is-entry(decl)

\[ s-dn: \mu(\text{DN}; n:s-den(\text{arg})) \]

is-BUILTIN s-type(arg) & is-entry(decl)

\[ s-dn: \mu(\text{DN}; n:s-den(\text{arg})) \]
\[ s-at: \mu(\text{AT}; n:s-builtin.s-attr:n:T) \]

is-FILE s-type(arg) & is-file(decl) & is-file-attr(decl) \subseteq s-file-attr(attr_arg)

\[ s-dn: \mu(\text{DN}; n:s-den(\text{arg})) \]
\[ s-at: \mu(\text{AT}; n:s-attr:n:attr_{arg}) \]

T \rightarrow \text{error}

where: \[ \text{attr}_{arg} = s-attr(s-den(\text{arg})(\text{AT})) \]

Ref.: eval-da 5-13(18), un-name 2-17(36), ctl-match 5-37(61), install-gen 5-41(72)

Note: The prepass entered the attributes of controlled variable and file declarations under their denotation unique names into the attribute directory AT (cf. 3-9(14), 3-10(15)). So this information is available by just passing the denotation of a parameter.

(86) is-gen-param(decl) =

\[ \text{is-prop-var(decl) & is-\{s-stg-cl(decl) &} \]
\[ \text{is-\{s-init-set(decl) & intg-star-extents \} s-da(decl)} \]

(87) intg-star-extents(da) =

\[ ( \forall \text{extent} ) ( \text{extent} \in \text{extents-set(da)} \rightarrow \text{is-intg-const(extent)} \lor \text{is-\{extent\} }) \]

Ref.: is-intg-const 5-41(69)
5.6.3 Call of procedure body

The proper procedure call, i.e. the establishing of a new block activation, is performed by the instruction \texttt{int-call}. It stacks the current local state components as described in 5.1.1 and installs the prepared local state components \texttt{env, ei, cs} (cf. 5.6.1).

The control of the called block activation is initialized by the instruction \texttt{int-proc-body(id, body, arg-list)} which interprets the procedure body of the entry declaration. This instruction is also used to interpret the procedure body of the initial program call (cf. 3.2) and of the task call (cf. 4.1).

The actual entry name \texttt{id} is necessary to specify the entry point \texttt{ep_id'}, since the same procedure body may have different entry points, which determine (1.) the start of interpretation of the statement list, (2.) the parameter list used, (3.) the return type in case of a function reference.

First, the environment and attribute directory are updated as described in 5.1.2 and 5.1.3. This means that entries into \texttt{E} and \texttt{AT} are made for all locally declared identifiers, even if they are not used parameters (parameters occurring in parameter lists of other entry points than the used one).

Second, the parameters are installed into the denotation directory as described in 5.6.2.4.

Third, after the usual updating of the condition state (cf. 5.1.4), the denotations of the local non-parameter declarations are entered into the denotation directory (cf. 5.1.5) and the denotations of the passed entry parameters are completed by evaluation of the parameter descriptor lists and return types of the local parameter declaration (Note: This has to be done after installation of all arguments and updating of the condition state, since expression evaluation is involved).
So, denotations of all non-parameter declarations and of all used parameters are to be found in DN. If, however, it is tried to refer illegally to a parameter occurring only in a not used entry point, then under its unique name n there is no denotation in DN, and thereby this illegal use fails.

Fourth, if the procedure is called by a function reference, the return type of the entry point $e_{id}$ is tested against the data attributes of the dummy variable allocated for the function value (cf. 7.2.1).

Fifth the statement list of the body is interpreted, starting at the specified entry point. This is done by using the mechanism defined for the goto statement (cf. 5.5).

Finally, if the procedure is not terminated earlier by an explicit return statement, the procedure is terminated by an implied return statement (cf. 5.6.4).

**Metavariabes**

<table>
<thead>
<tr>
<th>Metavarible</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>body</td>
<td>is-body</td>
</tr>
<tr>
<td>env</td>
<td>is-e</td>
</tr>
<tr>
<td>ei</td>
<td>is-ei</td>
</tr>
<tr>
<td>cs</td>
<td>is-cs</td>
</tr>
<tr>
<td>t</td>
<td>is-p-proc-st-list</td>
</tr>
<tr>
<td>dp</td>
<td>is-decl-part</td>
</tr>
<tr>
<td>bl</td>
<td>is-body-list</td>
</tr>
<tr>
<td>idl</td>
<td>is-id-list</td>
</tr>
<tr>
<td>rt</td>
<td>is-ret-type</td>
</tr>
<tr>
<td>ert</td>
<td>is-e-ret-type</td>
</tr>
</tbody>
</table>

5.6.3
\((89)\) \text{int-call}(id, body, arg-list, env, ei, cs, pa-opt) =

\[
\text{is-\(Q\)}(\text{pa-opt}) \\
\text{s-e : env} \\
\text{s-ei : ei} \\
\text{s-cs : cs} \\
\text{s-d : stack(\(\_\))} \\
\text{s-ci : \(Q\)} \\
\text{s-c : int-proc-body}(id, body, arg-list)
\]

\(T \rightarrow \)

\text{int-task-call}(id, body, arg-list, env, ei, cs, pa-opt)

Ref.: stack 5-4(2), \text{int-task-call} 4-3(1)

\((90)\) \text{int-proc-body}(id, body, arg-list) =

\[
\text{int-return-st}(\(Q\)); \\
\text{int-proc-st-list}(id, st-list_{bd}); \\
\text{test-ret-type}(\text{s-ret-type}(\text{ep}_id)); \\
\text{update-proc-dn}(\text{dp}_bd, \text{bl}_bd, \text{param-list}_id); \\
\text{update-cs}(\text{cp}_bd); \\
\text{install-arg-list}(\text{arg-list}, \text{param-list}_id, \text{dp}_bd); \\
\text{update-at}(\text{dp}_bd); \\
\text{update-env}(\text{dp}_bd)
\]

where:
\[
\text{dp}_bd = \text{s-decl-part}(body) \\
\text{bl}_bd = \text{s-body-list}(body) \\
\text{cp}_bd = \text{s-cond-part}(body) \\
\text{st-list}_{bd} = \text{s-st-list}(body) \\
\text{ep}_id = \text{ep}(id, body) \\
\text{param-list}_id = \text{s-param-list}\cdot\text{ep}(id, body)
\]

Ref.: \text{update-env} 5-5(4), \text{update-at} 5-6(7), \text{update-cs} 5-7(8), \text{int-return-st} 5-53(97), \text{install-arg-list} 5-46(84)

\((91)\) \text{ep}(id, body) =

\[
(\text{lt})(\text{is-entry-point}(t) \& \text{s-id}(t) = id \& \\
(\exists \chi)(t = \chi \cdot \text{s-st-list}(body) \& \\
\neg(\exists \chi - 1, \chi - 2)(\chi = \chi - 1 \cdot \chi - 2 \& \text{is-p-block}\cdot\chi - 2 \cdot \text{s-st-list}(body)))))
\]

Ref.: \text{is-p-block} 3-10(17)
(92) \[\text{update-proc-dn}(dp, bl, idl) =\]

\[
\text{null;}
\]

\[
\{\text{update-decl-dn}((\text{sel}(\langle id\rangle)(E), \text{den});
\quad \text{den}: \text{eval-den}(id, id(dp), bl) \quad | \quad id \in \text{ids}\_dp \quad \& \quad \text{is-PARAM}\_s\_scope\_id(dp)\} \cup
\quad \{\text{update-entry-dn}((\text{sel}(\langle id\rangle)(E), \text{edl-1}, \text{ert-1});
\quad \text{edl-1}: \text{eval-descr-list}(\text{s-\text{param}\_list}(\text{id}(dp))),
\quad \text{ert-1}: \text{eval-da}(\text{s-ret-type}\_id(dp), E) \mid
\quad \text{den} : \text{eval-den}(id, id(dp), bl) \quad \& \quad \text{is-\text{PARAM}\_s\_scope}(dp)\} \}
\]

\[\text{where: } \text{ids}\_dp = \{id \mid \text{is-p-decl}\_id(dp)\}\]

\[\text{attr}_{id} = \text{s-attr}(\text{sel}(\langle id\rangle)(E)(\text{AT}))\]

Ref.: eval-da 5-13(18), eval-den 5-9(12), update-decl-dn 5-9(11)

eval-descr-list 5-11(14), is-p-decl 3-10(17)

(93) \[\text{update-entry-dn}(n, \text{edl-1}, \text{ert-1}) =\]

\[
\text{s-dn: } \mu(DN; \langle \text{s-\text{param}\_\text{list}\_n: \text{edl-1}}\rangle,
\langle \text{s-ret-type}\_\text{n: \text{ert-1}}\rangle)
\]

\[\text{for: is-\text{edescr}\_\text{list}()} \cup \text{is-\text{*}()} \cup \text{is-\text{ret-type}()}\]

(94) \[\text{test-ret-type}(\text{rt}) =\]

\[
is-\emptyset \ast \text{s-funct-gen}(\text{EI}) \longrightarrow \text{null}
\]

\[
\text{intg-star-extents}(\text{rt}) \longrightarrow
\]

\[\text{test-ret-type-1}(\text{ert}) ;
\quad \text{ert}: \text{eval-da}(\text{rt, } E)
\]

\[
\text{T} \longrightarrow \text{error}
\]

Ref.: eval-da 5-13(18), intg-star-extents 5-47(87)

(95) \[\text{test-ret-type-1}(\text{ert}) =\]

\[
\text{gen-da-match-1}(\text{s-da}\ast \text{s-funct-gen}(\text{EI}), \text{ert}) \longrightarrow \text{null}
\]

\[\text{T} \longrightarrow \text{error}
\]

Ref.: gen-da-match-1 5-42(75)

5.6.3
(96) \textbf{int-proc-st-list} (id, t) =

\begin{align*}
\neg \text{is}_\Omega \cdot \text{search} (id, t, \text{ENTRY}) & \rightarrow \\
\text{s-ci} : & \mu (\langle \text{s-ct} : t, \langle \text{s-cd} : c \rangle \rangle) \\
\text{s-c} : & \text{goto-jump} (\text{search} (id, t, \text{ENTRY})) \\
T & \rightarrow \text{error}
\end{align*}

for: \text{is-p-proc-st-list} (t)
Ref.: \text{goto-jump} 5-29(50), \text{search} 5-28(48)

\textbf{5.6.4 Return statement}

A return statement, and implicitly the end of a procedure statement list (cf. 5.6.3), terminates the innermost currently active procedure activation and all nested block activations established later (cf. 5.1.7). Termination of an active on-unit by a return statement is illegal.

If the procedure to be terminated is a function (i.e. \text{s-funct-gen} (EI) is a generation gen, cf. 5-14(19) and 5-33(53)) and the return statement contains an expression, this expression is evaluated and assigned to the allocated dummy as the function value.

If the procedure to be terminated is the main procedure of the program (cf. 3.2) the finish condition is raised before the termination of any block. This is recognized by the component \text{s-main} (EI) of the epilogue information, which is \text{false} in the initial state, inherited by all begin block activations (cf. 5.1.7) and reset by all procedure calls (cf. 5.6.1).

If the procedure to be terminated is a task, the task is detached by the instruction \text{task-epilogue} (cf. 4.2).

Metavariable

\begin{align*}
t & \quad \text{is-return-st} \lor \text{is-} \Omega \quad \text{the return statement to be executed}; \Omega \text{ if it is an implicit return by the end of the procedure statement list.}
\end{align*}
(97) \( \text{int-return-st}(t) = \)

\[ \text{is-} \Omega \text{-s-funct-gen}(EI) \& \text{is-} \Omega \text{-s-expr}(t) \& \text{is-} \Omega \text{-s-main}(EI) \rightarrow \]

\[ \text{return} \]

\[ \text{is-} \Omega \text{-s-funct-gen}(EI) \& \text{is-} \Omega \text{-s-expr}(t) \rightarrow \]

\[ \text{return}; \]

\[ \text{call-cond}(\text{FINISH,305}) \]

\[ \neg \text{is-} \Omega \text{-s-funct-gen}(EI) \& \neg \text{is-} \Omega \text{-s-expr}(t) \& \text{is-} \Omega \text{-s-main}(EI) \rightarrow \]

\[ \text{return}; \]

\[ \text{convert-assign}(s\text{-funt-gen}(EI),op); \]

\[ \text{op:eval-expr}(s\text{-expr}(t),E) \]

\[ \neg \text{is-} \Omega \text{-s-funct-gen}(EI) \& \neg \text{is-} \Omega \text{-s-expr}(t) \rightarrow \]

\[ \text{return}; \]

\[ \text{call-cond}(\text{FINISH,305}) \]

\[ \text{convert-assign}(s\text{-funt-gen}(EI),op); \]

\[ \text{op:eval-expr}(s\text{-expr}(t),E) \]

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

Ref.: \text{call-cond} 9-11(18), \text{eval-expr} 7-18(43), \text{convert-assign} 7-9(19)

(98) \text{return} =

\[ \text{is-BLOCK\text{-}s\text{-}block\text{-}act}(EI) \rightarrow \]

\[ \text{s-d: } \mu(D;\text{<s-c:return>)} \]

\[ \text{s-c: epilogue} \]

\[ \text{is-PROC\text{-}s\text{-}block\text{-}act}(EI) \rightarrow \]

\[ \text{epilogue} \]

\[ \text{is-TASK\text{-}s\text{-}block\text{-}act}(EI) \rightarrow \]

\[ \text{task-epilogue}(\text{NORMAL}) \]

\[ \text{is-ON\text{-}s\text{-}block\text{-}act}(EI) \rightarrow \text{error} \]

Ref.: \text{epilogue} 5-14(20), \text{task-epilogue} 4-9(18)
5.6.5 Generic selection

The generic-selection is performed by the instruction **generic-sel** which returns a unique name denoting the selected family member. This is done in two steps. At first the attributes for each argument-place are evaluated and then matched against the parameter description of the generic family members.

The instruction above is also used in the instructions **eval-generic** and in **eda-ref** defined in chapter 7.2.1 and 7.2.3.

The evaluation of the attributes of an expression is done in chapter 7.2.3.

Metavariables

<table>
<thead>
<tr>
<th>Metavariable</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>list</td>
<td>list of argument attributes attr</td>
</tr>
<tr>
<td>attr</td>
<td>attribute of argument</td>
</tr>
<tr>
<td>is-attr v</td>
<td>attribute of argument</td>
</tr>
<tr>
<td>(&lt;s-da:is-da v is-eda&gt;, <a href="">s-dens:is-dens</a>)</td>
<td>data attributes of argument</td>
</tr>
<tr>
<td>da-1</td>
<td>is-da v is-eda</td>
</tr>
<tr>
<td>da-2</td>
<td>is-da</td>
</tr>
</tbody>
</table>

(99) generic-sel(id-1, expr-list, env) =

\[
\text{pass-generic-sel-1} \left( \text{attr}_1, \text{list}, \text{env}_1 \right) = \\
\{ \text{elem}(i)(\text{list}):\text{attr-arg}(\text{elem}(i, \text{expr-list}), \text{env}) \mid \text{list}=\text{length}(\text{expr-list}) \}
\]

where: \( \text{attr}_1 = \text{s-attr}(\text{sel}(\text{id}-1))(\text{env})(\text{AT}) \)

\( \text{env}_1 = \text{s-e}(\text{sel}(\text{id}-1))(\text{env})(\text{AT}) \)

(100) generic-sel-1(attr, list, env) =

\( \text{sel}(\text{s-id}\text{generic-sel-2}(\text{attr}, \text{list}))(\text{env}) \)

(101) generic-sel-2(attr, list) =

\[
(\forall x)(x \in s-\text{generic-ref-set}(\text{attr}) \& \\
\text{length}(\text{list}) = \text{length}(s-\text{param-list}(x)) \& \\
\text{generic-match}(\text{elem}(i, \text{list}), \text{elem}(i, s-\text{param-list}(x)))
\]

5.6.5
(102) \( \text{attr-arg}(t,\text{env}) = \)
\[
\text{is-simple-ref}(t) \land (\text{is-entry}(\text{attr}_t) \lor \text{is-file}(\text{attr}_t)) \lor \\
\text{is-param-built}(\text{attr}_t) \land \text{is-generic}(\text{attr}_t) \lor \text{is-COND}(\text{attr}_t) \implies \\
\text{PASS:attr}_t
\]
\( \text{is-expr}(t) \implies \\
\text{attr-expr}(\text{dens},\text{eda}); \\
\text{dens:}\text{dens-expr}(t,\text{env},\text{eda}); \\
\text{eda:}\text{eda-expr}(t,\text{env})
\]
\( T \implies \text{error} \)

where: \( \text{attr}_t = \text{s-attr}(\text{sel}\cdot\text{s-id-list}(t)(\text{env})(\text{AT})) \)
Ref.: \( \text{eda-expr} 7-22(54), \text{is-param-built} 2-19(41) \)

(103) \( \text{dens-expr}(t,\text{env},\text{eda}) = \)
\[
\text{is-ref}(t) \land \text{is-var}(\text{attr}_t) \implies \text{PASS:}\text{s-dens}(\text{attr}_t)
\]
\( \text{is-array}(\text{eda}) \implies \text{PASS:ALIGNED} \\
T \implies \text{PASS:PACKED} 
\]

where: \( \text{attr}_t = \text{s-attr}(\text{sel}\cdot\text{s-id-list}(t)(\text{env})(\text{AT})) \)
for: \( \text{is-expr}(t) \)

(104) \( \text{attr-expr}(\text{dens},\text{eda}) = \)
\[
\text{PASS:}\mu_\text{O}(<\text{s-da:eda},\text{s-dens:dens}>) 
\]

(105) \( \text{generic-match}(\text{attr},\text{descr}) = \)
\[
\text{(is-entry}(\text{attr}) \lor \text{is-param-built}(\text{attr})) \land \text{is-entry-descr}(\text{descr}) \lor \\
\text{is-file}(\text{attr}) \land \text{is-file-descr}(\text{descr}) \lor \\
\text{generic-da-match}(\text{s-da}(\text{attr}),\text{s-da}(\text{descr})) \land \text{s-dens}(\text{attr}) = \text{s-dens}(\text{descr}) \\
\text{ctl-da-match}(\text{s-da}(\text{attr}),\text{s-da}(\text{descr})) \land \text{s-dens}(\text{attr}) = \text{s-dens}(\text{descr}) 
\]
Ref.: \( \text{ctl-da-match} 5-38(62), \text{is-param-built} 2-19(41) \)
6. ALLOCATION, INITIALIZATION, FREEING OF VARIABLES

This chapter defines the allocate statement, the initialization of variables, and the free statement. The elementary storage functions on which the definitions are built are described in chapter 2.4.1.

The following elementary storage functions are used without reference in this chapter:

el-alloc 2-23(75), alloc-space 6-11(19), area-alloc-space 6-11(20), alloc-state 2-23(73), el-rf 2-22(68), el-ass 2-22(70), el-free 2-23(76), is-appli 2-21(57), is-indep 2-22(63), size 2-21(59), is-usuable 2-24(80)

Metavariables

<table>
<thead>
<tr>
<th>Metavariable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>al</td>
<td>data attributes specified in the allocate-statement</td>
</tr>
<tr>
<td>b</td>
<td>unique aggregate name</td>
</tr>
<tr>
<td>da</td>
<td>non-evaluated data attributes</td>
</tr>
<tr>
<td>dens</td>
<td>evaluated data attributes specified in the allocate-statement</td>
</tr>
<tr>
<td>eal</td>
<td>evaluated data attributes</td>
</tr>
<tr>
<td>eda</td>
<td>data attributes specified in most recent generation of a variable</td>
</tr>
<tr>
<td>edag</td>
<td>environment</td>
</tr>
<tr>
<td>env</td>
<td>generation of a variable</td>
</tr>
<tr>
<td>gen</td>
<td>list of generations</td>
</tr>
<tr>
<td>genl</td>
<td>integer value</td>
</tr>
<tr>
<td>i</td>
<td>identifier</td>
</tr>
<tr>
<td>id</td>
<td>initial specification</td>
</tr>
<tr>
<td>init-spec</td>
<td>initial set specified in the declaration</td>
</tr>
<tr>
<td>isd</td>
<td>initial set specified in the allocate-statement</td>
</tr>
<tr>
<td>iss</td>
<td>list of evaluated subscripts and/or name qualifiers</td>
</tr>
<tr>
<td>list</td>
<td>offset value</td>
</tr>
<tr>
<td>o</td>
<td>operand</td>
</tr>
<tr>
<td>op</td>
<td>pointer value</td>
</tr>
<tr>
<td>p</td>
<td>reference to variable</td>
</tr>
<tr>
<td>ref</td>
<td>is-intg-val-list</td>
</tr>
<tr>
<td>rl</td>
<td>cont'd</td>
</tr>
</tbody>
</table>
Abbreviations

for is-allocate-list(t) or is-free-list(t):

\[
\begin{align*}
  t_1 &= \text{head}(t) & \text{head of the list of allocates or frees} \\
  n_1 &= \text{sel}(\langle \text{s-id}(t_1) \rangle)(E) & \text{unique name of allocated or freed variable} \\
  \text{attr}_1 &= \text{s-at} \text{tr}(n_1(\text{AT})) & \text{attributes of allocated or freed variable} \\
  \text{env}_1 &= \text{s-env}(n_1(\text{AT})) & \text{environment in which variable was declared} \\
  \text{da}_1 &= \text{s-da}(\text{attr}_1) & \text{data attributes of allocated or freed variable} \\
  \text{gen}_1 &= \text{head}(n_1(\text{DN}(\text{AG}))) & \text{most recent generation of allocated or freed variable}
\end{align*}
\]

for is-ref(t) and is-e(env):

\[
\begin{align*}
  n_t &= \text{sel} \cdot \text{s-id-list}(t)(\text{env}) & \text{unique name of the referenced variable} \\
  \text{sl}_t &= \text{s-arg-list}(t) & \text{subscript list of the reference}
\end{align*}
\]

6.1 The Allocate Statement

The allocate statement is interpreted by the instruction
\text{int-allocate-st}(t), where t is the text of the statement. The allocations
specified in the statement are executed from left to right. The distinction
is made between the allocation of controlled variables, the allocation of
based variables in main storage, and the allocation of based variables in
areas.

The initialization of variables, executed immediately after allocation,
is defined in chapter 6.2

If the check condition is enabled for the set reference in a based
allocation, the check condition is called after the allocation.
(1) \[
\text{int-allocate-st}(t) = \text{int-allocate-list}(s\text{-allocate-list}(t))
\]
for: \text{is-allocate-st}(t)

(2) \[
\text{int-allocate-list}(t) = \text{null}
\]

is-CTL: \text{s-stg-class}(\text{attr}_1) \& is-\text{Q\text{-}set-ref}(t_1) \& is-\text{Q\text{-}set}(t_1) \rightarrow
\]
\[
\text{int-allocate-list}(\text{tail}(t))
\]
\[
\text{ctl-initialize}(\text{s-\text{id}}(t_1), \text{s-\text{init-set}}(t_1), \text{s-\text{init-set}}(\text{attr}_1), \text{gen}, \text{env}_1)
\]
\[
\text{gen:allocate}(n_1(\text{DN}), \text{eda}, \text{s-\text{dens}}(\text{attr}_1))
\]
\[
\text{eda:pass-mix-spec}(\text{eda-1}, \text{eal}, \text{s-da}(\text{gen}_1))
\]
\[
\text{eda-1:eval-da}(\text{da-al}(\text{da}_1, \text{s-al}(t_1)), \text{env}_1)
\]
\[
\text{eal:eval-da}(\text{s-al}(t_1), E)
\]
\[
is\text{-based}(\text{attr}_1) \& \text{is-\text{Q\text{-}set}(t_1) \& is-\text{Q\text{-}set}(t_1) \rightarrow
\]
\[
\text{int-allocate-list}(\text{tail}(t))
\]
\[
\text{initialize}(\text{s-\text{id}}(t_1), \text{s-\text{init-set}}(\text{attr}_1), \text{da}_1, \text{gen}, \text{env}_1)
\]
\[
\text{call-set-check}(t_1)
\]
\[
\text{gen:based-allocate}(n_1(\text{DN}), \text{s-\text{dens}}(\text{attr}_1), \text{gen-2})
\]
\[
\text{gen-2:eval-set-gen}(t_1)
\]
\[
is\text{-based}(\text{attr}_1) \& \text{is-\text{Q\text{-}set}(t_1) \& is-\text{Q\text{-}set}(t_1) \rightarrow
\]
\[
\text{int-allocate-list}(\text{tail}(t))
\]
\[
\text{initialize}(\text{s-\text{id}}(t_1), \text{s-\text{init-set}}(\text{attr}_1), \text{da}_1, \text{gen}, \text{env}_1)
\]
\[
\text{call-set-check}(t_1)
\]
\[
\text{gen:area-allocate}(\text{gen-1}, n_1(\text{DN}), \text{s-\text{dens}}(\text{attr}_1), \text{gen-2}, \text{s-area}(t_1))
\]
\[
\text{gen-1:eval-ref-gen}(\text{s-area}(t_1), E)
\]
\[
\text{gen-2:eval-set-gen}(t_1)
\]

T \rightarrow \text{error}

for: \text{is-allocate-list}(t)

Ref.: \text{ctl-initialize} 6-15(31), \text{initialize} 6-16(34), \text{eval-da} 5-13(18),
\text{call-set-check} 6-4(3), \text{eval-ref-gen} 7-25(59), \text{eval-set-gen} 6-8(14)
6.1.1 Allocation of Controlled Variables

Specifications of bounds and lengths used for allocations 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 da-al (which modifies the declared data attributes such that expressions specifying bounds, lengths, and sizes which are also specified in the allocate statement are replaced by *), and the function mix-spec (which merges evaluated bounds, length, and size specifications).

The instruction allocate enters the pointer identifying the new generation in the allocation state of the main storage $S$, and enters the new generation on the top of the stack of generations, associated with the controlled variable, in AG. It returns the new generation for use in the instruction ctl-initialize.

(4) \[\text{allocate}(b, \text{eda}, \text{dens}) = \]

\[\neg \text{is-correct-eda}(\text{eda}) \rightarrow \text{error} \]
\[\text{is-mstg-applic}(p_1, S) \rightarrow \]
\[\text{PASS: mk-gen}(\text{eda}, \text{dens}, I, S) \]
\[s = s \text{ sel-alloc}(p_1, I, S) \]
\[s =_{\text{ag}} \mu(AG; b < \text{mk-gen}(\text{eda}, \text{dens}, I, S) > \wedge b(AG)) \]
\[T \rightarrow \text{mstg-overflow} \]

where: \(p_1 = \text{alloc-space}(\text{eda}, \text{dens}, S)\)

cont'd
Note: $p_1$ is the pointer value identifying that part of $S$ which is reserved on allocation. The kind of storage reserved on allocation is described in chapter 6.1.4.1.

(5) \( (\forall p, stg) (\text{is-mstg-applic}(p, stg) \supset \text{is-usable}(p, \text{size}(stg)) \) 

for: \( \text{is-mstg}(stg) \)

Note: The implementation defined predicate is used for testing whether allocation in main storage is possible.

(6) \( \text{mstg-overflow} = \) 

Note: This instruction is implementation defined. It comprises actions performed on overflow of the main storage.

(7) \( \text{is-correct-eda}(eda) = \)

\begin{align*}
\text{is-array}(eda) & \land s-\text{lbd}(eda) \leq s-\text{ubd}(eda) \land \text{is-correct-eda}.s-\text{elem}(eda) \\
\text{is-struct}(eda) \lor \text{is-cell}(eda) & \implies \text{order}(eda) \\
\text{order}(eda) \geq 1 \land \bigwedge_{i=1}^{\text{order}(eda)} \text{is-correct-eda.s-\text{elem}(i,eda)} \\
\text{is-scalar}(eda) & \implies T 
\end{align*}

Note: This predicate is used to test whether storage can be allocated for the evaluated data attributes $eda$.

(8) \( \text{mk-gen}(eda, dens, p, stg) = \)

\begin{align*}
p = I & \implies \\
\mu^a_0(<s-\text{da};eda>, \\
\mu^a_0(<s-mi;\mu^a_0(<s-\text{da};eda>,<s-dens:dens>), \\
\mu^a_0(<s-\text{pp};\text{alloc-space}(eda, dens, stg)>) \\
T & \implies \\
\mu^a_0(<s-\text{da};eda>, \\
\mu^a_0(<s-\text{da};eda>,<s-dens:dens>), \\
\mu^a_0(<s-\text{pp};(\text{area-alloc-space}(eda, dens, \text{alloc-state}.\text{el-rf}(stg))) \cdot p>) 
\end{align*}

Note: The function constructs the generation defined on allocation of a variable in the part $p(stg)$ of $stg$. If the allocation is in $stg$ itself then $p = I$ and the function alloc-state is used. Otherwise the function area-alloc-space is used.
(9) \[ \text{da-al}(da,al) = \]
\[ \text{is-*}(al) \rightarrow da \]
\[ T \rightarrow \text{da-al-1}(da,al) \]

(10) \[ \text{da-al-1}(da,al) = \]
\[ \text{is-array}(da) \& \text{is-array-al}(al) \& \text{dim}(da) = \text{dim}(al) \rightarrow \]
\[ \mu_o(\langle \text{s-lbd:*}, \text{s-ubd:*}, \text{s-elem:da-al-1}(\text{s-elem}(da),\text{s-elem}(al)) \rangle) \]
\[ \text{is-array}(da) \& \neg \text{is-array-al}(al) \rightarrow \]
\[ \mu(\langle \text{s-elem:da-al-1}(\text{s-elem}(da),al) \rangle) \]
\[ (\text{is-struct}(da) \& \text{is-struct-al}(al) \lor \text{is-cell}(da) \& \text{is-cell-al}(al)) \& \]
\[ \text{order}(da) = \text{order(al)} \rightarrow \]
\[ \mu_o(\langle \text{s-da-elem}(i):da-al-1(s-da-elem(i,da),\text{elem}(i,al)) \rangle | \]
\[ 1 \leq i \leq \text{order}(da) \} \)}
\[ \text{is-string}(da) \& \text{is-string-al}(al) \& \text{s-base}(da) = \text{s-base(al)} \rightarrow \]
\[ \mu(\langle \text{s-length:*} \rangle) \]
\[ \text{is-area}(da) \& \text{is-spec-area}(al) \rightarrow \]
\[ \mu_o(\langle \text{s-size:*} \rangle) \]
\[ \text{is-area}(da) \& \text{is-AREA}(al) \rightarrow \text{AREA} \]
\[ \text{is-scalar}(da) \& \text{is-*}(al) \rightarrow da \]
\[ T \rightarrow \text{error} \]
6.1.2 Allocation of Based Variables

The generation of the pointer variable to be set on allocation is evaluated by the instruction eval-set-gen. If the set-option is not specified in the allocate statement, the generation of the pointer variable specified in the declaration of the based variable is evaluated.

The allocation of a based variable by the instruction based-allocate includes the assignment of the pointer value identifying the new generation to the pointer variable. This pointer is also entered into the free-set of
the current task to insure freeing of the based storage at task end. The instruction based-allocate returns the new generation for use in the instruction initialize.

(13) **based-allocate** (eda, dens, gen) = 

\[ \text{is-mstg-applic}(p_1, S) \rightarrow \]

\[ \text{PASS}: \text{mk-gen}(eda, dens, I, S) \]

\[ s = \text{el-alloc}(p_1, I, \text{el-ass}(s-pp(gen), \text{vr}_{p_1}, S)) \]

\[ s \text{-tp}(TE; <s\text{-based-free-set}; s\text{-based-free-set}(TE) \cup \{p_1\}) \]

\[ T \rightarrow \text{mstg-overflow} \]

where: \( p_1 = \text{alloc-space}(eda, dens, S) \)

\[ \text{vr}_{p_1} = \text{represent}(\text{PTR}, p_1) \]

Ref.: is-mstg-applic 6-5(5), mk-gen 6-5(8), represent 8-9(33)

mstg-overflow 6-5(6), is-active 4-7(11), is-correct-eda 6-5(7)

Note: \( p_1 \) is the pointer value identifying that part of \( S \) which is reserved on allocation. The kind of storage reserved on allocation is described in chapter 6.1.4.1.

(14) **eval-set-gen** (t) = 

\[ \text{is-}Q \circ s\text{-set-ref}(t) \rightarrow \]

\[ \text{eval-ref-gen}(s\text{-set-ref}(t), E) \]

\[ \text{is-}Q \circ s\text{-ptr(attr_id)} \rightarrow \]

\[ \text{eval-ref-gen}(s\text{-ptr(attr_id)}, env_id) \]

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

where: \( \text{attr_id} = s\text{-attr}(s\text{-sel}(<s\text{-id}(t)>)(E)(AT)) \)

\( \text{env_id} = s\text{-e}(s\text{-sel}(<s\text{-id}(t)>)(E)(AT)) \)

for: is-allocate(t) \lor is-locate-st(t) \lor is-set-read(t) \lor is-free(t)

Ref.: eval-ref-gen 7-25(59)
6.1.3 Allocation in Areas

The generation gen-2 of the pointer or offset variable to be set on allocation is evaluated as in chapter 6.1.2. The allocation in the area generation gen-1 by the instruction area-allocate includes the assignment of the pointer value identifying the new generation to this variable. If the variable is an offset variable, the offset of the new generation relative to the area is assigned.

If the AREA condition is raised, the allocation is retried on return from the on-unit, including the evaluation of the area generation from the in-option t.

The instruction area-allocate returns the new generation for use in the instruction initialize.

(15) area-allocate (gen-1, eda, dens, gen-2, t) =

\[ \text{is-area-applic}(o_1, \text{size-1}(p_{g1})) \]

\[ \text{error} \]

\[ \text{is-area-applic}(o_1, \text{size-1}(p_{g1})) \]

\[ \text{PASS:mk-gen(eda, dens, p_{g1}, p_{g2})} \]

\[ \text{s-s :el-alloc(o_1, p_{g1}, el-ass(p_{g2}, ptr-offset(s-da(gen-2), o_1, p_{g1}), \text{S}))} \]

T =

area-allocate (gen-3, eda, dens, gen-2, t);

gen-3: eval-ref-gen(t, \text{p_{g1}});

call-cond (AREA, 603)

where:

- \( o_1 \) = area-alloc-space(eda, dens, alloc-state \cdot el-rf \cdot p_{g1}(\text{S}))
- \( p_{g1} = \text{s-pp}(\text{gen-1}) \)
- \( p_{g2} = \text{s-pp}(\text{gen-2}) \)

for: \text{is-ref(t)}

Ref.: mk-gen 6-5(8), eval-ref-gen 7-25(59), call-cond 9-11(18), is-correct-eda 6-5(7), is-active 4-7(11), size-1 2-22(62)
6.1.4 Storage Mapping

The manipulation of storage and pointers on allocation and reference is determined by the properties of the storage mapping function map. The function

\[
\text{map}(\text{eda}, \text{dens}, i)
\]

gives a pointer value \( p \) such that if a variable with data attributes \( \text{eda} \) and density \( \text{dens} \) is stored in a storage \( \text{stg} \), this pointer value \( p \) identifies the storage \( p(\text{stg}) \) belonging to the \( i \)-th immediate sub-part of that variable.

The properties of the storage mapping function are implicitly defined by the equations presented in this chapter. First, the relation is established between the storage mapping function and the predicate enough-size(\( \text{da}, \text{dens}, z \)), which is true if a variable with data attributes \( \text{eda} \) and density \( \text{dens} \) can be stored in storage of size \( z \). This predicate then is related to the function alloc-space which identifies storage to be used for new allocations. Finally, special properties of map are defined which are essential for certain specific features of PL/I.
6.1.4.1 Storage space reserved on allocation

(19) \( (\forall \text{eda}, \text{dens}, p, \text{stg})(\text{is-applic}(p, \text{stg}) \& \text{is-mstg}(\text{stg}) \supset \text{enough-size}(\text{eda}, \text{dens}, \text{size-l}(p, \text{stg})) \&
(p \in \text{alloc-state}\text{-el}(\text{stg}) \supset \text{is-indep}(\text{p}(\text{stg}), p_1(\text{stg}))) \)

where: \( p_1 = \text{alloc-space}(\text{eda}, \text{dens}, \text{stg}) \)

Note: This equation relates the function alloc-space to the predicate enough-size in stating the requirement that an allocation space identified by the function alloc-space(eda, dens, stg) in main storage must have enough size to store a variable with data attributes eda and density dens, and that it must be independent of all storage spaces identified in the allocation state of stg.

(20) \( (\forall \text{eda}, \text{dens}, p, \text{allst})((p \in \text{allst} \supset \text{is-indep-l}(p, p_1)) \&
\text{enough-size}(\text{eda}, \text{dens}, \text{size-l}(p_1)) \)

where: \( p_1 = \text{area-alloc-space}(\text{eda}, \text{dens}, \text{allst}) \)

Ref.: is-indep-l 2-22(67), size-l 2-22(62)

Note: This equation relates the function area-alloc-space to the predicate enough-size in stating the requirement that an allocation space identified by the function area-alloc-space(eda, dens, allst) in an area must have enough size to store a variable with data attributes eda and density dens, and that it must be independent of all storage spaces identified in the allocation state allst. Note that for allocation in areas the new allocation space only depends on the allocation state of the area, i.e. not on the size of the area.

(21) \( \text{enough-size}(\text{eda}, \text{dens}, z) =
\begin{align*}
&\text{is-scalar}(\text{eda}) \rightarrow \text{enough-scalar-size}(\text{eda}, z) \\
&T \rightarrow \\
&\text{ubd}(\text{eda}) \rightarrow \\
&\text{enough-size}(\text{da-part}(\text{eda}, i), \text{dens}, \text{size-l-map}(\text{eda}, \text{dens}, i)) \\
&\text{i-lpd}(\text{eda}) \\
\end{align*}
\)

Ref.: size-l 2-22(62)

Note: The predicate is true if there is enough size for all immediate subparts of a variable with data attributes eda and density dens, or, if eda is scalar, if the implementation-defined predicate enough-scalar-size is true.
(22) \((\forall \text{eda}, \text{dens}, i, j)(\text{ldb(eda)} \leq i, j \leq \text{ubd(eda)} \land i \neq j \land 
\neg \text{is-cell(eda)} \Rightarrow \text{is-indep-l(map(eda, dens, i), map(eda, dens, j)))})\)

Ref.: is-indep-l 2-22(67)

Note: This equation assures that all immediate sub-parts of a variable, except when the variable is a cell, are given mutually independent storage.

(23) \(\text{enough-scalar-size(eda, z)} = \)

\(\text{vr-set(eda)} \subseteq \text{range(z)}\)

for: is-scalar(eda)

Ref.: range 2-23(71)

Note: The set of value representations associated with a scalar attribute in a specific implementation must be a subset of the range of value representations that can be stored in the storage reserved for a variable having this attribute.

(24) \(\text{vr-set(eda)} = \)

\(\text{is-string(eda)} \land \text{is-CHAR-s-base(eda)} \land \text{is-\text{\textit{Q}}-s-varying}(\text{eda}) \lor \text{is-pic(eda)} \land \neg \text{is-bin-pic(eda)} \rightarrow \)

\(\{\text{vrl} \mid \text{is-char-vr-list}(\text{vrl}) \land \text{length}(\text{vrl}) = \text{str-length}(\text{eda})\}\)

\(\text{is-string(eda)} \land \text{is-BIT-s-base(eda)} \land \text{is-\text{\textit{Q}}-s-varying}(\text{eda}) \lor \text{is-bin-pic(eda)} \rightarrow \)

\(\{\text{vrl} \mid \text{is-bit-vr-list}(\text{vrl}) \land \text{length}(\text{vrl}) = \text{str-length}(\text{eda})\}\)

\(\text{is-string(eda)} \land \text{is-CHAR-s-base(eda)} \land \neg \text{is-\text{\textit{Q}}-s-varying}(\text{eda}) \rightarrow \)

\(\{\text{vrl} \mid \text{is-char-vr-list}(\text{vrl}) \land \text{length}(\text{vrl}) \leq \text{s-length}(\text{eda})\}\)

\(\text{is-string(eda)} \land \text{is-BIT-s-base(eda)} \land \neg \text{is-\text{\textit{Q}}-s-varying}(\text{eda}) \rightarrow \)

\(\{\text{vrl} \mid \text{is-bit-vr-list}(\text{vrl}) \land \text{length}(\text{vrl}) \leq \text{s-length}(\text{eda})\}\)

\(T \rightarrow \text{non-string-vr-set(eda)}\)

Ref.: str-length 8-40(157), is-bit-vr 8-5(16), is-char-vr 8-5(16)

Note: is-bit-vr and is-char-vr are implementation defined predicates associated with value representations of bits and characters, respectively. non-string-vr-set is an implementation defined function giving the set of value representations associated with non-string scalar data attributes.
6.1.4.2 Storage space reserved on buffer allocation

For allocation of a buffer (see chapter 10.4.3) the storage space is not determined by data attributes and density of a variable but by the size of a record. The function buffer-space(z, stg) gives a pointer identifying the storage space reserved on allocation, depending on the size z and on the storage stg in which the allocation is made.

\[(\forall z, p, stg) (\text{is-applic(buffer-space(z, stg), stg)} \implies \text{size(buffer-space(z, stg)(stg)) = z} \land (p \in \text{alloc-state-el-rf(stg)} \implies \text{is-indep(p(stg), buffer-space(z, stg)(stg))))\]

Note: The function buffer-space has the property that it gives a pointer such that the associated storage space is independent of all storage spaces identified by the allocation state of the storage in which the buffer allocation is made.

6.1.4.3 Special properties of the mapping function

6.1.4.3.1 String overlay defining

Overlay defining of strings requires the storage mapping of packed string aggregates to be independent of their structuring. This property is expressed by the following equations.

\[(\forall \text{eda-1, eda-2, rl-1, rl-2}) ((\text{is-bit-aggr(eda-1)} \& \text{is-bit-aggr(eda-2)} \lor \text{is-char-aggr(eda-1)} \& \text{is-char-aggr(eda-2)}) \land (\text{is-non-varying(eda-1)} \& \text{is-non-varying(eda-2)} \lor \text{string-index(eda-1, rl-1)} = \text{string-index(eda-2, rl-2)} \land \text{string-extent\text{-}sub-eda(eda-1, rl-1)} = \text{string-extent\text{-}sub-eda(eda-2, rl-2)}) = (\text{string-ref\text{-}refer(eda-1, rl-1)} = \text{string-ref\text{-}refer(eda-2, rl-2)})\]

Ref.: is-bit-aggr 7-30(76), is-char-aggr 7-30(77), is-non-varying 7-29(75), string-extent 7-30(78), sub-eda 7-33(88)

Note: string-ref\text{-}refer(eda, rl) is defined for string aggregates and gives a pointer value identifying the storage associated with that part of the aggregate which is determined by the reference list rl (see chapter 7.3.5). The equation says that this pointer depends only on the extent of the sub-part and its position in the aggregate.
(27) \[
\text{string-ref er(eda,rl)} = \\
\quad \text{is-<>}(rl) \implies I \\
\quad T \implies \\
\quad (\text{string-ref er(da-part(eda,head(rl)),tail(rl)))) \cdot \text{map(eda,PACKED,head(rl))}
\]

(28) \[
\text{string-index(eda,rl)} = \\
\quad \text{is-<>}(rl) \implies 0 \\
\quad T \implies \\
\quad \sum_{i=\text{lbd(eda)}}^{\text{head(rl)}} \text{string-extent(da-part(eda,i))} + \\
\quad \text{string-index(da-part(eda,head(rl)),tail(rl))}
\]

Ref.: string-extent 7-30(78)

6.1.4.3.2 String assignment

There is a relationship between the value representation of a storage associated with a fixed length string variable, and the value representations of the individual bit- or character-parts of that storage. This means that on assignment of a fixed length string also the elementary (bit or character) parts of the storage to which the assignment is made, get defined vr's.

(29) \[
\forall (eda, \text{stg}) (\text{is-string(eda)} \& \text{is-\Omega-s-varying(eda)} \& \text{enough-size(eda,PACKED,size(stg))} \supset \\
\quad \text{el-rf(stg)} = \bigcup_{i=1}^{\text{s-length(eda)}} \text{el-rf(map(eda,PACKED,i))(stg)})
\]

Ref.: enough-size 6-11(21)

6.1.4.3.3 Left-to-right equivalence

The rules saying which based variable may be combined with which pointer in order to get defined results follow from the properties of the storage mapping function. Besides the general rule, that defined results are obtained for complete matching of the attributes of the based variable and of the allocation which is identified by the pointer, there are certain other defined cases. An additional constraint on the mapping function is required by the
following equation. The consequence is that the matching of based variable and pointer for structures is required only up to the point of reference (left-to-right equivalence rule).

\[(\forall \text{eda}, i, \text{dens})(\text{is-struct}(\text{eda}) \& \text{lsisorder}(\text{eda}) \Rightarrow \\
\text{map}(\text{eda}, \text{dens}, i) = \\
\text{map}(\mu_{\emptyset}(\langle \text{elem}(j) : \text{elem}(j, \text{eda}) \rangle \mid 1 \leq j \leq 1), \text{dens}, i))\]

6.2 Initialization of Variables

6.2.1 Initialization of controlled variables

Controlled variables are initialized by the instruction

\[
\text{ctl-
initialize}(\text{id}, \text{iss}, \text{isd}, \text{gen}, \text{env}),
\]

where \text{id} is the identifier of the controlled variable, \text{iss} and \text{isd} are the initial sets specified in the allocate statement and in the declaration, respectively, \text{gen} is the generation to be initialized, and \text{env} is the environment in which the controlled variable was declared.

First, all areas contained in the controlled variable are initialized, i.e. the allocation states of the areas are set empty. The initial attributes specified in the allocate statement and those in the declaration of the controlled variable are then merged, with the rule that a specification in the statement overrides a specification in the declaration.

\[(\exists \text{ctl-
initialize}(\text{id}, \text{iss}, \text{isd}, \text{gen}, \text{env}) = \\
\text{ctl-
initialize-1}(\text{id}, \text{iss}, \text{isd}, \text{gen}, \text{env}); \\
\text{area-
initialize}(\text{gen})\]

\[(\exists \text{ctl-
initialize-1}(\text{id}, \text{iss}, \text{isd}, \text{gen}, \text{env}) = \\
\text{null}; \\
\{\text{init}(\text{id}, \text{s-ref}(x), \text{s-init-spec}(x), \text{da}_{\text{id}}, \text{gen}, \text{env}) \mid x \in \text{iss}\} \cup \\
\{\text{init}(\text{id}, \text{s-ref}(x), \text{s-init-spec}(x), \text{da}_{\text{id}}, \text{gen}, \text{env}) \mid x \in \text{isd} \& \neg \exists y(y \in \text{iss} \& \text{s-ref}(x) = \text{s-ref}(y))\}
\]

where: \text{da}_{\text{id}} = \text{s-da-s-attr}(\text{sel}(\langle \text{id}\rangle)(\text{E})(\text{AT}))

Ref. : \text{init} 6-16(36)
6.2.2 Initialization of non-controlled variables

Non-controlled variables are initialized by the instruction
\( \text{initialize}(id, isd, da, gen, env) \), where \( id \) is the identifier, \( isd \) the initial set specified in the declaration, \( da \) the data attributes of the variable, \( gen \) the generation to be initialized, and \( env \) the environment in which the variable was declared.

\[ \text{(34) } \text{initialize}(id, isd, da, gen, env) = \]
\[ \text{initialize}_1(id, isd, da, gen, env); \]
\[ \text{area-initialize}(gen) \]

\[ \text{(35) } \text{initialize}_1(id, isd, da, gen, env) = \]
\[ \text{null}; \]
\[ \{ \text{init}(id, s-ref(x), s-init-spec(x), da, gen, env) | x \in isd \} \]

\[ \text{(36) } \text{init}(id, t, init-spec, da, gen, env) = \]
\[ n_t = \text{sel}(<id>)(env) \land (\text{is-<>}(sl_t) \lor \text{is-label}(dar_t, da)) \land \text{is-scalar-base-elem}(dar_t, da) \land \text{is-init-elem-list}(init-spec) \]
\[ \text{initial-assign-list}(gen_l, init-spec, env); \]
\[ \text{gen-l:pass-gen-list}(gen); \]
\[ \text{gen-l:pass-sub-gen}(gen, rl); \]
\[ \text{rl:mk-rl}(s-da(gen), sl_t, eql_t, da', env) \]
\[ n_t = \text{sel}(<id>)(env) \land \text{is-<>}(sl_t) \land \text{is-scalar-base-elem}(dar_t, da) \land \text{is-call-st}(init-spec) \land \text{is-Q: s-pa-opt}(init-spec) \]
\[ \text{int-call-st}(init-spec, env) \]
\[ T \rightarrow \text{error} \]

\( \text{where: } \text{eql}_{t, da} = \text{eval-ql}(\text{tail} \cdot s-id-list(t), da) \)
\( \text{dar}_{t, da} = \text{sub-da}(da, sl_t, eql_t, da) \)
for: is-ref(t)

Ref.: base-elem 7-28(71), sub-gen 7-32(84), mk-rl 7-26(62),
     int-call-st 5-32(51), eval-ql 7-27(66), sub-da 7-34(89)

(37) "initial-assign-list"(genl,init-spec,env) =
    is-<>(genl) v is-<>(init-spec) → null
    is-init-iter*head(init-spec) →
    initial-assign-list(genl,init-spec-1,env);
    init-spec-1:pass-rep-conc(1,s-item-list:head(init-spec),
                             tail(init-spec));
    i:eval-intg-expr(s:rep-factor*head(init-spec),env)

    is-•*head(init-spec) →
    initial-assign-list(tail(genl),tail(init-spec),env)

    is-expr*head(init-spec) →
    initial-assign-list(tail(genl),tail(init-spec),env);
    convert-assign(head(genl),op);
    op:eval-expr(head(init-spec),env)

Ref.: eval-intg-expr 7-21(52), convert-assign 7-9(19), eval-expr 7-18(34)

(38) rep-conc(i,list-1,list-2) =

    \( (\bigcup_{k=1}^{i} list-1) \cap list-2 \)

(39) gen-list(gen) =
    is-scalar*gen-da(gen) → <gen>
    is-array*gen-da(gen) v is-struct*gen-da(gen) →
    ubd.*gen-da(gen)
    gen-list*sub-gen(gen,<i>)

Ref.: sub-gen 7-32(84)
6.3 The Free Statement

The free statement is interpreted by the instruction \( \text{int-free-st}(t) \), where \( t \) is the text of the statement. The frees specified in the statement are interpreted from left to right. The distinction is made between the freeing of controlled variables, the freeing of based variables in main storage, and the freeing of based variables in areas.

\[
\text{int-free-st}(t) = \\
\text{int-free-list}(s\text{-free-list}(t))
\]

for: \( \text{is-free-st}(t) \)

\[
\text{int-free-list}(t) = \\
is<>(t) \rightarrow \text{null}
\]

\[
is\text{-CTL}\cdot s\text{-stg\text{-class}}(\text{attr}_1) \& \text{is-\text{Q}\text{-s\text{-set-ref}}}(t_1) \& \text{is-\text{Q}\text{-s\text{-area}}}(t_1) \& \\
\neg \text{is-active}(\text{gen}_1,\text{PA}) \rightarrow \\
\text{int-free-list}(\text{tail}(t));
\]

\[
\text{free}(n_1(\text{DN}))
\]

\[
is\text{-based}(\text{attr}_1) \& \text{is-\text{Q}\text{-s\text{-area}}}(t_1) \rightarrow \\
\text{int-free-list}(\text{tail}(t));
\]

\[
\text{based-free}(n_1(\text{DN}),s\text{-dens}\text{-class}(\text{attr}_1),\text{op});
\]

\[
\text{op}:\text{pass-gen-op}(\text{gen},\text{S});
\]

\[
\text{gen}:\text{eval-set-gen}(t_1)
\]

\[
is\text{-based}(\text{attr}_1) \rightarrow \\
\text{int-free-list}(\text{tail}(t));
\]

\[
\text{area-free}(\text{gen-1},n_1(\text{DN}),s\text{-dens}(\text{attr}_1),\text{op});
\]

\[
\text{gen-1}:\text{eval-ref-gen}(\text{s\text{-area}}(t_1),\text{E}),
\]

\[
\text{op}:\text{pass-gen-op}(\text{gen-2},\text{S});
\]

\[
\text{gen-2}:\text{eval-set-gen}(t_1)
\]

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

for: \( \text{is-free-list}(t) \)

Ref.: gen-op 7-19(47), eval-set-gen6-8(14), eval-ref-gen 7-25(59)

is-active 4-7(11)
(42) \( \text{free}(b) = \)
\[
\begin{align*}
\text{length}\cdot b(AG) & > 1 \quad \rightarrow \\
\text{s-s : el-free(}\{s-p\cdot \text{head}\cdot b(AG)\},I,S) & \\
\text{s-ag : } \mu(AG; b:\text{tail}\cdot b(AG)) & \\
is-\text{NULL}\cdot \text{head}\cdot b(AG) & \rightarrow \text{null} \\
T & \rightarrow \text{error}
\end{align*}
\]

(43) \( \text{based-free}(eda,dens,op) = \)
\[
\begin{align*}
(\exists \text{stg}) (\text{is-mstg}(\text{stg}) & \& \text{alloc-space}(da,dens,\text{stg}) = p_\text{op}) & \& \\
\text{is-PTR}\cdot s\cdot da(\text{op}) & \& p_\text{op} \in \text{alloc-state}\cdot \text{el}\cdot \text{rf}(S) & \& \\
p_\text{op} \in \text{s-based-free-set}(TE) & \& \text{is-active}(\text{gen}_\text{op},\text{PA}) \quad \rightarrow \\
\text{s-s : el-free}(\{p_\text{op}\},I,S) & \\
\text{s-te : } \mu(\text{TE}; \{\text{s-based-free-set} : \text{s-based-free-set}(\text{TE}) - \{p_\text{op}\}\}) & \\
T & \rightarrow \text{error}
\end{align*}
\]

where: \( p_\text{op} = \text{op-val}(\text{op}), \)
\( \text{gen}_\text{op} = \mu(\langle s\cdot da:eda, s\cdot \text{m}=m_0(\langle s\cdot da:eda, s\cdot dens:dens\rangle), \rangle, \langle s\cdot pp:p_\text{op}\rangle) \)

Ref.: op-val 8-11(40), is-active 4-7(11)

(44) \( \text{area-free}(\text{gen},eda,dens,op) = \)
\[
\begin{align*}
\text{is-area}\cdot s\cdot da(\text{gen}) & \& \text{is-PTR}\cdot s\cdot da(\text{op}) & \& (\exists \text{allst})(\text{is-ptr-val-set}(\text{allst}) & \& \\
\text{alloc-space}(da,dens,\text{allst}) = o_\text{op}) & \& \\
o_\text{op} \in \text{alloc-state}\cdot \text{el}\cdot \text{rf}(s\cdot pp(\text{gen})(S)) & \& \neg \text{is-active}(\text{gen}_\text{op},\text{PA}) \quad \rightarrow \\
\text{s-s : el-free}(\{o_\text{op}\},s\cdot pp(\text{gen}),S) & \\
T & \rightarrow \text{error}
\end{align*}
\]

where: \( o_\text{op} = (\text{o}) (\text{op-val}(\text{op}) = o\cdot s\cdot pp(\text{gen})) \)
\( \text{gen}_\text{op} = \mu(\langle s\cdot da:eda, s\cdot \text{m}=m_0(\langle s\cdot da:eda, s\cdot dens:dens\rangle), \rangle, \langle s\cdot pp:op-val(\text{op})\rangle) \)

Ref.: is-ptr-val 2-21(52), op-val 8-11(40), is-active 4-7(11)
The first section of this chapter presents the mechanism for handling aggregate assignment statements, including BY NAME assignment, the syntactic modification of expressions used in the expansion of assignment statements, and the definition of scalar and area assignment statements. The second section defines the evaluation of scalar expressions down to the level of scalar data operations. These are described in chapter 8. The third section defines the reference to variables including proper variables, defined variables, and based variables.

Metavariables

<table>
<thead>
<tr>
<th>attr</th>
<th>is-attr</th>
<th>attributes of a variable</th>
</tr>
</thead>
<tbody>
<tr>
<td>b</td>
<td>is-n</td>
<td>aggregate name</td>
</tr>
<tr>
<td>da</td>
<td>is-da</td>
<td>data attributes of a variable</td>
</tr>
<tr>
<td>dens</td>
<td>is-PACKED v is-ALIGNED</td>
<td></td>
</tr>
<tr>
<td>eda</td>
<td>is-eda</td>
<td>evaluated data attributes of a variable</td>
</tr>
<tr>
<td>edal</td>
<td>is-eda-list</td>
<td>list of evaluated data attributes</td>
</tr>
<tr>
<td>env</td>
<td>is-e</td>
<td>environment</td>
</tr>
<tr>
<td>eql</td>
<td>is-intg-val-list</td>
<td>evaluated list of name qualifiers</td>
</tr>
<tr>
<td>ert</td>
<td>is-e-ret-type</td>
<td>evaluated return type</td>
</tr>
<tr>
<td>expr</td>
<td>is-expr</td>
<td>expression</td>
</tr>
<tr>
<td>gen</td>
<td>is-gen</td>
<td>generation of a variable</td>
</tr>
<tr>
<td>genl</td>
<td>is-gen-list</td>
<td>list of generations</td>
</tr>
<tr>
<td>i</td>
<td>is-intg-val</td>
<td>integer value</td>
</tr>
<tr>
<td>k</td>
<td>is-intg-val</td>
<td>integer value</td>
</tr>
<tr>
<td>list</td>
<td>is-list</td>
<td></td>
</tr>
<tr>
<td>lp</td>
<td>is-ref-list</td>
<td>left-part of an assignment statement</td>
</tr>
<tr>
<td>n</td>
<td>is-n</td>
<td>unique name of a function</td>
</tr>
<tr>
<td>op</td>
<td>is-op</td>
<td>operand</td>
</tr>
<tr>
<td>opor</td>
<td>is-operator</td>
<td>data operator</td>
</tr>
<tr>
<td>pp</td>
<td>is-pp</td>
<td>pointer part of a generation</td>
</tr>
<tr>
<td>ql</td>
<td>is-id-list</td>
<td>list of name qualifiers</td>
</tr>
<tr>
<td>rl</td>
<td>is-r-list, where :s-r = is-intg-val v is-*</td>
<td>list of evaluated subscripts and/or name qualifiers</td>
</tr>
<tr>
<td>sl</td>
<td>is-arg-exp-list</td>
<td>list of subscript expressions</td>
</tr>
<tr>
<td>stg</td>
<td>is-stg</td>
<td>storage</td>
</tr>
<tr>
<td>t</td>
<td>is-assign-st v is-expr</td>
<td>text of assignment statement or expression</td>
</tr>
<tr>
<td>x</td>
<td>is-selector function</td>
<td></td>
</tr>
</tbody>
</table>
Abbreviations

for is-ref(t) and is-e(env):

\[ sl_t = s-arg-list(t) \]  
\[ ql_t = tail*s-id-list(t) \]  
\[ id_t = head*s-id-list(t) \]  
\[ eql_t = eval-ql(ql_t,dat) \]  
\[ n_t = sel*s-id-list(t)(env) \]  
\[ attr_t = s-attr*n_t(AT) \]  
\[ env_t = s-e*n_t(AT) \]  
\[ da_t = s-da(attr_t) \]  
\[ dens_t = s-dens(attr_t) \]  
\[ eda_t = \begin{cases} 
& \text{is-prop-var}(attr_t) \rightarrow s-da=head(n_t(DN)(AG)), \\
& \text{is-defined}(attr_t) \lor \text{is-based}(attr_t) \rightarrow n_t(DN) 
\end{cases} \]  
\[ dar_t = \text{sub-da}(da_t,sl_t,eql_t) \]  
\[edar_t = \text{sub-da}(eda_t,sl_t,eql_t) \]

Note: If the referenced variable had not been declared then \( n_t = \emptyset \). It is an error if \( n_t \) is applied as a selector in this case.

7.1 The Assignment Statement

The assignment statement is interpreted by the instruction \( \text{int-assi gn-st}(t) \), where \( t \) is the text of the statement. For non-scalar assignment statements the text is iteratively expanded into a sequence of modified texts. Each modifying step replaces references to aggregate variables occurring in the text by references to an immediate sub-part of these aggregate variables. The expansion is governed by the data attributes of the left-most reference in the left-part of the assignment statement. For assignment statements with \( \text{BY NAME} \) option, the evaluated data attributes (containing evaluated array bounds) as well as the non-evaluated data attributes (containing identifiers of sub-aggregates) of this master reference are needed. For assignment statements without \( \text{BY NAME} \) option only the evaluated data attributes are necessary.

A scalar assignment statement is interpreted by evaluating the generations of the references of the left-part in the order left-to-right, evaluating the right-part expression, and assigning the resulting operand to the
left-part generations in the order left-to-right. Before the actual assignment to a generation the operand to be assigned is converted to the data attributes of the generation. The conversion of operands is defined in chapter 8.

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.

\[
\text{int-assign-st}(t) = \begin{align*}
&\text{is-}\neg\circ s\text{-byname}(t) \rightarrow \\
&\text{call-check-cond}(s\text{-lp}(t), E, 01); \\
&\text{int-assign}(s\text{-lp}(t), s\text{-rp}(t), \text{edal}); \\
&\text{edal} : \text{eda-ref-list}(s\text{-lp}(t), E)
\end{align*}
\]

\[
T \rightarrow \\
\text{call-check-cond}(s\text{-lp}(t), E, 502); \\
\text{int-byname}(s\text{-lp}(t), s\text{-rp}(t))
\]

for: \text{is-assign-st}(t)

Ref.: \text{call-check-cond} 9-15(27)

Note: The list of evaluated data attributes of the left-part references, provided by the instruction \text{eda-ref-list}, is needed for testing the consistency of the structuring of the left-part references. For assignment statements with BY NAME option the data attributes of the left-part references are determined at each step of the expansion, cf. 7.1.2.

(2) \text{eda-ref-list}(lp, \text{env}) = 

\[
\text{is-\neg}(lp) \rightarrow \text{PASS:}\neg \\
T \rightarrow \\
\text{mk-list}(\text{eda}, \text{edal}); \\
\text{edal} : \text{eda-ref-list}(\text{tail}(lp), \text{env}), \\
\text{eda} : \text{eda-ref}(\text{head}(lp), \text{env})
\]

Ref.: \text{eda-ref} 7-24(58)
7.1.1 Non BY NAME expansion

The expansion is governed by the data attributes head(\textit{edal}) of the leftmost variable of the left-part. The effect of the expansion is the sequential execution of scalar assignment statements.

(3) \texttt{int-assign}(lp,expr,edal) =

\texttt{is-scalar\cdot head}(edal) \rightarrow
\texttt{convert-assign-list}(genl,op);
\texttt{op:eval-expr(expr,E)};
\texttt{genl:eval-lp-list}(lp)

\texttt{Length}(edal) \texttt{is-array\cdot elem(i,edal)} \lor \texttt{is-struct\cdot elem(i,edal)} \rightarrow
\texttt{iterate-assign}(lp,expr,edal,lbd\cdot head(edal))

\texttt{T} \rightarrow \texttt{error}

Ref.: \texttt{convert-assign-list} 7-9(18), \texttt{eval-expr} 7-18(43)

(4) \texttt{eval-lp-list}(lp) =

\texttt{is-<>}(lp) \rightarrow \texttt{PASS:<>}
\texttt{T} \rightarrow
\texttt{mk-list}(gen,genl);
\texttt{genl:eval-lp-list}(\texttt{tail}(lp));
\texttt{gen:eval-lp}(\texttt{head}(lp))

(5) \texttt{eval-lp}(ref) =

\texttt{is-builtin(attr}_{\textit{ref}} \rightarrow \texttt{eval-ps-lp}(ref)
\texttt{is-var(attr}_{\textit{ref}} \rightarrow \texttt{eval-ref-gen}(ref,E)
\texttt{T} \rightarrow \texttt{error}

where: \texttt{attr}_{\textit{ref}} = \texttt{s-attr} (\texttt{sel\cdot s-id-list}(ref)(E)(AT))

Ref.: \texttt{eval-ps-lp} 7-12(27), \texttt{eval-ref-gen} 7-25(59)

Note: The instruction \texttt{eval-ps-lp} evaluates a pseudo-generation, in case the reference is a pseudo variable. The instruction \texttt{eval-ref-gen} returns a generation.
(6) \texttt{iterate-assign}(lp, expr, eda, i) = \\
\begin{align*}
& i > \text{ubd} \cdot \text{head}(eda) \rightarrow \text{null} \\
&T \rightarrow \\
& \text{iterate-assign}(lp, expr, eda, i+1); \\
& \text{int-assign}(t-1, t-2, \text{da-part-list}(eda,i)); \\
& \{ \text{elem}(j)(t-1) \mod (\text{elem}(j,lp), \text{head}(eda,i,E)) | \lceil j \rceil \leq \text{length}(lp) \} \\
& \{ t-2 : \mod (expr, \text{head}(eda), i, E) \} \\
\end{align*}

Ref.: \texttt{mod} 7-7(13)

(7) \texttt{da-part-list}(eda, i) = \\
\begin{align*}
& \text{length}(eda) \\
& \begin{array}{c}
\text{LIST} \\
\hline k = 1 \\
\end{array} \\
& \text{da-part}(\text{elem}(k, eda), i)
\end{align*}

7.1.2 BY NAME Expansion

The expansion is governed by the data attributes of the leftmost variable of the left-part. In case this variable is an array the expansion works as in the non BY NAME case. If it is a structure, the names of the sub-aggregates of this structure are used to construct references to sub-aggregates of the aggregate variables in the left-part and in the expression. The effect of the expansion is the sequential execution of scalar assignment statements.

(8) \texttt{int-bynamew2}(t-1, t-2) = \\
\begin{align*}
& \texttt{int-bynamew}(t-1, t-2, eda, da); \\
& \text{eda: eda-ref-list}(t-1, E), \\
& \text{da: da-ref}(\text{head}(t-1), E)
\end{align*}

Ref.: \texttt{eda-ref-list} 7-3(2)

(9) \texttt{da-ref}(t, env) = \\
\begin{align*}
& \text{is-var}(\text{attr}_t) \rightarrow \text{dar}_t \\
& \text{is-builtin}(\text{attr}_t) \& \text{is-expr elem}(1, \text{sl}_t) \rightarrow \text{da-ref}(\text{elem}(1, \text{sl}_t), \text{env}) \\
&T \rightarrow \text{error}
\end{align*}

for: \text{is-ref}(t) \\
Ref.: \text{is-builtin} 2-18(40)

Note: If \( t \) is the reference to a pseudo variable, the first argument is taken as the master reference.
(10) \text{int-byname-1}(lp, expr, edal, da) = \\
\quad \text{is-\ast (expr) } \vee (\exists\, i)(\text{is-\ast \text{elem}(i, lp)} \longrightarrow \text{null} \\
\quad \text{is-scalar\text{-}head(edal)} \longrightarrow \\
\quad \text{convert assign-list(\text{genl}, \text{op});} \\
\quad \text{op:eval-expr(expr, E);} \\
\quad \text{genl:eval-1p-list(lp)} \\
\quad \text{length(edal)} \\
\quad \text{i} \leq \text{is-array\text{-}elem(i, edal)} \vee \text{i} \leq \text{is-struct\text{-}elem(i, edal)} \\
\quad \text{iterate-byname(lp, expr, edal, da, lbd\text{-}head(edal))} \\
\quad \text{T} \longrightarrow \text{error} \\
\quad \text{for: } \text{is-arg-expr-list(lp),} \\
\quad \text{is-arg-expr(expr)} \\
\quad \text{Ref.: eval-1p-list 7-4(4), convert assign-list 7-9(18),} \\
\quad \text{eval-expr 7-18(43)} \\
\quad \text{Note: The first alternative of the definition (the expression or one of the} \\
\quad \text{elements of the left-part being \ast ) accounts for the case that the con} \\
\quad \text{struction of a modified expression or reference with a specific sub} \\
\quad \text{aggregate name has not been possible (i.e. not each reference contained} \\
\quad \text{that sub-aggregate name). The instruction mod returns * in this case.} \\

(11) \text{iterate-byname}(lp, expr, edal, da, i) = \\
\quad \text{i} \leq \text{ubd\text{-}head(edal)} \longrightarrow \text{null} \\
\quad \text{T} \\
\quad \text{iterate-byname}(lp, expr, edal, da, i+1); \\
\quad \text{int-byname(t-1, t-2);} \\
\quad \{\text{elem}(j)(t-1):\text{mod(elem}(j, lp), \text{head(edal), aggr-part(da, i), E}) | \\
\quad \text{1} \leq j \leq \text{length(lp)}\} \cup \\
\quad \{t-2:\text{mod(expr, head(edal), aggr-part(da, i), E})}\} \\
\quad \text{Ref.: mod 7-7(13)} \\

(12) \text{aggr-part(da, i) =} \\
\quad \text{is-array(da) } \longrightarrow \text{ i} \\
\quad \text{is-struct(da) } \longrightarrow \text{ s-id\text{-}elem(i, da)} \\
\quad \text{Note: For BY NAME expansion, if the master-variable is a structure, the} \\
\quad \text{name of the sub-aggregate in the master-variable is given to the} \\
\quad \text{modifying instruction mod. In all other cases it is the number of} \\
\quad \text{the sub-aggregate.}
7.1.3 Syntactic Modification of Expressions

\[ \text{mod}(t, \text{eda}, i, \text{env}) = \]

\[ \begin{align*}
\text{is-infix-expr}(t) \rightarrow & \\
\text{mk-infix}(\text{op-1}, \text{op-2}, \text{s-operator}(t)); & \\
\text{op-1} : \text{mod}(\text{s-op-1}(t), \text{eda}, i, \text{env}) , & \\
\text{op-2} : \text{mod}(\text{s-op-2}(t), \text{eda}, i, \text{env}) & \\
\end{align*} \]

\[ \text{is-prefix-expr}(t) \rightarrow & \\
\text{mk-prefix}(\text{op}, \text{s-operator}(t)); & \\
\text{op} : \text{mod}(\text{s-op}(t), \text{eda}, i, \text{env}) & \\
\]

\[ \text{is-paren-expr}(t) \rightarrow & \\
\text{mk-paren}(\text{op}); & \\
\text{op} : \text{mod}(\text{s-op}(t), \text{eda}, i, \text{env}) & \\
\]

\[ \text{is-op}(t) \rightarrow \text{PASS}:t \]

\[ \text{is-isub}(t) \rightarrow \text{error} \]

\[ \text{is-builtin}(\text{attr}_t) \rightarrow \]

\[ \text{mod-builtin}(t, \text{eda}, i, \text{env}) \]

\[ \text{is-var}(\text{attr}_t) \& \text{is-array}(\text{edar}_t) \& \text{is-array}(\text{eda}) \& \text{dim}(\text{edar}_t) = \text{dim}(\text{eda}) \& \text{s-lbd}(\text{edar}_t) = \text{s-lbd}(\text{eda}) \& \text{s-ubd}(\text{edar}_t) = \text{s-ubd}(\text{eda}) \rightarrow \]

\[ \text{PASS}:\mu(t; <\text{s-arg-list:insert}(s-arg-list(t), \text{bintg-op}(i))>) \]

\[ \text{is-var}(\text{attr}_t) \& \text{is-struct}(\text{dar}_t) \& \text{is-struct}(\text{eda}) \& \text{order}(\text{dar}_t) = \text{order}(\text{eda}) \& \text{is-intg-val}(i) \rightarrow \]

\[ \text{PASS}:\mu(t; <\text{s-id-list:s-id-list}(t) < s-id:elem(i, \text{dar}_t)>)>) \]

\[ \text{is-var}(\text{attr}_t) \& \text{is-struct}(\text{dar}_t) \& \text{is-struct}(\text{eda}) \& \text{is-id}(i) \& \]

\[ (\exists k)(\text{s-id:elem}(k, \text{dar}_t) = i) \rightarrow \]

\[ \text{PASS}:\mu(t; <\text{s-id-list:s-id-list}(t) < i>)>) \]

\[ \text{is-var}(\text{attr}_t) \& \text{is-struct}(\text{dar}_t) \& \text{is-struct}(\text{eda}) \& \text{is-id}(i) \rightarrow \]

\[ \text{PASS}:* \]

\[ (\text{is-var}(\text{attr}_t) \& \text{is-struct}(\text{dar}_t) \& \text{is-array}(\text{eda})) \lor \]

\[ (\text{is-var}(\text{attr}_t) \& \text{is-scalar}(\text{dar}_t) \lor \text{is-entry}(\text{attr}_t) \lor \text{is-generic}(\text{attr}_t) \lor \]

\[ \text{is-LABEL}(\text{attr}_t) \lor \text{is-format-attr}(\text{attr}_t)) \& \text{is-intg-val}(i) \rightarrow \]

\[ \text{PASS}:t \]

\[ T \rightarrow \text{error} \quad \text{cont'd} \]
for: is-expr(t), is-intg-val(i) v is-id(i)
Ref.: mod-built in 11-2(1), bintg-op 8-11(43)
   is-built in 2-18(40), is-op 8-11(38)
Note: If i is an identifier, the instruction returns * if no sub-aggregate
   having this identifier is contained in one of the operands.

(14) insert(list,op) =

   is-<>(list)   ----> <op>
   is-<>head(list) ----> <op> tail(list)
   T ----> <head(list) insert(tail(list),op)

(15) mk-infix(op-1,op-2,opor) =

   is-*(op-1) v is-*(op-2) ----> PASS:* 
   T ----> PASS: μ_{o} (<s-op-1:op>,<s-op-2:op-2>,
   <s-operator:opor>)

(16) mk-prefix(op,opor) =

   is-*(op) ----> PASS:* 
   T ----> PASS: μ_{o} (<s-op:op>,<s-operator:opor>)

(17) mk-paren(op) =

   is-*(op) ----> PASS:* 
   T ----> PASS: μ_{o} (<s-op:op>)

7.1.4 Scalar Assignment

The assignment of the operand resulting from the evaluation of the
right-part of a scalar assignment statement to the list of generations re-
sulting from the evaluation of the left-part is done in the order left-to-
right by the instruction convert-assign-list. In the instruction convert-assign
the distinction is made between assignment to pseudo-variables, area assign-
ment, and proper scalar assignment. The instruction assign in proper scalar
assignment is always preceded by the instruction convert, which performs
the conversion of the operand to the data attributes of the generation to
which the assignment is made.
(18) \textbf{convert-assign-list}(genl,op) =
\begin{align*}
is-\leftrightarrow & (genl) \rightarrow \text{null} \\
T & \rightarrow \\
\text{convert-assign-list}(\text{tail}(genl),op) & \\
\text{convert-assign}(\text{head}(genl),op) \\
\end{align*}

for: \text{is-gg-list}(genl), where \text{is-gg} = \text{is-gen} \lor \text{is-ps-gen}

(19) \textbf{convert-assign}(gen,op) =
\begin{align*}
is-ps-gen & \rightarrow \\
pseudo-assign & (gen,op) \\
is-area\ast s-da & (gen) \land is-area\ast s-da(op) \rightarrow \\
\text{area-assign}(gen,op) \\
T & \rightarrow \\
\text{assign}(gen,op-1); & \\
op-1:\text{convert}(s-da(gen),op) \\
\end{align*}

for: \text{is-gg}(gen), where \text{is-gg} = \text{is-gen} \lor \text{is-ps-gen}

Ref.: \text{is-ps-gen} 7-11(26), \text{pseudo-assign} 7-12(29), \text{convert} 8-30(118), \\
\text{area-assign} 7-10(21)

(20) \textbf{assign}(gen,op) =
\begin{align*}
is-EVENT\ast s-da & (gen) \land \neg is-active(gen,PA) \land \\
s-compl\ast op-val & (op) = 1\text{-BIT} \rightarrow \\
s-s\ast s-el-ass & (s-pp(gen),s-\nu r(op),S) \\
s-pa\ast activate-tasks & (PA) \\
s-set\ast <gen>^{\wedge}ET \\
\neg is-active & (gen,PA) \rightarrow \\
s-s\ast s-el-ass & (s-pp(gen),s-\nu r(op),S) \\
T & \rightarrow \text{error} \\
\end{align*}

Ref.: \text{is-active} 4-7(11), \quad \text{el-ass} 2-22(70), \quad \text{activate-tasks} 4-18(44) \\
\text{op-val} 8-11(40)

Note: The \text{vr}-part of the operand is assigned to that part of \textit{S} selected by 
the pointer-part of the generation. 
For special properties of varying string assignment see the definition 
of the \text{vr-range} of a storage allocated for varying length strings 
(\text{chapter 6,1,4.1}).
For special properties of fixed length string assignment see the respective property of the storage mapping function (chapter 6.1.4.3.2). The assignment is illegal if the generation belongs to an active event variable.

If the generation is an event generation and it is set complete, waiting tasks are activated and the generation is appended to the event trace ET (cf. chapter 4).

7.1.5 Area assignment

The assignment of the contents of the area operand of the source area to the target area generation is preceded by a test. If any allocation in the source area has no corresponding space in the target area, the AREA condition is raised.

If the area assignment is possible then the target area is cleared and all allocations in the source area, identified by the offsets in the allocation state of the source area, are repeated in the target area using these offsets, and assignment is made from allocations of the source area to corresponding allocations in the target area.

\[
\text{area-assign}(gen, op) = \\
\text{no-area-cond}(s-pp(gen), alloc-state \cdot s-vr(op)) \quad \rightarrow \\
\text{area-assign-l}(gen, s-vr(op)); \\
\text{clear-area}(gen) \\
T \quad \rightarrow \quad \text{call-cond}(\text{AREA}, 503) \\
\text{for: is-area} \cdot s-da(gen), \text{is-area} \cdot s-da(cp) \]

Ref.: call-cond 9-11(18), alloc-state 2-23(73)

\[
\text{no-area-cond}(p, allst) = \\
\neg(\exists o) (o \in allst & \neg \text{is-area-applic}(o, size-l(p))) \\
\text{Ref.: is-area-applic 6-10(17), size-l 2-22(62)}
\]
7.1.6 Assignment to pseudo-variables

References to pseudo-variables in the left-part of an assignment statement are evaluated by the instruction $eval-ps-lp(t)$, where $t$ is the text of the reference. The evaluation results in a pseudo-generation. A pseudo-generation satisfies the predicate $is-ps-gen$:

(26) $is-ps-gen = (<s-id:is-id>,<s-arg-list:is-list>)$

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.

Metavariables

$ps-gen$  $is-ps-gen$  pseudo generation
(27)  \textbf{eval-ps-\textit{lp}}(t) =

\begin{align*}
\text{id}_t &= \text{mk-id(COMPLEX)} \vee \text{id}_t = \text{mk-id(CPLX)} \& \text{length}(s_1_t) = 2 \vee \\
\text{id}_t &= \text{mk-id(REAL)} \vee \text{id}_t = \text{mk-id(IMAG)} \\
\text{id}_t &= \text{mk-id(COMPLETION)} \vee \text{id}_t = \text{mk-id(STATUS)} \vee \\
\text{id}_t &= \text{mk-id(PRIORITY)} \vee \text{id}_t = \text{mk-id(UNSPEC))} \& \text{length}(s_1_t) = 1 \\
\text{mk-ps-gen}(\text{id}_t, \text{list}) ; \\
\{ \text{elem}(i)(\text{list}) : \text{eval-ref-gen}(\text{elem}(i, s_1_t), \emptyset) \ |
\text{id}_t = \text{mk-id(SUBSTR)} \& \text{(length}(s_1_t) = 2 \vee \text{length}(s_1_t) = 3) \\
\text{mk-ps-gen}(\text{id}_t, \text{list}) ; \\
\{ \text{elem}(1)(\text{list}) : \text{eval-ref-gen}(\text{head}(s_1_t), \emptyset), \\
\text{elem}(2)(\text{list}) : \text{pass}(\text{head}(s_1_t)) \cup \\
\text{elem}(i+1)(\text{list}) : \text{eval-intg-expr}(\text{elem}(i, s_1_t), \emptyset) \ |
\text{id}_t = \text{mk-id(ONSOURCE)} \vee \text{id}_t = \text{mk-id(ONCHAR)} \\
\text{id}_t = \text{mk-id(PRIORITY)} \rightarrow \\
\text{T} \rightarrow \text{error}
\end{align*}

\text{for: is-ref(}t,\text{), \ Ref.: eval-ref-gen 7-25(59), eval-intg-expr 7-21(52), mk-id 10-76(214)}

(28)  \textbf{mk-ps-gen}(id, list) =

\text{PASS: } \mu_0(\langle s-id:id_>, s-arg-list:<>)

(29)  \textbf{pseudo-assign}(ps-gen, op) =

\begin{align*}
\text{id}_p &= \text{mk-id(COMPLEX)} \vee \text{id}_p = \text{mk-id(CPLX)} \rightarrow \\
\text{assign-complex}(\text{gen}_1, \text{gen}_2, \text{op-1}); \\
\text{op-1: convert(} \mu(\text{AR-DA}; <s-mode:CPLX>), \text{op}) \\
\text{id}_p &= \text{mk-id(REAL)} \& \text{is-CPLX}\& s-mode=s-da(\text{gen}_1) \\
\text{assign-real}(\text{gen}_1, \text{op-1}); \\
\text{op-1: convert(} \mu(s-da(\text{gen}_1); <s-mode:REAL>), \text{op})
\end{align*}

\text{cont'd}
idp = mk-id(IMAG) & is-CPLX.s-mode.s-da(gen1) →
  assign-imag(gen1, op-1);
  op-1:convert(μ(s-da(gen1);<s-mode:REAL>), op)

idp = mk-id(ONSOURCE) & is-Q.s-onsource.s-cbif(CS) →
  convert-assign(head(s-onsource.s-cbif(CS)(AG)), op)

idp = mk-id(ONCHAR) & is-Q.s-onsource.s-cbif(CS) &
  is-Q.s-onchar.s-cbif(CS) →
  assign-onchar(head(s-onsource.s-cbif(CS)(AG)),
  s-onchar.s-cbif(CS), op-1);
  op-1:convert(μ₀(<s-base:CHAR>,<s-length:1>), op)

idp = mk-id(SUBSTR) & is-string-type.s-da(gen1) →
  assign-substr(gen1, elem(2,s-arg-list(ps-gen)),
  elem(3,s-arg-list(ps-gen)),
  elem(4,s-arg-list(ps-gen)), op)

idp = mk-id(COMPLETION) →
  assign-completion(gen1, op-1);
  op-1:convert(μ₀(<s-base:BIT>,<s-length:1>), op)

idp = mk-id(STATUS) →
  assign-status(gen1, op-1);
  op-1:convert(BINTG-DA, op)

idp = mk-id(PRIORITY) →
  assign-priority(gen1, op-1);
  op-1:convert(PRI-DA, op)

idp = mk-id(UNSPEC) & (is-num-type v is-string-type v is-area v is-PTP v
  is-OFFSET) (s-da(gen1)) →
  assign-unspec(gen1, op-1);
  op-1:convert(μ₀(<s-base:BIT>,
  <s-length:unspec-length(s-da(gen1))>), op)

T → error

where: idp = s-id(ps-gen)
  gen1 = elem(1,s-arg-list(ps-gen))
  gen2 = elem(2,s-arg-list(ps-gen))

cont'd
Ref.: AR-DA 8-13(56), convert 8-30(118), is-num-type 8-23(92),
is-string-type 8-25(93), BINTG-DA 8-12(53), PRI-DA 4-5(6),
convert-assign 7-9(19)

(30) assign-complex(gen-1, gen-2, op) =

\[
\text{convert-assign}(\text{gen-2, } \mu_o(<s-da:da_r>,<s-vr:s-real*vr(op)>));
\]

\[
\text{convert-assign}(\text{gen-1, } \mu_o(<s-da:da_r>,<s-vr:s-real*vr(op)>));
\]

where: \( da_r = \mu(s-da(op);<s-mode:REAL>) \)
Ref.: convert-assign 7-9(19)

(31) assign-real(gen, op) =

\[
\text{assign}(\text{gen, } \mu_o(<s-da:s-da(gen)>,<s-real*s-vr:s-vr(op)>,
<s-image:s-vr:s-image*vr:gen-op(gen,S)>))
\]

Ref.: assign 7-9(20), gen-op 7-19(47)

(32) assign-imag(gen, op) =

\[
\text{assign}(\text{gen, } \mu_o(<s-da:s-da(gen)>,<s-real*s-vr:s-real*vr:gen-op(gen,S)>,
<s-real*s-vr:s-vr*gen-op(gen,S)>)
\]

Ref.: assign 7-9(20), gen-op 7-19(47)

(33) assign-onchar(gen, i, op) =

\[
i > \text{length}s-vr*gen-op(gen,S) \rightarrow \text{error}
\]

\[
T \rightarrow \text{assign}(\text{gen, } \mu(gen-op(gen,S);\langle\text{elem}(i)*s-vr:s-vr(op)>)
\)

Ref.: assign 7-9(20), gen-op 7-19(47)

(34) assign-substr(gen, t, i, j, op) =

is-\( \Omega \) (j) & \( 1 \leq k \leq \text{gen} \)

\[
\text{assign-substr-l}(\text{gen, i, op-l});
\]

\[
\text{op-l: convert}(\mu (<s-base:s-base*s-da(gen)>),
<s-length:k_{gen - i + 1}>), \text{op})
\]

cont'd
\[ 0 \leq \Omega(j) \land 1 \leq k_{gen} \leq i + j - 1 \]

assign-substr-1\((gen, i, op-1)\);

\[ op-1: convert(\mu_{<s-base:s-base\cdot s-da(gen)>,<s-length:j>}, op) \]

\[ T \]

assign-substr-2\((gen-1, i, j, op)\);

\[ gen-1: eval-ref-gen(t, E); \]

\[ call-cond(STRG, 504) \]

where: \( k_{gen} = length \cdot op \cdot val \cdot gen \cdot op(val, \bar{S}) \)

Ref.: \ convert 8-30(118), \ eval-ref-gen 7-25(59), \ call-cond 9-11(18)

gen-op 7-19(47), \ op-val 8-11(40)

(35) \ assign-substr-1\((gen, i, op)\) =

\[ assign(gen, \mu_{gen-op(gen, \bar{S})}; \{elem(i + k - 1) . s-vr: elem(k, s-vr(op)) \mid 1 \leq s \leq length(op)) \}) \]

Ref.: \ assign 7-9(20), \ gen-op 7-19(47)

(36) \ assign-substr-2\((gen, i, j, op)\) =

\[ is-\Omega(j) \]

assign-substr-1\((gen, i_{m}, op-1)\);

\[ op-1: convert(\mu_{<s-base:s-base\cdot s-da(gen)>,<s-length:k_{gen} - i_{m} + 1>}, op) \]

\[ T \]

assign-substr-1\((gen, i_{m}, op-1)\);

\[ op-1: convert(\mu_{<s-base:s-base\cdot s-da(gen)>,<s-length:j_{m}>}, op) \]

where: \( k_{gen} = length \cdot op \cdot val \cdot gen \cdot op(val, \bar{S}) \)

\[ i_{m} = max(l, i) \]

\[ j_{m} = min(j, k_{gen} - i + 1) \]

Ref.: \ convert 8-30(118), \ gen-op 7-19(47), \ op-val 8-11(40)
(37) \[ \text{assign-completion}(\text{gen}, \text{op}) = \]
\[ \text{is-EVENT}\ast \text{s-da} (\text{gen}) \land \overline{\text{is-active}(\text{gen}, \text{PA})} \land \]
\[ \text{is-1-BIT}\ast \text{head}\ast \text{op-val}(\text{op}) \rightarrow \]
\[ \begin{align*}
& \text{s-s: el-ass}(\text{s-pp}(\text{gen}), \text{represent}(\text{EVENT}, v_{ev}), \mathbb{S}) \\
& \text{s-pa: activate-tasks}(\text{PA}) \\
& \text{s-et: }<\text{gen}> \wedge \text{ET} \\
& \text{is-EVENT}\ast \text{s-da} (\text{gen}) \land \overline{\text{is-active}(\text{gen}, \text{PA})} \rightarrow \\
& \text{s-s: el-ass}(\text{s-pp}(\text{gen}), \text{represent}(\text{EVENT}, v_{ev}), \mathbb{S}) \\
& T \rightarrow \text{error} \\
\end{align*} \]
\[ \text{where: } v_{ev} = \mu (\text{op-val} \ast \text{gen-op}(\text{gen}, \mathbb{S}); <\text{compl: head-op-val}(\text{op})> ) \]

Ref.: is-active 4-7(11), el-ass 2-22(70), represent 8-9(33), activate-tasks 4-18(44), op-val 8-11(40), gen-op 7-19(47)

Note: If the event variable is set complete waiting tasks are activated and the event generation is appended to the event trace ET.

(38) \[ \text{assign-status}(\text{gen}, \text{op}) = \]
\[ \text{is-EVENT}\ast \text{s-da} (\text{gen}) \land ( \overline{\text{is-active}(\text{gen}, \text{PA})} \lor \]
\[ ( \exists \text{ten})(\text{s-ev}\ast \text{s-te}\ast \text{ten}(\text{PA}) = \text{gen}) \rightarrow \]
\[ \begin{align*}
& \text{s-s: el-ass}(\text{s-pp}(\text{gen}), \text{represent}(\text{EVENT}, v_{ev}), \mathbb{S}) \\
& T \rightarrow \text{error} \\
\end{align*} \]
\[ \text{where: } v_{ev} = \mu (\text{op-val} \ast \text{gen-op}(\text{gen}, \mathbb{S}); <\text{status: op-val}(\text{op})> ) \]

Ref.: is-active 4-7(11), el-ass 2-22(70), represent 8-9(33), op-val 8-11(40), gen-op 7-19(47)

Note: Active task and event generations, other than the event generation being assigned to directly, are protected against being affected by the assignment.

(39) \[ \text{assign-priority}(\text{gen}, \text{op}) = \]
\[ \text{assign-priority-1}(\text{gen}, \text{op-1}); \]
\[ \text{op-1: eval-pri-2}(\text{op}) \]

Ref.: eval-pri-2 4-5(8), gen-op 7-19(47), op-val 8-11(40)
(40) \( \text{assign-priority}(\text{gen}, \text{op}) = \)

\[ \text{is-TASK} \cdot \text{s-da}(\text{gen}) \land (\neg \text{is-active}(\text{gen}, \text{PA}) \lor \) \]

\[ (\exists \text{ten})(\text{s-tv} \cdot \text{s-te} \cdot \text{ten}(\text{PA}) = \text{gen}) \rightarrow \]

\[ \text{s-s:el-ass}(\text{s-pp}(\text{gen}), \text{s-vr}(\text{op}), S) \]

\[ \text{is-} \Omega (\text{gen}) \rightarrow \]

\[ \text{s-s:el-ass}(\text{s-pp} \cdot \text{s-tv}(\text{TE}), \text{s-vr}(\text{op}), S) \]

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

Ref.: \text{is-active} 4-7(11), \text{el-ass} 2-22(70)

Note: Active task and event generations, other than the task generation being assigned to directly, are protected against being affected by the assignment.

(41) \( \text{assign-unspec}(\text{gen}, \text{op}) = \)

\[ \neg \text{is-active}(\text{gen}, \text{PA}) \rightarrow \]

\[ \text{s-s:el-ass}(\text{s-pp}(\text{gen}), \text{s-vr}(\text{op}), S) \]

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

Ref.: \text{el-ass} 2-22(70), \text{is-active} 4-7(11)

(42) \text{unspec-length}(\text{eda})

Note: This implementation-defined function gives the length of the bit-string which is to be assigned to a variable with data-attributes \text{eda}, via the UNSPEC pseudo-variable.

7.2 Evaluation of Expressions

This chapter defines the evaluation of scalar expressions, some special cases of expression evaluation needed in other parts of the definition, and the evaluation of the data attributes of an expression.

The evaluation of aggregate expressions in PL/I is reduced to the sequential evaluation of scalar expression. For this mechanism see the definition of the assignment statement (chapter 7.1), the definition for the evaluation of arguments (chapter 5.6.2), and the definition of the put statement (chapter 10.5.4).
7.2.1 Evaluation of Scalar Expressions

Scalar expressions are evaluated by the instruction `eval-expr(t,env)`, where \( t \) is the text of the expression and \( env \) is the environment in which the evaluation is to be performed. The instruction returns an operand.

The definition of data operations performed in expression evaluation is given in chapter 8, the definition of procedure calls, performed on evaluating functions, in chapter 5.

\[(43) \quad \text{eval-expr}(t,env) = \begin{cases} \text{is-infix-expr}(t) & \rightarrow \begin{pmatrix} \text{eval-infix-expr}(\text{op-1}, \text{op-2}, \text{infix-operator}(t)) \end{pmatrix} \\ \text{op-1} : \text{eval-expr}(\text{s-op-1}(t), env), & \text{op-2} : \text{eval-expr}(\text{s-op-2}(t), env) \end{cases} \\
\text{is-prefix-expr}(t) & \rightarrow \begin{pmatrix} \text{eval-prefix-expr}(\text{op}, \text{s-operator}(t)) \end{pmatrix} \\
\text{op} & : \text{eval-expr}(\text{s-op}(t), env) \\
\text{is-paren-expr}(t) & \rightarrow \begin{pmatrix} \text{eval-expr}(\text{s-op}(t), env) \end{pmatrix} \\
\text{is-ref}(t) & \rightarrow \begin{pmatrix} \text{eval-ref}(t, env) \end{pmatrix} \]

\[
\text{is-op}(t) \rightarrow \text{PASS}:t \\
\text{is-isub}(t) \rightarrow \text{PASS}:\text{isub-val}(t, env)
\]

for: \( \text{is-expr}(t) \)

Ref.: \( \text{eval-infix-expr} \ 8-14(65), \text{eval-prefix-expr} \ 8-27(108), \text{is-op} \ 8-11(38) \)

\[(44) \quad \text{infix-operator}(t) = \begin{cases} \text{is-intg-exponent}(t) & \rightarrow \text{C-EXP} \\ T & \rightarrow \text{s-operator}(t) \end{cases} \]

for: \( \text{is-infix-expr}(t) \)

Ref.: \( \text{is-intg-exponent} \ 8-26(104) \)
(45) \( \text{isub-val}(t, \text{env}) = \)
\[-\text{is-def} \circ \text{sub}(s-i(t))(\text{env}) \rightarrow \text{sub}(s-i(t))(\text{env}) \]
\( T \rightarrow \text{error} \)
for: \( \text{is-isub}(t) \)

(46) \( \text{eval-ref}(t, \text{env}) = \)
\[\text{is-var}(\text{attr}_t) \rightarrow \]
\[\text{pass-gen-op}(\text{gen}, \text{stg}) ; \]
\[\text{gen} : \text{eval-ref-gen}(t, \text{env}) \]
\[\text{is-entry}(\text{attr}_t) \rightarrow \]
\[\text{eval-function}(t, \text{env}) \]
\[\text{is-generic}(\text{attr}_t) \rightarrow \]
\[\text{eval-generic}(t, \text{env}) \]
\[\text{is-built-in}(\text{attr}_t) \rightarrow \]
\[\text{eval-built-in}(t, \text{env}) \]
\[(\text{is-LABEL}(\text{attr}_t) \vee \text{is-format-attr}(\text{attr}_t)) \& \text{is-simple-ref}(t) \rightarrow \]
\[\text{PASS}:\mu_0 (\langle \text{s-da}:\text{LABEL} >, \]
\[\langle \text{s-vr}:\text{represent} (\text{LABEL}, n_t (\text{DN})) >) \]
\( T \rightarrow \text{error} \)
for: \( \text{is-ref}(t) \)
Ref.: \( \text{eval-built-in} \ 11-4(4), \text{represent} \ 8-9(33), \text{eval-ref-gen} \ 7-25(59) \)
\( \text{is-built-in} \ 2-18(40) \)

(47) \( \text{gen-op}(\text{gen}, \text{stg}) = \)
\[\text{is-scalar}\circ \text{s-da}(\text{gen}) \rightarrow \]
\[\mu_0 (\langle \text{s-da}:\text{s-da}(\text{gen}) >, \langle \text{s-vr}:\text{el-rf}(s-\text{pp}(\text{gen}) (\text{stg}) >) \]
\( T \rightarrow \text{error} \)
Ref.: \( \text{el-rf} \ 2-22(68) \)
(48) `eval-function(t, env) =`

    return-free(b);
    int-call-l(n_t, s_l_t, env, gen, id_t, \emptyset);
    gen: allocate-l(b, s-ret-type*n_t(DN), PACKED);
    b: un-name

for: is-entry(attr_t)

Ref.: `int-call-l` 5-33(52), `allocate-l` 5-44(80), `un-name` 2-17(36)

Note: A dummy variable is allocated for keeping the operand returned by the function. It is freed after return, in case the function is not terminated by a return statement it is freed at task end.

(49) `eval-generic(t, env) =`

    return-free(b);
    int-call-l(n, s_l_t, env, gen, id_t, \emptyset);
    gen: allocate-l(b, ert, PACKED);
    b: un-name,
    ert: ret-type(n);
    n: generic-sel(id_t, s_l_t, env)

for: is-generic(attr_t)

Ref.: `int-call-l` 5-33(52), `allocate-l` 5-44(80), `un-name` 2-17(36), `generic-sel` 5-54(99)

(50) `return-free(b) =`

    PASS: gen-op(head*b(A), S)
    s-s: el-free(s-pp+head*b(A), I, S)
    s-ag: \delta(A; b)

Ref.: el-free 2-23(76)

(51) `ret-type(n) =`

    PASS: s-ret-type*n(DN)
7.2.2 Special Cases of Expression Evaluation

7.2.2.1 Evaluation of integer values

(52) \[ \text{eval-intg-expr}(\text{expr}, \text{env}) = \]
\[ \text{is-*}(\text{expr}) \lor \text{is-\bigcirc}(\text{expr}) \rightarrow \text{PASS:}\text{expr} \]
\[ \text{is-expr}(\text{expr}) \rightarrow \]
\[ \text{pass-op-val}(\text{op}); \]
\[ \text{op:convert}(\text{BINTG-DA}, \text{op-1}); \]
\[ \text{op-1:eval-expr}(\text{expr}, \text{env}) \]
\[ \text{for: is-expr}(\text{expr}) \lor \text{is-*}(\text{expr}) \lor \text{is-\bigcirc}(\text{expr}) \]
Ref.: \text{op-val} 8-11(40), \text{convert} 8-30(118), \text{BINTG-DA} 8-12(53),
\text{eval-expr} 7-18(43)

7.2.2.2 Evaluation of optional expressions

(53) \[ \text{eval-opt-expr}(\text{expr}, \text{env}) = \]
\[ \text{is-*}(\text{expr}) \lor \text{is-\bigcirc}(\text{expr}) \rightarrow \text{PASS:}\text{expr} \]
\[ \text{is-expr}(\text{expr}) \rightarrow \]
\[ \text{eval-expr}(\text{expr}, \text{env}) \]
\[ \text{for: is-expr}(\text{expr}) \lor \text{is-*}(\text{expr}) \lor \text{is-\bigcirc}(\text{expr}) \]
Ref.: \text{eval-expr} 7-18(43)

7.2.3 Evaluation of the Data Attributes of Expressions

Given an expression it is possible to evaluate the data attributes of the aggregate returned by the expression without actually doing the expression evaluation. The data attributes of the expression \text{expr} in the environment \text{env} are returned by the instruction \text{eda-expr}(t,\text{env}). It is needed for doing the expansion of assignment statements, for determining the attributes of a dummy variable to be allocated on argument passing, and for determining the attributes of arguments governing generic selection. The data attributes of the results of infix and prefix operations are defined in chapter 8.
(54) \(eda\text{-}expr(t,env) = \)

\[
\text{is-intg-exponent}(t) \rightarrow \\
\text{pass}\text{-}eda\text{-intg-exponent}(eda,s\text{-}op\text{-}2(t)); \\
eda\text{: }eda\text{-}expr(s\text{-}op\text{-}1(t),env)
\]

\[
\text{is-infix-expr}(t) \rightarrow \\
\text{pass}\text{-}eda\text{-infix-expr}(eda\text{-}1,eda\text{-}2,s\text{-}operator(t)); \\
eda\text{-}1:eda\text{-}expr(s\text{-}op\text{-}1(t),env), \\
eda\text{-}2:eda\text{-}expr(s\text{-}op\text{-}2(t),env)
\]

\[
\text{is-prefix-expr}(t) \rightarrow \\
\text{pass}\text{-}eda\text{-prefix-expr}(eda,s\text{-}operator(t)); \\
eda\text{: }eda\text{-}expr(s\text{-}op(t),env)
\]

\[
\text{is-paren-expr}(t) \rightarrow \\
eda\text{-}expr(s\text{-}op(t),env)
\]

\[
\text{is-ref}(t) \rightarrow \\
edaref(t,env)
\]

\[
is-op(t) \rightarrow \\
\text{PASS: }s\text{-}da(t)
\]

\[
is\text{-isub}(t) \rightarrow \text{error}
\]

\text{for : } is\text{-}expr(t) \\
\text{Ref.: } is\text{-}intg\text{-}exponent \ 8\text{-}26(104), \ is\text{-}on \ 8\text{-}11(38)

(55) \(eda\text{-intg-exponent}(eda,op) = \)

\[
is\text{-array}(eda) \rightarrow \\
(eda;\langle s\text{-}elem:eda\text{-intg-exponent}(s\text{-}elem(eda),op)\rangle)
\]

\[
is\text{-struct}(eda) \rightarrow \\
\mu_o\{\langle s\text{-}da\text{elem}(i):eda\text{-intg-exponent}(s\text{-}da\text{elem}(i,eda),op) \mid \\
\text{isorder}(eda)\}\}
\]

\[
is\text{-scalar}(eda) \rightarrow \\
edain\text{-intg}\text{-exp}(eda,op)
\]

\[
is\text{-cell}(eda) \rightarrow \text{error}
\]

\text{for : } is\text{-intg}\text{-op}(op) \\
\text{Ref.: } edain\text{-intg\text{-exp} \ 8\text{-}26(105)}
(56) \( \text{eda-infix-expr}(\text{eda-1}, \text{eda-2}, \text{opor}) = \)

\[
\begin{align*}
is\text{-array}(\text{eda-1}) & \& is\text{-array}(\text{eda-2}) & \& (\text{dim}(\text{eda-1}) = \text{dim}(\text{eda-2})) & \& (s-lbd(\text{eda-1}) = s-lbd(\text{eda-2})) & \& (s-ubd(\text{eda-1}) = s-ubd(\text{eda-2})) \\
& \rightarrow \\
& \mu(\text{eda-1}; <s\text{-elem}; \text{eda-infix-expr}(s\text{-elem}(\text{eda-1}), s\text{-elem}(\text{eda-2}), \text{opor}>) ) \\
is\text{-array}(\text{eda-1}) & \& \neg is\text{-array}(\text{eda-2}) \\
& \rightarrow \\
& \mu(\text{eda-1}; <s\text{-elem}; \text{eda-infix-expr}(s\text{-elem}(\text{eda-1}), \text{eda-2}, \text{opor}>) ) \\
\neg is\text{-array}(\text{eda-1}) & \& is\text{-array}(\text{eda-2}) \\
& \rightarrow \\
& \mu(\text{eda-2}; <s\text{-elem}; \text{eda-infix-expr}(\text{eda-1}, s\text{-elem}(\text{eda-2}), \text{opor}>) ) \\
is\text{-struct}(\text{eda-1}) & \& is\text{-struct}(\text{eda-2}) & \& (\text{order}(\text{eda-1}) = \text{order}(\text{eda-2})) \\
& \rightarrow \\
& \mu_0(\{<s\text{-da}\text{elem}(i); \text{eda-infix-expr}(s\text{-da}\text{elem}(i, \text{eda-1}), \text{s}\text{-da}\text{elem}(i, \text{eda-2}), \text{opor}>) \mid 1 \text{isorder}(\text{eda-1})\}) \\
is\text{-struct}(\text{eda-1}) & \& is\text{-scalar}(\text{eda-2}) \\
& \rightarrow \\
& \mu_0(\{<s\text{-da}\text{elem}(i); \text{eda-infix-expr}(s\text{-da}\text{elem}(i, \text{eda-1}), \text{eda-2}, \text{opor}>) \mid 1 \text{isorder}(\text{eda-1})\}) \\
is\text{-scalar}(\text{eda-1}) & \& is\text{-struct}(\text{eda-2}) \\
& \rightarrow \\
& \mu_0(\{<s\text{-da}\text{elem}(i); \text{eda-infix-expr}(\text{eda-1}, s\text{-da}\text{elem}(i, \text{eda-2}), \text{opor}>) \mid 1 \text{isorder}(\text{eda-2})\}) \\
is\text{-scalar}(\text{eda-1}) & \& is\text{-scalar}(\text{eda-2}) \\
& \rightarrow \\
& \text{eda-infix}(\text{eda-1}, \text{eda-2}, \text{opor}) \\
T & \rightarrow \text{error} \\
\text{Ref.}: \text{eda-infix } 8-22(88)
\]

(57) \( \text{eda-prefix-expr}(\text{eda}, \text{opor}) = \)

\[
\begin{align*}
is\text{-array}(\text{eda}) & \rightarrow \\
& \mu(\text{eda}; <s\text{-elem}; \text{eda-prefix-expr}(s\text{-elem}(\text{eda}), \text{opor}>) ) \\
is\text{-struct}(\text{eda}) & \rightarrow \\
& \mu_0(\{<s\text{-da}\text{elem}(i); \text{eda-prefix-expr}(s\text{-da}\text{elem}(i, \text{eda}), \text{opor}>) \mid 1 \text{isorder}(\text{eda})\}) \\
is\text{-scalar}(\text{eda}) & \rightarrow \\
& \text{eda-prefix}(\text{eda}, \text{opor}) \\
is\text{cell}(\text{eda}) & \rightarrow \text{error} \\
\text{Ref.}: \text{eda-prefix } 8-29(16)
\]
(58) \[ \text{eda-ref}(t, \text{env}) = \]
\[
\begin{align*}
&\text{(is-prop-var}(\text{attr}_t) \& \text{is-gen-head}(n_t(N)}(\text{AG})) \lor \text{is-defined}(\text{attr}_t) \lor \\
&\text{is-based}(\text{attr}_t) \rightarrow \\
&\text{PASS;edar}_t \\
&\text{is-entry}(\text{attr}_t) \lor \text{is-param-builtin}(\text{attr}_t) \rightarrow \\
&\text{ret-type}(n_t) \\
&\text{is-generic}(\text{attr}_t) \rightarrow \\
&\text{ret-type}(n); \\
&n:\text{generic-sel}(\text{id}_t, \text{sl}_t, \text{env}) \\
&\text{is-BUILTIN}(\text{attr}_t) \rightarrow \\
&\text{eda-builtin}(t, \text{env}) \\
&\text{is-LABEL}(\text{attr}_t) \lor \text{is-format-attr}(\text{attr}_t) \rightarrow \text{PASS;LABEL} \\
\end{align*}
\]

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

do for: is-ref(t)

Ref.: \text{ret-type 7-20}(51), \quad \text{generic-sel 5-54}(99), \quad \text{eda-builtin 11-8}(16)
\quad \text{is-gen 2-27}(96), \quad \text{is-param-builtin 2-19}(41)

## 7.3 Evaluation of References to Variables

The generation referenced by a reference \(t\) to a variable is evaluated in the environment \(\text{env}\) by the instruction \(\text{eval-ref-gen}(t, \text{env})\). The variable may be a proper variable of any storage class, a defined variable, or a based variable.

**Abbreviation**

\[
\text{attr}_b = \text{s-attr}(\text{sel}\ast\text{id-list}\ast\text{s-base}(\text{attr}_t)(\text{env}_t)(\text{AT}))
\]

(the attributes of the base reference of a defined variable)
(59) \text{eval-ref-gen}(t, \text{env}) = \\
\text{is-ref}(t) \& \text{length}(s_{lt}) = 0 \lor (\text{length}(s_{lt}) \geq \text{ref-dim}(d_{at}, t_{eq})) \rightarrow \\
\text{eval-sub-gen}(t, r_{l}, \text{env}); \\
r_{l} : \text{mk-ref}(d_{at}, s_{lt}, t_{eq}, \text{env}) \\
T \rightarrow \text{error} \\
\text{for: is-ref}(t), \text{is-var}(\text{attr}_t) \\
\text{Ref.: ref-dim 7-27(65)}

(60) \text{eval-sub-gen}(t, r_{l}, \text{env}) = \\
\neg \text{is-isub-def}(\text{attr}_t) \rightarrow \\
\text{pass-sub-gen}(\text{gen}, r_{l}); \\
\text{gen} : \text{eval-gen}(t, \text{env}) \\
\text{is-isub-def}(\text{attr}_t) \& \text{is-non-varying}(d_{at}) \& \neg \text{is-based}(\text{attr}_b) \& \\
\neg \text{is-defined}(\text{attr}_b) \rightarrow \\
\text{pass-sub-gen}(\text{gen}-1, \text{rest-list}(r_{l}, \text{dim}(d_{at}))); \\
\text{gen}-1 : \text{mk-base-gen}(\text{gen}-2, n_{t}(\text{DN}), d_{st}); \\
\text{gen}-2 : \text{eval-ref-gen}(s_{base}(\text{attr}_t), \text{isub-env}(e_{nt}, r_{l}, \text{dim}(d_{at}))) \\
T \rightarrow \text{error} \\
\text{for: is-ref}(t), \text{is-var}(\text{attr}_t) \\
\text{Ref.: sub-gen 7-32(84), rest-list 7-28(72), mk-base-gen 7-28(70), isub-env 7-28(69), is-non-varying 7-29(75)}

(61) \text{eval-gen}(t, \text{env}) = \\
\text{is-prop-var}(\text{attr}_t) \rightarrow \\
\text{eval-prop-gen}(t, \text{env}) \\
\text{is-defined}(\text{attr}_t) \& \text{is-non-varying}(d_{at}) \& \neg \text{is-based}(\text{attr}_b) \& \\
\neg \text{is-defined}(\text{attr}_b) \rightarrow \\
\text{eval-def-gen}(\text{gen}, n_{t}(\text{LN}), d_{st}, s_{pos}(\text{attr}_t)); \\
\text{gen} : \text{eval-ref-gen}(s_{base}(\text{attr}_t), e_{nt}) \\
\text{is-based}(\text{attr}_t) \& \text{is-non-varying}(d_{at}) \rightarrow \\
\text{eval-based-gen}(o_{p}, n_{t}(\text{LN}), d_{st}); \\
o_{p} : \text{eval-ptr-ref}(t, \text{env}) \\
T \rightarrow \text{error} \\
\text{cont'd}
7.3.1 Evaluation of subscripts and name qualification

The instruction make-rl returns a list of integer values and *'s, representing evaluated subscripts and name qualifications. The correct length of the subscript list is tested in mk-rl, together with the test made in eval-ref-gen with the help of the function ref-dim. If subscript values exceed array bounds, the SUBSCRIPT RANGE condition is called.

(62) $\text{mk-rl}(\text{eda}, \text{sl}, \text{eql}, \text{env}) =$

\[
\begin{align*}
\text{is-<> (eql) & (is-<> (sl) \lor (is-*\text{-list}(sl) \& \text{length}(sl) = \text{dim}(\text{eda})))} \\
\text{PASS:<>}
\end{align*}
\]

\[
\begin{align*}
\text{is-<> (sl)} \\
\text{PASS:mk-rl-1(eda,eql)}
\end{align*}
\]

\[
\begin{align*}
\text{is-struct(eda) \lor is-cell(eda)}
\end{align*}
\]

\[
\begin{align*}
\text{mk-list(head(eql),list);}
\text{list:mk-rl(s-da\text{-elem}(head(eql),eda),sl,tail(eql),env)}
\end{align*}
\]

\[
\begin{align*}
\text{is-array(eda) \& \text{length}(sl)\geq \text{dim}(eda)}
\end{align*}
\]

\[
\begin{align*}
\text{mk-list(i-1,list);}
\text{list:mk-rl(s-elem(eda),tail(sl),eql,env);}
\text{i-1: test-subscript(s-1b3(eda),s-ubd(eda),i-2);}
\text{i-2:eval-intg-expr(head(sl),env)}
\end{align*}
\]

T $\rightarrow$ error

Ref.: eval-intg-expr 7-21(52)

(63) $\text{mk-rl-1(eda,eql) =}$

\[
\begin{align*}
\text{is-<> (eql) \rightarrow <>}
\end{align*}
\]

\[
\begin{align*}
\text{is-struct(eda) \lor is-cell(eda) \rightarrow}
\end{align*}
\]

\[
\begin{align*}
\text{<head(eql)> \&} \text{mk-rl-1(s-da\text{-elem}(head(eql),eda),tail(eql))}
\end{align*}
\]

cont'd
is-array(eda) →
<*> mk-rl-1(s-element(eda),eq1)
T → error

(64) test-subscript(k-1,k-2,i) =

is-->(i) ∨ (k-1≤i≤k-2) → PASS:i
T →
    error;
    call-cond(SUBRG,505)
Ref.: call-cond 9-11(18)

(65) ref-dim(da,eq1) =

is<>(eq1) → 0
is-array(da) → 1+ref-dim(s-element(da),eq1)
is-struct(da) ∨ is-cell(da) →

ref-dim(s-da-element(head(eq1),da),tail(eq1))
T → error

(66) eval-ql(ql,da) =

is<>(ql) → <>
is-struct(da) ∨ is-cell(da) →

<ql,da> eval-ql(tail(ql),s-da-element(ql,da,da))
T → error

where: i_{ql,da} = (li)(s-id-element(i,da) = head(ql))

7.3.2 Reference to proper variables

(67) eval-prop-gen(t,env) =

is-gen-head(n_t(DN)(AG)) →
    PASS:head(n_t(DN)(AG))
T → error

Ref.: is-gen 2-27(96)
7.3.3 Reference to Defined Variables

7.3.3.1 Isub-defined variables

See the definition of the instruction eval-sub-gen. The number of dimensions of the defined array variable determines the number of isub-variables to be created. The names sub(i) of these variables are associated in the environment directly with their integer values, determined by evaluated subscripts of the reference to the defined variable. Subscript expressions of the base array are evaluated in the environment containing the isubs. The resulting base generation is treated as in the correspondence-defined case. That part of the reference list not used to give values of isubs (determined by the function rest-list) is used to determine the appropriate sub-generation of the base generation.

(68) \text{is-isub-def(attr) =}
\text{is-defined(attr) \& (\exists \gamma)(is-isub\gamma:s\cdot base(attr)) \& is-\Omega\cdot s\cdot pos(attr)}

(69) \text{isub-env(env,rl,k) =}
\mu(env;\{<\text{sub}(i):\text{elem}(i,rl) > | 1 \leq \text{k}))

(70) \text{mk-base-gen(gen,eda,dens) =}
\text{is-corresp(s-da(gen),base-elem(eda)) \& dens-match(s-mi(gen),dens) \rightarrow}
\text{PASS; } \mu_\Omega(<s-da;base-elem(eda) >, <s-mi:s-mi(gen) , <s-pp:s-pp(gen) >)
\text{T \rightarrow error}

Ref.: dens-match 5-38(63), is-corresp 7-31(81)

(71) \text{base-elem(da) =}
\text{is-array(da) \rightarrow base-elem \cdot s \cdot elem(da)}
\text{T \rightarrow da}

(72) \text{rest-list(rl,k) = } \mu_\Omega(\{<\text{elem}(i):\text{elem}(i+k,rl) > | 1 \leq \text{length(rl)-k})

7.3.3.1
7.3.3.2 Correspondence and overlay defined variables

The discrimination between the two cases of defining is made by the predicates is-overlay and is-corresp. In the overlay-case, a new generation is formed from the base generation by collecting the elements of the pointer-part to a new single pointer, and installing the data-attributes of the defined variable in the generation. In the correspondence-case, the pointer-part and the mapping information of the base generation is retained, and only the da-part is replaced by the data-attributes of the defined variable. The data-attributes in the da-part and those in the mapping information of the resulting generation then may differ in array bounds and string lengths.

(73) \[
\text{eval-def-gen}(\text{gen}, \text{eda}, \text{dens}, i) =
\]
\[
\text{is-overlay}(\text{gen}, \text{eda}, \text{dens}, i) \rightarrow
\]
\[
\text{PASS}: \mu_0(\langle s\text{-da}: \text{eda} \rangle, \langle s\text{-mi}: \mu_0(\langle s\text{-da}: \text{eda} \rangle, \langle s\text{-dens}: \text{PACKED} \rangle) \rangle, \langle s\text{-pp}: \text{collect-p}(s\text{-da}(\text{gen}), \text{pos}(i), \text{string-extent} (\text{eda})) \rangle)
\]
\[
\text{is-corresp}(s\text{-da}(\text{gen}), \text{eda}) \land \text{dens-match}(s\text{-mi}(\text{gen}), \text{dens}) \rightarrow
\]
\[
\text{PASS}: \mu_0(\langle s\text{-da}: \text{eda} \rangle, \langle s\text{-mi}: s\text{-mi}(\text{gen}) \rangle, \langle s\text{-pp}: s\text{-pp}(\text{gen}) \rangle)
\]
T \rightarrow \text{error}

Ref.: dens-match 5-38(63)

(74) \[
\text{is-overlay}(\text{gen}, \text{eda}, \text{dens}, i) =
\]
\[
\text{is-PACKED}(s\text{-dens} \land s\text{-mi}(\text{gen}) \land \text{is-PACKED}(\text{dens}) \land 
\text{is-connected(}\text{gen}) \land \text{is-non-varying}(s\text{-da}(\text{gen}) \land 
(\text{is-bit-aggr}(s\text{-da}(\text{gen}) \land \text{is-bit-aggr}(\text{eda}) \lor 
\text{is-char-aggr}(s\text{-da}(\text{gen}) \land \text{is-char-aggr}(\text{eda})) \land 
((\text{string-extent}(s\text{-da}(\text{gen}) \land \text{pos}(i)) \land (\text{string-extent}(\text{eda}) \land 1)) \land 
(\neg \text{is-} Q(i) \lor \neg \text{is-corresp}(s\text{-da}(\text{gen}), \text{eda}))
\]

(75) \[
\text{is-non-varying}(\text{da}) =
\]
\[
\neg (\exists x) (\neg \text{is-} Q \land \text{is-varying}(x)(\text{da}))
\]
(76) \( \text{is-bit-aggr}(da) = \)

\( \text{is-string}(da) \land \text{is-BIT-s-base}(da) \lor \text{is-bin-pic}(da) \quad \rightarrow \quad T \)

\( \text{is-scalar}(da) \quad \rightarrow \quad F \)

\( \frac{\text{ubd}(da)}{i} \quad \text{is-bit-aggr}\cdot \text{da-part}(da,i) \)

(77) \( \text{is-char-aggr}(da) = \)

\( \text{is-string}(da) \land \text{is-CHAR-s-base}(da) \lor \text{is-pic}(da) \land \neg \text{is-bin-pic}(da) \quad \rightarrow \quad T \)

\( \text{is-scalar}(da) \quad \rightarrow \quad F \)

\( \frac{\text{ubd}(da)}{i} \quad \text{is-char-aggr}\cdot \text{da-part}(da,i) \)

(78) \( \text{string-extent}(eda) = \)

\( \text{is-array}(eda) \quad \rightarrow \quad (s\text{-ubd}(eda) - s\text{-lbd}(eda) + 1) \cdot \text{string-extent}(s\text{-elem}(eda)) \)

\( \text{is-struct}(eda) \quad \rightarrow \quad \sum_{i=1}^{\text{order}(eda)} \text{string-extent}(s\text{-da}\cdot \text{elem}(i,eda)) \)

\( \text{is-string}(eda) \lor \text{is-pic}(eda) \quad \rightarrow \quad \text{str-length}(eda) \)

Ref.: str-length 8-40(157)

(79) \( \text{pos}(i) = \)

\( \text{is}\cdot \mathcal{Q}(i) \quad \rightarrow \quad 1 \)

\( \quad T \quad \rightarrow \quad i \)

(80) \( \text{collect-p}(eda,i-1,i-2) = \)

\( \text{string-refer}(eda_1,<2>) \)

where: (is-bit-aggr(eda) \supset \text{is-bit-aggr}(eda_1)) \land

(is-char-aggr(eda) \supset \text{is-char-aggr}(eda_1)) \land

\( \text{string-extent}\cdot \text{da-part}(eda_1,1) = i-1 - 1 \land \text{string-extent}\cdot \text{da-part}(eda_1,2) = i-2 \)

Ref.: string-refer 6-14(27)

Note: The properties stated for eda are sufficient to guarantee a unique result. See the properties of the storage mapping function for packed strings in chapter 6.1.4.3.1.
7.3.4 Reference to based variables

A new generation is formed by using the attributes of the based variable and the pointer-value resulting from evaluation by the instruction \texttt{eval-ptr-ref}.

\begin{align*}
\text{(82) } & \texttt{eval-ptr-ref}(t, \text{env}) = \\
& \text{is-ptr-qual-ref}(t) \\
& \text{eval-ref-gen}(s-\text{ptr-qual}(t), \text{env}) \\
& \overline{\text{is-}Q\text{-}s-\text{ptr}(\text{attr})} \\
& \text{eval-ref-gen}(s-\text{ptr}(\text{attr}), \text{env})
\end{align*}

Ref.: \texttt{eval-ref-gen} 7-25(59)

\begin{align*}
\text{(83) } & \texttt{eval-based-gen}(\text{op}, \text{eda}, \text{dens}) = \\
& \text{fs-PTR} \cdot \text{da}(\text{op}) \\
& \text{PASS}: \mu(\textit{a}, \textit{b}, < s-\text{da}: \text{eda}, < s-\text{mi}: \mu(\textit{a}, < s-\text{da}: \text{eda}, < s-\text{dens}: \text{dens} >), < s-pp: \text{op-val}(\text{op}) >)
\end{align*}

Ref.: \texttt{op-val} 8-11(40)

Note: The resulting generation in general allows a well-defined reference to storage only when the part of storage identified by the pointer has
been allocated with attributes matching those of the based variable. Exceptions to this rule are given by special properties of the storage mapping function. For left-to-right equivalence see chapter 6.1.4.3.3.

### 7.3.5 Subgenerations of generations

A reference list \( rl \) determines a sub-generation of the generation \( gen \) of a variable, given by the function \( \text{sub-gen}(gen, rl) \). 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 the variable.

(84) \[
\text{sub-gen}(gen, rl) =
\]

\[
\begin{align*}
\mu_0 & (\{ s \text{-da} : \text{sub-edia}(s \text{-da}(gen), rl) \}, \\
& \{ s \text{-mi} : \mu_0 ( s \text{-da} : \text{sub-edia}(s \text{-da} s \text{-mi}(gen), rl), \\
& \{ s \text{-dens} : s \text{-dens} s \text{-mi}(gen) \}, \\
& \{ s \text{-pp} : \text{refer}(s \text{-da} s \text{-mi}(gen), s \text{-dens} s \text{-mi}(gen), s \text{-pp}(gen), rl) \})
\end{align*}
\]

(85) \[
\text{refer}(eda, dens, pp, rl) =
\]

\[
\begin{align*}
is \Rightarrow (rl) & \rightarrow pp \\
is \Rightarrow \text{ptr-val}(pp) & \rightarrow \text{conn-refer}(eda, dens, pp, rl) \\
is \Rightarrow \text{**}(head(rl)) & \& \text{is-array}(eda) & \rightarrow
\]

\[
\begin{align*}
\text{is-array}(eda) & \rightarrow \\
\text{length}(pp) \rightarrow [\text{refer}(s \text{-elem}(eda), dens, elem(i, pp), tail(rl))]_{i=1}^{\text{list}(pp)} \\
is \Rightarrow \text{array}(eda) & \rightarrow \\
\text{refer}(s \text{-elem}(eda), dens, elem(head(rl) - s \text{-lad}(eda) + 1, pp), tail(rl))
\end{align*}
\]

Ref.: \text{is-\text{ptr-val} 2-21}(52)

Note: The function \( \text{refer} \) returns a new pointer part. If \( pp \) is a list of pointers, i.e. if the generation was not connected, then \( eda \) must be an array.
(86) \( \text{conn-refer}(\text{eda}, \text{dens}, p, rl) = \)

\[
\begin{align*}
\text{is-\textless\\rangle}(rl) & \quad \rightarrow \quad p \\
\text{is-\ast\textbullet head}(rl) & \quad \& \quad \text{is-array}(\text{eda}) \quad \rightarrow \\
\mu(\text{eda}; \langle s\text{-elem}; \text{sub-eda}(s\text{-elem}(\text{eda}), \text{tail}(rl)) \rangle) \\
\text{is-\textless \textbullet struct}(\text{eda}) & \quad \vee \quad \text{is-cell}(\text{eda}) \quad \vee \quad \text{is-array}(\text{eda}) \quad \rightarrow \\
\text{conn-refer}(\text{da-part}(\text{eda}, \text{head}(rl)), \text{dens}, \\
\quad (\text{map}(\text{eda}, \text{dens}, \text{head}(rl))) \ast p, \text{tail}(rl))
\end{align*}
\]

Ref.: map 6-10(18)

Note: If the head of \( rl \) is \( \ast \) and \( \text{eda} \) is an array, then the result is a list of pointers and the corresponding generation will be non-connected.

(87) \( \text{is-connected}(\text{gen}) = \text{is-\textbullet ptr-val}\ast \text{-pp}(\text{gen}) \ast \text{s-da}(\text{gen}) = \text{s-da} \ast \text{s-mi}(\text{gen}) \)

Ref.: is-\textbullet ptr-val 2-21(52)

Note: The generation of a variable is connected if all storage belonging to the variable, and only this storage, is identified by a single pointer. On allocation of a variable a connected generation is created. Non-connected generations may result, first, from taking the cross-section of an array or a substructure of an array of structures, second, from correspondence defined variables. In the first case, storage is identified by a list of pointers, in the second case, the data attributes of the generation and the data attributes which are argument to the storage mapping function may differ in array bounds and string lengths (see chapter 7.3.5).

(88) \( \text{sub-eda}(\text{eda}, rl) = \)

\[
\begin{align*}
\text{is-\textless\\rangle}(rl) & \quad \rightarrow \quad \text{eda} \\
\text{is-\ast\textbullet head}(rl) & \quad \& \quad \text{is-array}(\text{eda}) \quad \rightarrow \\
\mu(\text{eda}; \langle s\text{-elem}; \text{sub-eda}(s\text{-elem}(\text{eda}), \text{tail}(rl)) \rangle) \\
\text{is-\textless \textbullet struct}(\text{eda}) & \quad \vee \quad \text{is-cell}(\text{eda}) \quad \vee \quad \text{is-array}(\text{eda}) \quad \rightarrow \\
\text{sub-eda}(\text{da-part}(\text{eda}, \text{head}(rl)), \text{tail}(rl))
\end{align*}
\]
(89) \text{sub-da}(\text{da}, \text{sl}, \text{eql}) =
\begin{align*}
\text{is-}\langle\text{eql}\rangle & \land (\text{is-}\langle\text{sl}\rangle \lor \text{is-}\langle\text{list}\rangle) \rightarrow \text{da} \\
\text{is-array}(\text{da}) & \land \text{is-}\langle\text{sl}\rangle \rightarrow \\
\mu_b(\langle\text{s-lbd}\rangle, \langle\text{s-ubd}\rangle, \langle\text{s-elem}\rangle, \text{sub-da}(\text{s-elem}(\text{da}), \langle\rangle, \text{eql})), \\
\text{is-array}(\text{da}) & \land \neg\text{is-}\langle\text{sl}\rangle \rightarrow \\
\text{sub-da}(\text{s-elem}(\text{da}), \text{tail}(\text{sl}), \text{eql}) \\
(\text{is-struct}(\text{da}) \lor \text{is-cell}(\text{da})) & \land \neg\text{is-}\langle\text{eql}\rangle \rightarrow \\
\text{sub-da}(\text{s-da-elem}(\text{head}(\text{eql}), \text{da}), \text{sl}, \text{tail}(\text{eql}))
\end{align*}

\textbf{T} \rightarrow \text{error}

\textbf{Note:} This function is used to determine the data attributes of a reference, given the subscript list, the evaluated name qualification list, and the data attributes of the referenced variable. The function is not specific to the present chapter.
3. DATA, OPERATIONS AND CONVERSIONS

This chapter describes operands and the operations that can be applied to them. Values, value representations, and the functions value and represent for the transition between them are defined in §8.1, the evaluation of prefix expressions in §8.3, the evaluation of infix expressions in §8.4, the convert-instruction in §8.5. In §8.2, some auxiliary definitions on data attributes are given, and in §8.6, all definitions necessary for the interpretation of pictures are collected.

Metavariables

| da, da-tg | is-val-da | evaluated scalar data attribute that can be associated with a value (cf. §8.1), or area attribute |
| v, v-re, v-im | is-value |
| vr | is-vr | value representation |
| op | is-op | operand |
| opor | is-operator | operator |
| base | is-CHAR ∨ is-BIT | base of a string |
| n, m, i, j | is-intg-val | integer value |
| % | is-arithm(da) | arbitrary selector function |

Abbreviations

For is-arithm(da):

\[
\begin{align*}
P_{da} &= \text{s-prec}(da) & \text{precision of } da \\
q_{da} &= \text{s-scale-f}(da) & \text{scale-factor of } da \\
\text{base}_{da} &= (\text{is-DEC}\cdot\text{s-base}(da) \rightarrow 10, \\
& & \text{is-BIN}\cdot\text{s-base}(da) \rightarrow 2) & \text{number base of } da \\
N_{da} &= \text{max-prec}(da) & \text{maximum precision associated with } da
\end{align*}
\]

For is-arithm(da) & is-FLT.s-scale(da):

\[
\begin{align*}
\text{max-flt}_{da} &= \text{max-flt}(da) & \text{upper limit for values representable with } da \\
\text{min-flt}_{da} &= \text{min-flt}(da) & \text{lower limit for values representable with } da
\end{align*}
\]
for is-arithmetic(da) & is-CPLX*s-mode(da):

da_real = \mu(da; s-mode: REAL) \quad \text{real data attribute corresponding to da} \\

for is-op(op), is-op(op-1), is-op(op-2):

da_op = s-da(op) \quad \text{data attribute part of operand op} \\
da_{op-1} = s-da(op-1) \quad \text{data attribute part of operand op-1} \\
da_{op-2} = s-da(op-2) \quad \text{data attribute part of operand op-2} \\
v_{op} = s-vr(op) \quad \text{value representation part of operand op} \\
v_{op-1} = s-vr(op-1) \quad \text{value of operand op-1} \\
v_{op-2} = s-vr(op-2) \quad \text{value of operand op-2}
8.1 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-val-da and is essentially the class of evaluated scalar data attributes, excluding areas. Some of the values and their associated attributes are said to be elementary and satisfy the predicate is-elem-val resp. is-elem-da. Others, namely complex numeric values and string values, are composed of elementary values.

Given a value v and a matching data attribute da, the function represent(da,v) may be applied to yield a representation of v. Conversely, given a value representation vr and a matching data attribute da, the function value(da,vr) will yield a value again. The concept of value representation was introduced to cope with the situation that a value cannot always be stored and retrieved without loss of accuracy. The functions value and represent are defined in terms of the functions val and rep that treat the elementary cases; only under certain restrictions val and rep are inverses of one another.

8.1.1 A class of data attributes

(1) is-val-da =

   is-arithm & is-intg-val*s-prec v is-string & is-intg-val*s-length v
   is-pic v is-elem-1-da

(2) is-elem-da =

   is-arithm & is-REAL*s-mode v is-CHAR v is-BIT v is-elem-1-da

Note: Besides the elementary members of is-val-da, the class is-elem-da also contains the "degenerate" attributes CHAR and BIT.

(3) is-elem-1-da = is-LABEL v is-PTR v is-OFFSET v is-TASK v is-EVENT
8.1.2 Values

(4) \( \text{is-value} = \text{is-num-val} \lor \text{is-char-val-list} \lor \text{is-bit-val-list} \lor \text{is-elem-1-val} \)

(5) \( \text{is-elem-val} = \text{is-real-val} \lor \text{is-char-val} \lor \text{is-bit-val} \lor \text{is-elem-1-val} \)

(6) \( \text{is-elem-1-val} = \text{is-label-den} \lor \text{is-format-den} \lor \text{is-ptr-val} \lor \text{is-event-val} \)

Ref.: \( \text{is-label-den} 2-20(49), \) \( \text{is-format-den} 2-20(50), \)
\( \text{is-ptr-val} 2-21(52), \) \( \text{is-event-val} 4-7(13) \)

8.1.2.1 Numeric values

(7) \( \text{is-num-val} = \)

Note: This predicate characterizes the class of (real and complex) numbers that either are rational or have rational real and imaginary part.

(8) \( \text{is-real-val}(v) = \text{is-num-val}(v) \land \text{imag}(v) = 0 \)

8.1.2.2 Character values

(9) \( \text{is-char-val} = \)
\( \text{is-alpham-char} \lor \text{is-BLANK} \lor \text{is-APOSTR} \lor \text{is-EQ} \lor \text{is-PLUS} \lor \text{is-MINUS} \lor \text{is-ASTER} \lor \text{is-SLASH} \lor \text{is-LEFT-PAR} \lor \text{is-RIGHT-PAR} \lor \text{is-COMMA} \lor \text{is-POINT} \lor \text{is-SEMIC} \lor \text{is-COLON} \lor \text{is-AND} \lor \text{is-OR} \lor \text{is-NOT} \lor \text{is-GT} \lor \text{is-LT} \lor \text{is-QUEST} \lor \text{is-PERC} \lor \text{is-extralingual-char} \)

(10) \( \text{is-alpham-char} = \text{is-letter} \lor \text{is-digit} \lor \text{is-BREAK} \)

(11) \( \text{is-letter} = \)
\( \text{is-A-CHAR} \lor \text{is-B-CHAR} \lor \text{is-C-CHAR} \lor \text{is-D-CHAR} \lor \text{is-E-CHAR} \lor \text{is-F-CHAR} \lor \text{is-G-CHAR} \lor \text{is-H-CHAR} \lor \text{is-I-CHAR} \lor \text{is-J-CHAR} \lor \text{is-K-CHAR} \lor \text{is-L-CHAR} \lor \text{is-M-CHAR} \lor \text{is-N-CHAR} \lor \text{is-O-CHAR} \lor \text{is-P-CHAR} \lor \text{is-Q-CHAR} \lor \text{is-R-CHAR} \lor \text{is-S-CHAR} \lor \text{is-T-CHAR} \lor \text{is-U-CHAR} \lor \text{is-V-CHAR} \lor \text{is-W-CHAR} \lor \text{is-X-CHAR} \lor \text{is-Y-CHAR} \lor \text{is-Z-CHAR} \lor \text{is-DOLLAR} \lor \text{is-COMM-AT} \lor \text{is-NUMBER-SIGN} \)
(12) \text{is-digit} =
\text{is-0-CHAR} \lor \text{is-1-CHAR} \lor \text{is-2-CHAR} \lor \text{is-3-CHAR} \lor \text{is-4-CHAR} \lor \text{is-5-CHAR} \lor 
\text{is-6-CHAR} \lor \text{is-7-CHAR} \lor \text{is-8-CHAR} \lor \text{is-9-CHAR}

(13) \text{is-extralingual-char} =

Note: This predicate is implementation-defined.

8.1.2.3 Bit values

(14) \text{is-bit-val} = \text{is-0-BIT} \lor \text{is-1-BIT}

8.1.2.4 Matching elementary values and attributes

(15) \text{is-elem-type-match}\text{(da,v)} =

\text{(is-arithm} \land \text{is-REAL-s-mode)}(\text{da}) \land \text{is-real-val}(\text{v}) \lor 
\text{is-CHAR}(\text{da}) \land \text{is-char-val}(\text{v}) \lor \text{is-BIT}(\text{da}) \land \text{is-bit-val}(\text{v}) \lor 
\text{is-LABEL}(\text{da}) \land \text{(is-label-den} \lor \text{is-format-den)}(\text{v}) \lor 
\text{(is-PTR \lor \text{is-OFFSET})}(\text{da}) \land \text{is-trp-val}(\text{v}) \lor 
\text{is-TASK}(\text{da}) \land \text{is-intg-val}(\text{v}) \lor \text{is-EVENT}(\text{da}) \land \text{is-event-val}(\text{v}) 

\text{for: is-elem-da}(\text{da}) \land \text{is-elem-val}(\text{v})

Ref.: \text{is-label-den} 2-20(49), \text{is-format-den} 2-20(50), \text{is-real-val} 8-4(8), 
\text{is-ptr-val} 2-21(52), \text{is-event-val} 4-7(13), \text{is-bit-val} 8-5(14), 
\text{is-char-val} 8-4(9)

8.1.3 Value representations

Value representations are characterized by the implementation-defined predicate \text{is-vr} (cf. 2-22(69)). Complex numerical values and strings have composite representations; the predicates

(16) \text{is-real-vr}, \text{is-char-vr}, \text{is-bit-vr}

are again implementation-defined, but satisfy the following axiom:
(17) \( (\forall \text{vr})(\text{is-cplx-vr} (\text{vr}) \lor \text{is-real-vr} (\text{vr}) \lor \text{is-char-vr-list} (\text{vr}) \lor \text{is-bit-vr-list} (\text{vr}) \lor \text{is-char-vr} (\text{vr}) \lor \text{is-bit-vr} (\text{vr}) \supset \text{is-vr} (\text{vr})) \)

Ref.: is-vr 2-22(69)

Note: The implication sign expresses that there may be further sub-classes of is-vr. On the other hand, not even the sub-classes appearing in the formula are necessarily disjoint.

(18) is-cplx-vr = (\langle \text{s-real}: \text{is-real-vr}\rangle, \langle \text{s-imag}: \text{is-real-vr}\rangle)

8.1.4 Transition between a value and its representation

In 8.1.4.1, the relation between elementary values and their representations is treated; this is done axiomatically, by introducing two implementation-defined functions val and rep and formulating certain postulates on them. In 8.1.4.2 and 8.1.4.3, functions value and represent for the treatment of arbitrary values and their representations are defined in terms of val and rep; also, an instruction test-rep is defined that raises the SIZE condition if a value cannot be represented.

Metavariable

da is-val-da cf. 8.1.1

8.1.4.1 Representing and retrieving elementary values

For each elementary data attribute \( da \), the set \( vr-set(da) \) of value representations (defined in 6-11(22) for arbitrary scalar \( da \)) is taken, for the given \( da \), as the domain of the function \( val(da, vr) \). Also, a set \( v-set(da) \) of values is defined, which is the domain of the function \( rep(da, v) \); this is the set of all values whose type matches \( da \) and which, when represented with \( da \), do not raise the SIZE condition. Finally, a subset \( v-0-set(da) \) of \( v-set(da) \) is defined, which for non-arithmetic \( da \) is \( v-set(da) \) itself, and for arithmetic \( da \) is determined by the precision and scale-factor of \( da \). This set \( v-0-set(da) \) has the important property that its members are exactly representable with the given \( da \), i.e. the following can be derived from the axioms given below:

\[
(\forall \text{da,v})(\text{is-elem-da} (\text{da}) \land \text{is-elem-val} (\text{v}) \supset (\text{v} \in v-0-set (\text{da}) \supset val (\text{da}, rep (\text{da}, \text{v})) = \text{v}))
\]
Metavariable:

\( \text{da} \quad \text{is-elem-da} \quad \text{in this section only, and with the exception of 8-7(20).} \)

(19) \( \text{v-set}(\text{da}) = \)

\[ \{ v \mid \text{is-elem-val}(v) \land \text{is-elem-type-match}(\text{da}, v) \land (\text{is-arithm}(\text{da}) \Rightarrow \neg \text{is-size-cond}(\text{da}, v)) \} \]

Ref.: is-elem-val 8-4(5), is-elem-type-match 8-5(15), is-size-cond 8-9(32)

(20) \( \text{v-O-set}(\text{da}) = \)

\( (\text{is-arithm} \land \text{is-CPLX-s-mode})(\text{da}) \rightarrow \)

\[ \{ \text{cplx}(v\text{-re}, v\text{-im}) \mid v\text{-re} \in \text{v-O-set}(\text{da}_{\text{real}}) \land v\text{-im} \in \text{v-O-set}(\text{da}_{\text{real}}) \} \]

\( (\text{is-arithm} \land \text{is-FIX-s-scale})(\text{da}) \rightarrow \)

\[ \{ v \mid \text{is-intg-val}(v\text{.base}_{\text{da}} \uparrow q_{\text{da}}) \land \text{abs}(v\text{.base}_{\text{da}} \uparrow q_{\text{da}} < \text{base}_{\text{da}} \uparrow p_{\text{da}}) \} \]

\( (\text{is-arithm} \land \text{is-FLT-s-scale})(\text{da}) \rightarrow \)

\[ \{ v \mid (\exists n)(\text{is-intg-val}(n) \land \text{is-intg-val}(v\text{.base}_{\text{da}} \uparrow n) \land \text{abs}(v\text{.base}_{\text{da}} \uparrow n < \text{base}_{\text{da}} \uparrow p_{\text{da}}) \land \) \]

\( (v = 0 \lor \text{min-flt}_{\text{da}} \leq \text{abs}(v) < \text{max-flt}_{\text{da}}) \}

T \rightarrow \text{v-set}(\text{da})

for: is-elem-da(\text{da}) \lor \text{is-arithm}(\text{da})

Note: For arithmetic \( \text{da} \), this definition expresses the rule that \( p_{\text{da}} \) is "the number of digits" and \( q_{\text{da}} \) 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 \( \text{da} \).

The functions

(21) \( \text{val}(\text{da}, v) \)

(22) \( \text{rep}(\text{da}, v) \)

are implementation-defined and are characterized by the axioms

8-8(23/24/25).
(23) \( \forall da (\text{is-elem-da}(da) \supset v-0-set(da) \subseteq \{ \text{val}(da, vr) \mid vr \in \text{vr-set}(da) \} \subseteq v-set(da)) \)

Ref.: is-elem-da 8-3(2), vr-set 6-11(24)

Note: This axiom defines domain and range of the function \( \text{val}(da, vr) \) and ensures that, for given \( da \), the range is at least \( v-0-set(da) \).

(24) \( \forall da (\text{is-elem-da}(da) \supset \{ \text{rep}(da, v) \mid v \in v-set(da) \} \subseteq \text{vr-set}(da)) \)

Ref.: is-elem-da 8-3(2), vr-set 6-11(24)

Note: This axiom defines domain and range of the function \( \text{rep}(da, v) \).

(25) \( \forall da, vr (\text{is-elem-da}(da) \& vr \in \text{vr-set}(da) \supset \text{rep}(da, \text{val}(da, vr)) = vr) \)

Ref.: is-elem-da 8-3(2), vr-set 6-11(24)

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 concerns the opposite transition: from a value via a representation back to the value.)

8.1.4.2 Retrieving a scalar value from its representation

(26) \( \text{value}(da, vr) = \)

\[(\text{is-arithm} \& \text{is-CPLX}s\text{-mode})(da) \rightarrow \]

\[\text{cplx}(\text{value}(da_{\text{real}}, s\text{-real}(vr)), \text{value}(da_{\text{real}}, s\text{-imag}(vr))) \]

\[\text{is-string}(da) \rightarrow \text{val-list}(s\text{-base}(da), vr) \]

\[\text{is-pic}(da) \rightarrow \text{pic-val}(da, vr) \]

\[\text{is-arithm}(da) \supset \text{is-prop-arithm}(da) \rightarrow \text{val}(da, vr) \]

T \rightarrow \text{error}

Ref.: pic-val 8-61(230), val 8-7 (21).

(27) \( \text{val-list}(base, vr) = \sum_{i=0}^{\text{length}(vr)} \text{val}(base, \text{elem}(i; vr)) \)

for: (is-char-vr-list \& is-bit-vr-list)(vr)

Ref.: val 8-7 (21).

(28) \( \text{is-prop-arithm}(da) = (\text{is-}\mathcal{Q}\text{-s-scale}(da) \equiv \text{is-FLT}\text{-s-scale}(da)) \& \text{O}\text{p}_{da}\text{N}_{da} \)

for: is-arithm(da)

8.1.4.2
8.1.4.3 Representing a scalar value

Before representing a value, the instruction `test-rep` (da,v) tests whether the SIZE condition has to be raised. If it is known that no condition can occur, then the function `represent` (da,v) can be used directly.

\[(29)\) \text{test-rep}(da,v) = \]
\[
\begin{align*}
\text{if (is-arithm & is-CPLX\_s\_mode)(da)} & \rightarrow \text{pass-cplx-vr(vr-re, vr-im)}; \\
\text{vr-re: rep-real(da\_real, real(v))}, \\
\text{vr-im: rep-real(da\_real, imag(v))} \\
\text{is-pic(da)} & \rightarrow \text{rep-pic(da,v)} \\
T & \rightarrow \text{PASS:represent(da,v)}
\end{align*}
\]

Ref.: `rep-pic` 8-51(196)

\[(30)\) \text{cplx-vr(vr-re, vr-im)} = \mu_o(s\_real:vr-re, s\_imag:vr-im) \]

\[(31)\) \text{rep-real}(da,v) = \]
\[
\begin{align*}
\text{is-size-cond(da,v)} & \rightarrow \text{error}; \\
\text{call-cond(SIZE, 601)} \\
T & \rightarrow \text{PASS:represent(da,v)}
\end{align*}
\]

Ref.: `call-cond` 9-11(18)

\[(32)\) \text{is-size-cond(da,v)} = \]
\[
\begin{align*}
\text{is-FIX\_s\_scale(da)} & \rightarrow \text{abs}(v.\text{base}_da \uparrow q_{da}) \geq \text{base}_da \uparrow p_{da} \\
\text{is-FLT\_s\_scale(da)} & \rightarrow v=0 \& \neg(\text{min-flt}_da \leq \text{abs}(v) < \text{max-flt}_da)
\end{align*}
\]

for: (is-arithm & is-REAL\_s\_mode)(da) \& is-real-val(v)

\[(33)\) \text{represent}(da,v) = \]
\[
\begin{align*}
\text{if (is-arithm & is-CPLX\_s\_mode)(da)} & \rightarrow \text{cplx-vr(\text{represent}(da\_real, real(v)), represent(da\_real, imag(v)))} \\
\text{cont'd}
\end{align*}
\]
is-arithm(da) → (is-prop-arithm(da) → rep(da, real(v)), T → error)

is-string(da) → rep-list(s-base(da), adjust-string(s-base(da), result-l(da,v),v))
T → rep(da,v)

for : is-val-da(da) & ~is-pic(da)
Ref.: rep 8-7(22)

(34) rep-list(base,v) = \[
\bigl\lbrack I \bigr\rbrack_{i=1}^{\text{length}(v)} \text{rep}(\text{base}, \text{elem}(i,v))
\]

for : (is-char-val-list v is-bit-val-list)(v)
Ref.: rep 8-7(22)

(35) result-l(da,v) =

\[
\text{is-Q -s-varying(da) → s-length(da)}
\]

T → min(length(v), s-length(da))

for : is-string(da) & (is-char-val-list v is-bit-val-list)(v)

(36) adjust-string(base,n,v) =

\[
\bigl\lbrack I \bigr\rbrack_{i=1}^{n} (\text{is-length}(v) → \text{elem}(i,v), T → \text{fill-char}(base))
\]

for : (is-char-val-list v is-bit-val-list)(v)

(37) fill-char(base) =

\[
\text{is-CHAR}(base) → \text{BLANK}
\]

\[
\text{is-BIT}(base) → \text{O-BIT}
\]

8.1.5 Operands

The first formula defines an operand as an object that has two parts, a data attribute and a value representation. The rest are two auxiliary instructions 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.
(38) \( \text{is-op} = (\langle s\text{-da:is-da}, s\text{-vr:is-vr} \rangle) \)

Ref.: is-vr 2-22(69)

Note: Actually, only operands will be produced whose da-part satisfies
is-val-da \lor is-area.

(39) \( \text{mk-op}(\text{da, vr}) = \)

\[ \text{PASS}: \mu_0(\langle s\text{-da:da}, s\text{-vr:vr} \rangle) \]

(40) \( \text{op-val}(\text{op}) = \text{value}(\text{da}_{\text{op}}, \text{vr}_{\text{op}}) \)

for : \( \neg \) is-area(\text{da}_{\text{op}})

Ref.: value 8-8(26)

(41) \( \text{val-op}(\text{da, v}) = \mu_0(\langle s\text{-da:da}, s\text{-vr:represent(da,v)} \rangle) \)

for : is-val-da(da) \& \neg is-pi(da)

Ref.: represent 8-9(33)

Note: Since represent and not test-rep appears, the function can only be used
in cases where no SIZE condition can arise.

(42) \( \text{intg-op}(i) = \text{val-op}(\text{INTG-DA}, i) \)

Ref.: INTG-DA 8-12(52)

(43) \( \text{bintg-op}(i) = \text{val-op}(\text{BINTG-DA}, i) \)

Ref.: BINTG-DA 8-12(53)

(44) \( \text{is-intg-op} = \text{is-op} \& \text{is-intg-s-da} \)

Ref.: is-intg 8-13(54)

(45) \( \text{is-bintg-op} = \text{is-op} \& \text{is-bintg-s-da} \)

Ref.: is-bintg 8-13(55)
8.1.6 Operators

An operator is either an infix operator, or a prefix operator, or the object C-EXP introduced by the instruction eval-expr to treat a special case of exponentiation.

(46) is-operator \( = \) is-infix-operator \( \lor \) is-prefix-operator \( \lor \) is-C-EXP

(47) is-arithm-opor =

\[
\text{is-ADD} \lor \text{is-SUBTR} \lor \text{is-MULT} \lor \text{is-DIV} \lor \text{is-EXP} \lor \text{is-C-EXP} \lor \text{is-PLUS} \lor \text{is-MINUS}
\]

(48) is-comp-opor =

\[
\text{is-GT} \lor \text{is-GE} \lor \text{is-EQ} \lor \text{is-LE} \lor \text{is-LT} \lor \text{is-NE}
\]

(49) is-bit-opor = is-AND \( \lor \) is-OR \( \lor \) is-NOT

8.2 Auxiliary Definitions on Data Attributes

The definitions given in this section are used both in chapter 8 and in other chapters. In 8.2.1, integer data attributes, functions producing integer or string data attributes of given precision or length, and a few incomplete data attributes are defined. The functions listed in 8.2.2 yield implementation-defined limits associated with arithmetic data attributes.

9.2.1 Special data attributes and classes of data attributes

(50) \( \text{intg-da}(n) = \) \( \mu_0(\langle s\text{-mode:REAL}, s\text{-base:DEC}, s\text{-scale:FIX}, s\text{-prec:n}, s\text{-scale-f:O} \rangle) \)

(51) \( \text{bintg-da}(n) = \) \( \mu_0(\langle s\text{-mode:REAL}, s\text{-base:BIN}, s\text{-scale:FIX}, s\text{-prec:n}, s\text{-scale-f:O} \rangle) \)

(52) \( \text{INTG-DA} = \text{intg-da}(\text{DEF-PREC-DEC}) \)

(53) \( \text{BINTG-DA} = \text{bintg-da}(\text{DEF-PREC-BIN}) \)

Note: The elementary objects DEF-PREC-DEC and DEF-PREC-BIN are implementation-defined integers, the default precisions for decimal or binary arithmetic data attributes.
(54) \( \text{is-intg}(da) = (\exists n)(da = \text{intg}-da(n)) \)

(55) \( \text{is-bintg}(da) = (\exists n)(da = \text{bintg}-da(n)) \)

(56) \( \text{AR-DA} \equiv \mu_0(<\text{mode}:*, <\text{base}:*, <\text{scale}:*, <\text{prec}:*>) \)

(57) \( \text{STRING-DA} \equiv \mu_0(<\text{base}:*, <\text{length}:*, <\text{varying}:*>) \)

(58) \( \text{CHAR-DA} = \mu(\text{STRING-DA}; <\text{base}: \text{CHAR}>) \)

(59) \( \text{BIT-DA} = \mu(\text{STRING-DA}; <\text{base}: \text{BIT}>) \)

Note: The last four definitions introduce special "incomplete data attributes" (cf. 8.5.2).

(60) \( \text{char-da}(n) = \mu_0(<\text{base}: \text{CHAR}, <\text{length}:n>) \)

(61) \( \text{bit-da}(n) = \mu_0(<\text{base}: \text{BIT}, <\text{length}:n>) \)

8.2.2 Implementation-defined limits associated with arithmetic data attributes

(62) \( \text{max-prec}(da) = \text{max-prec-1}(\text{base}(da), \text{scale}(da)) \)

for : is-arithm(da)

Note: The definition expresses that max-prec(da), the maximum precision associated with da, depends only on the base and scale of da. The four possible values of max-prec-1 are implementation-defined integers.

(63) \( \text{max-flt}(da) = \text{max-flt-1}(\text{base}(da)) \)

for : (is-arithm & is-FLT & s-scale)(da)

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 da.

(64) \( \text{min-flt}(da) = \text{min-flt-1}(\text{base}(da)) \)

for : (is-arithm & is-FLT & s-scale)(da)

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 da.

8.2.2
8.3 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 `CEXP` (cf. `eval-expr` 7.2.1).

First, the operands are converted to the type 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 8.3.1, the functions `eda-infix` and `eda-intg-exp` which compute the result attributes are defined in 8.3.2.

(65) \[
\text{eval-infix-expr}(\text{op-1}, \text{op-2}, \text{opor}) =
\]

\[
\text{infix-op}(\text{c-op-1}, \text{c-op-2}, \text{opor});
\]

\[
c-\text{op-1}: \text{convert}(\text{datg-1}, \text{op-1}),
\]

\[
c-\text{op-2}: \text{convert}(\text{datg-2}, \text{op-2})
\]

where:

\[
\text{datg-1} =
\]

\[
is-C-\text{EXP}(\text{opor}) \rightarrow \text{c-exp-target-1}(\text{mk-arithm}(\text{da}_{\text{op-1}}), \text{da}_{\text{op-2}}, \text{v}_{\text{op-2}})
\]

\[
\text{T} \rightarrow \text{target-1}(\text{da}_{\text{op-1}}, \text{da}_{\text{op-2}}, \text{opor})
\]

\[
\text{datg-2} = \text{target-2}(\text{da}_{\text{op-1}}, \text{da}_{\text{op-2}}, \text{opor})
\]

Ref.: `convert` 8-30(118), `mk-arithm` 8-38(149)

(66) \[
\text{infix-op}(\text{op-1}, \text{op-2}, \text{opor}) =
\]

\[
\text{pass-val-op}(\text{da}_{\text{res}}, \text{v});
\]

\[
\text{v}: \text{infix-val}(\text{da}_{\text{op-1}}, \text{da}_{\text{op-2}}, \text{v}_{\text{op-1}}, \text{v}_{\text{op-2}}, \text{opor})
\]

where:

\[
\text{da}_{\text{res}} =
\]

\[
is-C-\text{EXP}(\text{opor}) \rightarrow \text{res-c-exp-da}(\text{da}_{\text{op-1}}, \text{da}_{\text{op-2}}, \text{v}_{\text{op-2}}),
\]

\[
\text{T} \rightarrow \text{res-da}(\text{da}_{\text{op-1}}, \text{da}_{\text{op-2}}, \text{opor})
\]

Ref.: `val-op` 8-11(41)
8.3.1 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, 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.

\[
\text{infix-val}(da\_1, da\_2, v\_1, v\_2, opor) =
\begin{align*}
\text{is-arithm-opor}(opor) \rightarrow \\
\quad \text{test-fl}(da\_res, v); \\
\quad v: \text{test-infnum}(da\_1, da\_2, v\_1, v\_2, opor)
\end{align*}
\]

\[
\begin{align*}
\text{is-comp-opor}(opor) \rightarrow \\
\quad \text{PASS: truth-to-bit(is-true-comp}(da\_1, da\_2, v\_1, v\_2, opor))
\end{align*}
\]

\[
\begin{align*}
\text{is-bit-opor}(opor) \rightarrow \\
\quad \text{PASS: infix-bit(adjust-string(BIT, lgth\_max, v\_1)}, \\
\quad \quad \text{adjust-string(BIT, lgth\_max, v\_2)}, opor)
\end{align*}
\]

\[
\begin{align*}
\text{is-CAT}(opor) \rightarrow \\
\quad \text{PASS: v\_1 \bigcap v\_2}
\end{align*}
\]

where: \( da\_res = (\text{is-C-EXP}(opor) \rightarrow \text{res-c-exp-da}(da\_1, da\_2, v\_2), \)
\[
T \rightarrow \text{res-da}(da\_1, da\_2, opor)
\]

\[
lgth\_\text{max} = \text{max(length(v\_1), length(v\_2))}
\]

Ref.: adjust-string 8-10(36), res-c-exp-da 8-27(107), res-da 8-25(101),
is-arithm-opor 8-12(47), is-comp-opor 8-12(48), is-bit-opor 8-12(49)

8.3.1.1 Arithmetic operations

The instruction `test-infnum` tests for certain condition situations and then applies `infnum` to calculate the result. The instruction `test-fl` tests for overflow and underflow.

8.3.1.1
Metavariables

\[ \text{da-1, da-2, da} \quad \text{is-arithm} \]
\[ \text{v-1, v-2, v} \quad \text{is-num-val} \]
\[ \text{opor} \quad \text{is-arithm-opor} \]

Abbreviation

\[ \text{cond}_{\text{inf}} = \text{cond-infix}(\text{da-1}, \text{da-2}, \text{v-1}, \text{v-2}) \]

(68) \quad \text{test-infix-num}(\text{da-1}, \text{da-2}, \text{v-1}, \text{v-2}, \text{opor}) =

\[ \text{is-} \{ \text{cond}_{\text{inf}} \} \quad \rightarrow \quad \text{PASS}; \text{infix-num}(\text{da-1}, \text{da-2}, \text{v-1}, \text{v-2}, \text{opor}) \]
\[ \text{T} \quad \rightarrow \quad \text{error}; \quad \text{call-cond}(\text{cond}_{\text{inf}}, 602) \]

Ref.: \text{call-cond} 9-11(18)

(69) \quad \text{cond-infix}(\text{da-1}, \text{da-2}, \text{v-1}, \text{v-2}, \text{opor}) =

\[ \text{is-DIV} (\text{opor}) \land \text{v-2} = 0 \quad \rightarrow \quad \text{ZDIV} \]
\[ \text{(is-EXP} \lor \text{is-C-EXP)} (\text{opor}) \land \]
\[ (\text{v-1} = 0 \land \neg(\text{imag}(\text{v-2}) = 0 \land \text{real}(\text{v-1}) > 0) \lor \]
\[ \text{is-REAL} \	ext{is-mode}(\text{da-1}) \land \text{v-1} \leq 0 \land \neg(\text{is-intg} \lor \text{is-bintg})(\text{da-2}) \quad \rightarrow \quad \text{ERROR} \]
\[ \text{T} \quad \rightarrow \quad \text{Q} \]

Ref.: \text{is-intg} 8-13(54), \text{is-bintg} 8-13(55)

(70) \quad \text{infix-num}(\text{da-1}, \text{da-2}, \text{v-1}, \text{v-2}, \text{opor})

Note: This function is characterized by the axiom 8-16(71).

(71) \quad (\forall \text{da-1, da-2, v-1, v-2, opor})(\text{is-} \{ \text{cond}_{\text{inf}} \} \land \)
\[ \text{(is-FIX} \land \text{is-scale}(\text{da-1}) \land \text{is-FIX} \land \text{is-scale}(\text{da-2}) \land \neg \text{is-DIV}(\text{opor}) \land \]
\[ \text{v-1} \in v-0\text{-set}(\text{da-1}) \land \text{v-2} \in v-0\text{-set}(\text{da-2}) \lor \]
\[ (\text{is-EXP} \lor \text{is-C-EXP}) (\text{opor}) \land (\text{v-1} = 0 \lor \text{v-2} = 0) \lor \]
\[ \text{infix-num}(\text{da-1}, \text{da-2}, \text{v-1}, \text{v-2}, \text{opor}) = \text{acc-infix-num}(\text{v-1}, \text{v-2}, \text{opor}) \]

cont'd
Ref.: v-0-set 8-7(20)

Note: This axiom defines \textit{infix-num} for the \textit{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, \textit{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).

\begin{equation}
\text{acc-infix-num}(v_1, v_2, \text{opor}) =
\begin{align*}
\text{is-ADD}(\text{opor}) & \quad \rightarrow v_1 + v_2 \\
\text{is-SUBTR}(\text{opor}) & \quad \rightarrow v_1 - v_2 \\
\text{is-MULT}(\text{opor}) & \quad \rightarrow v_1 \cdot v_2 \\
\text{is-EXP } \vee \text{is-C-EXP}(\text{opor}) & \quad \rightarrow v_1^t v_2
\end{align*}
\end{equation}

\begin{equation}
\text{test-fl}(\text{da}, v) =
\begin{align*}
\text{is-CPLX } \ast \text{mode}(\text{da}) & \quad \rightarrow \text{pass-cplx}(v_{\text{re}}, v_{\text{im}}); \\
& \quad \text{v_{re} : test-fl}(\text{da}_{\text{real}}, \text{real}(v)), \\
& \quad \text{v_{im} : test-fl}(\text{da}_{\text{real}}, \text{imag}(v)) \\
\text{is-FIX } \ast \text{scale}(\text{da}) & \quad \& \quad \text{is-fofl-cond}(\text{da}, v) \quad \rightarrow \text{error}; \\
\text{call-cond}(\text{FOFL}, 603)
\end{align*}
\end{equation}

\begin{equation}
\begin{align*}
\text{is-FLT } \ast \text{scale}(\text{da}) & \quad \& \quad \text{is-ofl-cond}(\text{da}, v) \quad \rightarrow \text{error}; \\
\text{call-cond}(\text{OFL}, 604)
\end{align*}
\end{equation}

\begin{equation}
\begin{align*}
\text{is-FLT } \ast \text{scale}(\text{da}) & \quad \& \quad \text{is-ufl-cond}(\text{da}, v) \quad \rightarrow \text{pass}(0); \\
\text{call-cond}(\text{UFL}, 605)
\end{align*}
\end{equation}

\begin{equation}
\begin{align*}
\text{T} & \quad \rightarrow \text{PASS} : v
\end{align*}
\end{equation}

Ref.: \text{call-cond} 9-11(18)

\begin{equation}
\text{is-fofl-cond}(\text{da}, v) =
\begin{align*}
\text{abs}(v. \text{base} \uparrow q. \text{da}) & \quad \rightarrow \text{base} \uparrow q. \text{da} \\
\text{for: } (\text{is-REAL } \ast \text{mode } \& \text{is-FIX } \ast \text{scale}(\text{da}) \& \text{is-real-val}(v)
\end{align*}
\end{equation}

8.3.1.1
8.3.1.2 Comparison Operations

First, the interpretation of the operator NE 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 a bit string of length one.

Metavariable

\[ \text{is-}T \; \text{v} \; \text{is-F} \quad \text{truth value} \]

\[ \text{truth-to-bit}(x) = \begin{cases} \text{<1-BIT>}, & \text{if } \text{is-T}(x) \\ \text{<0-BIT>}, & \text{if } \text{is-F}(x) \end{cases} \]

\[ \text{is-true-comp}(da-1, da-2, v-1, v-2, opor) = \]

\[ \begin{align*}
\text{is-NE(opor)} & \rightarrow \neg \text{is-true-comp}(da-1, da-2, v-1, v-2, \text{EQ}) \\
\text{is-GE(opor)} & \rightarrow \neg \text{is-true-comp}(da-1, da-2, v-1, v-2, \text{LT}) \\
\text{is-LE(opor)} & \rightarrow \neg \text{is-true-comp}(da-1, da-2, v-1, v-2, \text{GT}) \\
\text{is-LT(opor)} & \rightarrow \text{is-true-comp}(da-2, da-1, v-2, v-1, \text{GT}) \\
\text{is-arithm}(da-1) & \Rightarrow (\text{is-CPLX-s-mode}(da-1) \Rightarrow \text{is-EQ(opor)})) \\
\text{is-true-num-comp}(da-1, da-2, v-1, v-2, opor) & \\
\text{is-string}(da-1) & \\
\text{is-true-string-comp}(s\text{-base}(da-1), \text{adjust-string}(s\text{-base}(da-1), \text{lgth}_{\text{max}}, v-1), \\
\text{adjust-string}(s\text{-base}(da-2), \text{lgth}_{\text{max}}, v-2), opor) & \\
\text{(is-PTR v is-OFFSET)(da-1) & is-EQ(opor)} & \\
\text{is-equal-ptr}(v-1, v-2) & \\
\end{align*} \]

\[ T \rightarrow \text{error} \]
8.3.1.2.1 Numeric comparison

Metavariables

\[ \text{da}-1, \text{da}-2, \text{is-arithm} \]
\[ \text{v}-1, \text{v}-2, \text{is-num-val} \]
\[ \text{opor, is-EQ \lor is-GT} \]

where: \( \text{lgth}_{\text{max}} = \max(\text{length(v}-1), \text{length(v}-2)) \)

for: \( \text{is-comp-opor(opor)} \)

Ref.: adjust-string 8-10(36)

(79) \( \text{is-true-num-comp(da}-1, \text{da}-2, \text{v}-2, \text{v}-2, \text{opor}) \)

Note: This function is characterized by the axiom 8-19(80).

(80) \( (\forall \text{da}-1, \text{da}-2, \text{v}-1, \text{v}-2, \text{opor} ) \)

\( \text{(is-FIX\_s-scale(da}-1) \ \& \ \text{is-FIX\_s-scale(da}-2) \ \& \ \text{v}-1 \in \text{v}-0\text{-set(da}-1) \ \& \ \text{v}-2 \in \text{v}-0\text{-set(da}-2) \ \Rightarrow \ \text{is-true-num-comp(da}-1, \text{da}-2, \text{v}-1, \text{v}-2, \text{opor} ) = \text{is-true-acc-num-comp(v}-1, \text{v}-2, \text{opor} ) ) \)

Ref.: v-0-set 8-7(20)

Note: This axiom defines \( \text{is-true-num-comp} \) for the FIXED case, provided that the argument values are exactly representable with their data attributes. In all other cases, \( \text{is-true-num-comp} \) is implementation-defined and is assumed to be an approximation of the exact mathematical comparison operation.

(81) \( \text{is-true-acc-num-comp(v}-1, \text{v}-2, \text{opor} ) = \)

\( \text{(is-EQ(opor) \rightarrow v}-1 = v}-2, \text{is-GT(opor) \rightarrow v}-1 > v}-2 ) \)

8.3.1.2.2 String comparison

Character strings are compared according to an implementation-defined collating sequence, bit strings according to their interpretation as binary integers.
The predicate

\[(84) \text{is-equal-ptr} \]

is characterized by the axiom \(8-21(85)\), which guarantees an implementation-independent result only if the two pointer values are either identical or independent (cf. 2.4.1.1).
(85) \((\forall p-1, p-2)((p-1 = p-2 \land \text{is-equal-\text{ptr}(v-1, p-2)}) \land
  (p-1 \neq p-2 \land (\text{is-NPTR}(v-1) \lor \text{is-NPTR}(p-2) \lor \text{is-indep-1}(p-1, p-2)) \lor
  \text{is-equal-\text{ptr}(v-1, p-2)))\)

Ref.: is-indep-1 2-22(67), \text{NPTR} 11-49(162)

8.3.1.3 Bit string operations

(86) \text{infix-bit}\(v-1, v-2, o\text{por}) = \left\{\begin{array}{ll}
\text{length}(v-1) & \text{single-infix-bit}(\text{elem}(i, v-1), \text{elem}(i, v-2), o\text{por}) \\
\text{length}(v-2) & \text{single-infix-bit}(v-1, v-2, o\text{por})
\end{array}\right.

\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-opor}(o\text{por})

(87) \text{single-infix-bit}\(v-1, v-2, o\text{por}) =

\text{is-AND}(o\text{por}) \quad \rightarrow \quad (\text{is-1-BIT}(v-1) \land \text{is-1-BIT}(v-2) \quad \rightarrow \quad 1\text{-BIT}, T \quad \rightarrow \quad 0\text{-BIT})

\text{is-OR}(o\text{por}) \quad \rightarrow \quad (\text{is-1-BIT}(v-1) \lor \text{is-1-BIT}(v-2) \quad \rightarrow \quad 1\text{-BIT}, T \quad \rightarrow \quad 0\text{-BIT})

\text{for: is-bit-val}(v-1) \land \text{is-bit-val}(v-2) \land \text{is-bit-opor}(o\text{por})

8.3.2 Data attributes of the result

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 eda-infix-expr and treats the scalar case; the latter function is a sub-function of the instruction eda-expr (cf. 7-22(54)) which is used to determine the attributes of an expression independently from its value. On the other hand, only the subfunctions target-1, target-2 and res-da 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 8.3.2.1. The function res-da calculates the result attribute from the attributes of the converted operands and is defined in 8.3.2.2.

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-intg-exp, its sub-functions c-exp-target-1 and res-c-exp-da, and the predicate is-intg-exponent which is used by eval-expr to test for this situation (cf. 7.2.1) are defined in 8.3.2.3.
8.3.2.1 Target attributes for conversion of operands

Except for some cases of exponentiation, the target attributes for the first and the second operand are the same. Precision or length of arithmetic or string target attributes are set to *, because they will be determined by the convert-instruction from the source attributes da-1, da-2. (See complete-tg 8-37(143).)

\[ \text{(89) target-1(da-1,da-2,opor)} = \]
\[ \text{is-EXP(opor)} \rightarrow \mu(\text{target(da-1,da-2,opor)};<\text{s-scale:FLT>}) \]
\[ T \rightarrow \text{target(da-1,da-2,opor)} \]

\[ \text{(90) target-2(da-1,da-2,opor)} = \]
\[ (\text{is-EXP v is-C-EXP})(opor) \rightarrow \]
\[ (<\text{is-intg v is-bintg})(da-2) \rightarrow \mu(da-2;<\text{s-base:pref-base(da-1,da-2)>,}<\text{s-prec:*}>,<\text{s-scale-f:Q}>), \]
\[ T \rightarrow \mu(\text{target(da-1,da-2,opor)};<\text{s-scale:FLT>}) \]
\[ T \rightarrow \text{target(da-1,da-2,opor)} \]

Ref.: is-intg 8-13(54), is-bintg 8-13(55)

\[ \text{(91) target(da-1,da-2,opor)} = \]
\[ \text{is-arithm-opor(opor) v} \]
\[ \text{is-comp-opor(opor) & (is-num-type(da-1) v is-num-type(da-2))} \rightarrow \]
\[ \text{pref-ar-da(mk-arithm(da-1),mk-arithm(da-2))} \]
\[ \text{is-comp-opor(opor) & (is-string-type(da-1) v is-string-type(da-2))} \rightarrow \]
\[ \text{pref-str-da(mk-string(da-1),mk-string(da-2))} \]
\[ \text{is-comp-opor(opor)} \rightarrow \text{da-1} \]  

cont'd
is-CAT(opor) & is-BIT*s-base(da-1) & is-BIT*s-base(da-2) ——
  pref-str-da(da-1,da-2)

is-CAT(opor) ——— pref-str-da(mk-char-string(da-1),mk-char-string(da-2))

is-bit-opor(opor) ——— BIT-DA

Ref.: mk-arithm 8-38(149), mk-string 8-38(150), BIT-DA 8-13(59)
  is-arithm-opor 8-12(47), is-comp-opor 8-12(48),
  is-bit-opor 8-12(49), mk-char-string 8-38(151)

(92) is-num-type = is-arithm v is-num-pic

Ref.: is-num-pic 8-44(166)

(93) is-string-type = is-string v is-char-pic

8.3.2.1.1 Higher arithmetic characteristics

Metavariab1es

da-1, da-2 is-arithm

(94) pref-ar-da(da-1,da-2) =

μ₀(<s-mode:pref-mode(da-1,da-2)>,
  <s-base:pref-base(da-1,da-2)>,
  <s-scale:pref-scale(da-1,da-2)>,
  <s-prec:>)

(95) pref-mode(da-1,da-2) =

is-REAL*s-mode(da-1) & is-REAL*s-mode(da-2) ——— REAL
  T ——— CPLX

(96) pref-base(da-1,da-2) =

is-DEC*s-base(da-1) & is-DEC*s-base(da-2) ——— DEC,
  T ——— BIN

8.3.2.1.1
8.3.2.2 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.
Abbreviations

For is-arithm(da-1) & is-arithm(da-2):

\[ N_1 \text{ max-prec}(da-1) \]
maximum precision permitted for given base and scale

\[ p_1 \text{ s-prec}(da-1) \]
precision of converted first operand

\[ q_1 \text{ s-scale-f}(da-1) \]
scale-factor of converted first operand

\[ p_2 \text{ s-prec}(da-2) \]
precision of converted second operand

\[ q_2 \text{ s-scale}(da-2) \]
scale-factor of converted second operand

For is-string(da-1) & is-string(da-2):

\[ lgth_1 \text{ s-length}(da-1) \]
(maximum) length of converted first operand

\[ lgth_2 \text{ s-length}(da-2) \]
(maximum) length of converted second operand

(101) res-da(da-1,da-2,opor) =

\[ \text{is-arithm-opor}(opor) \]

\[ \text{is-EXP}(opor) \& \text{is-FIX}\text{*s-scale}(da-2) \rightarrow da-1, \]
\[ \text{is-FLT}\text{*s-scale}(da-1) \rightarrow \mu(da-1;\langle s\text{-prec}\text{max}(p_1,p_2)\rangle) \]
\[ \text{is-FIX}\text{*s-scale}(da-1) \rightarrow \]
\[ \mu(da-1;\langle s\text{-prec}\text{min}(N_1,\text{res-prec}(da-1,da-2,opor)), \]
\[ \langle s\text{-scale-f}\text{res-scale-f}(da-1,da-2,opor)\rangle\rangle) \]

\[ \text{is-comp-opor}(opor) \rightarrow \text{bit-da}(1) \]

\[ \text{is-bit-opor}(opor) \rightarrow \mu(da-1;\langle s\text{-length}\text{max}(lgth_1,lgth_2)\rangle) \]

\[ \text{is-CAT}(opor) \rightarrow \mu(da-1;\langle s\text{-length}\text{lgth}_1+lgth_2\rangle) \]

Ref.: bit-da 8-13(61), is-arithm-opor 8-12(47),

is-comp-opor 8-12(48), is-bit-opor 8-12(49)

(102) res-prec(da-1,da-2,opor) =

\[ \text{(is-ADD v is-SUBTR)(opor)} \rightarrow \max(p_1-q_1,p_2-q_2)+\max(q_1,q_2)+1 \]
\[ \text{is-MULT}(opor) \rightarrow p_1+p_2+1 \]
\[ \text{is-DIV}(opor) \rightarrow N_1 \]

for: is-arithm(da-1) & is-arithm(da-2) &
(is-ADD v is-SUBTR v is-MULT v is-DIV)(opor)
8.3.2.3 The special case of integer exponentiation

The predicate is-intg-exponent tests for exponentiation with an integer constant as second operand. In this case, the function eda-intg-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

\( t \) \( \text{is-expr} \)

(104) \( \text{is-intg-exponent}(t) = \)

\( \text{is-infix-expr}(t) \ & \text{is-EXP} \ & \text{s-operator}(t) \ & \text{is-const-} \ & \text{s-op-}2(t) \ & \)

\( (\text{is-intg} \ & \text{is-bintg}) \ & \text{s-da-} \ & \text{s-op-}2(t) \)

Ref.: is-intg 8-13(54), is-bintg 8-13(55)

(105) eda-intg-exp(da-1,op-2) =

\( \text{res-c-exp-da(complete-tg(da_{tg-1},da-1),} \)

\( \text{complete-tg(da_{tg-2},da_{op-2},v_{op-2})} \)

where: \( da_{tg-1} = c\text{-exp-target-1}(mk\text{-arithm}(da-1),da_{op-2},v_{op-2}) \)

\( da_{tg-2} = target-2(da-1,da_{op-2},C\text{-EXP}) \)

Ref.: complete-tg 8-37(143), mk-arithm 8-38(149), target-2 8-22(90)
The text you provided seems to be quite technical and involves mathematical expressions and algorithms. It seems to be related to the evaluation of prefix expressions in some computational context. The document includes several equations and definitions, but without a clear context or specific question, it's hard to provide a more detailed explanation or summary. If you need help with a specific part of the content or have a particular question, I'd be happy to assist! Please let me know how I can help.
(109) \[ \text{prefix-op}(op,opor) = \text{pass-val-op}(da_{op},v); \]
\[ v:\text{prefix-val}(da_{op},v_{op},opor) \]
Ref.: \text{val-op} 8-11(41)

8.4.1 Value of the result

In the case of an arithmetic operator, a test for overflow or underflow is made. This is necessary in the case of the operator MINUS, because asymmetric representations of negative numbers may be used (cf. the representation of two-complement pictures 8.6.3,2).

(110) \[ \text{prefix-val}(da,v,opor) = \]
\[ (\text{is-PLUS } v \text{ is-MINUS}(opor) \rightarrow \text{test-fl}(da,\text{prefix-num}(da,v,opor))) \]
\[ \text{is-NOT}(opor) \rightarrow \text{PASS:prefix-not}(v) \]
Ref.: \text{test-fl} 8-17(73)

8.4.1.1 Arithmetic prefix operations

Metavariables

\[ da \quad \text{is-arithm} \]
\[ v \quad \text{is-num-val} \]
\[ opor \quad \text{is-PLUS } v \text{ is-MINUS} \]

(111) \[ \text{prefix-num}(da,v,opor) \]
Note: This function is characterized by the axiom 8-28(112).

(112) \[ (\forall da,v,opor) \]
\[ (\text{is-PLUS}(opor) \land \text{is-MINUS}(opor) \land \text{is-FIX\cdot s-scale}(da) \land v \in v-O-set(da) \supset \text{prefix-num}(da,v,opor) = \text{acc-prefix-num}(v,opor)) \]
Ref.: \text{v-O-set} 8-7(20)
Note: This axiom gives a definition of prefix-num for the case that opor is PLUS, or that opor is MINUS, da is FIXED and v is exactly repre-
sentable with da. In the remaining cases, prefix-num is an implementation-defined approximation of the mathematical operation.

\[(113) \text{acc-prefix-num}(v, opor) =
\]
\[(\text{is-PLUS}(opor) \rightarrow v, \text{is-MINUS}(opor) \rightarrow -v)\]

8.4.1.2 Negation

\[(114) \text{prefix-not}(v) = \bigcup_{i=1}^{\text{length}(v)} \text{single-prefix-not}(\text{elem}(i, v))\]
for: \text{is-bit-val-list}(v)

\[(115) \text{single-prefix-not}(v) =
\]
\[(\text{is-O-BIT}(v) \rightarrow 1\text{-BIT}, \text{is-1-BIT}(v) \rightarrow 0\text{-BIT})\]
for: \text{is-bit-val}(v)

8.4.2 Data attribute of the result

\[(116) \text{eda-prefix}(da, opor) =
\]
\[
\text{complete-tg}(\text{prefix-target}(da, opor), da)
\]
Ref.: complete-tg 8-37(143)

\[(117) \text{prefix-target}(da, opor) =
\]
\[(\text{is-PLUS} v \text{is-MINUS})(opor) \rightarrow \text{AR-DA}
\]
\[
\text{is-NOT}(opor) \rightarrow \text{BIT-DA}
\]
Ref.: AR-DA 8-13(56), BIT-DA 8-13(59)
8.5 Conversion

This chapter describes the instruction `convert(da-tg,op)` which converts an operand op to a target attribute da-tg. The result of the instruction is again an operand. 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 the third step have been described already (cf. 8.1); the second step is performed by the instruction `value-conv` defined in 8.5.1.

The target attribute da-tg may be incomplete, and first, it is completed. The syntax of incomplete target attributes and the function `complete-tg` are defined in 8.5.2.

\[
\text{convert}(da-tg,op) = \\
\text{convert-l}(\text{complete-tg}(da-tg,da_op),op)
\]

for: is-inc-da(da-tg)

\[
\text{convert-l}(da-tg,op) = \\
is-area(da_op) \text{ v } is-area(da_{tg}) \rightarrow \text{ error}
\]

\[T \rightarrow \\
\text{mk-op}(da-tg,vr) ;
\]

\[vr : \text{test-rep}(da-tg,v);
\]

\[v : \text{value-conv}(da-tg,da_{op},op-val(op))
\]

Ref.: op-val 8-11(40), test-rep 8-9(29), mk-op 8-11(39)

8.5.1 Value conversion

If da-tg and da are of the same type, the result of `value-conv(da-tg, da,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 much characters as necessary are converted (and hence may raise the CONVERSION condition).
(12C) \[
\text{value-conv}(\text{da-tg}, \text{da}, \text{v}) =
\]
\[
\text{da-type}(\text{da}) = \text{da-type}(\text{da-tg}) \quad \text{PASS: v}
\]
\[
is-\text{NUM}\cdot \text{da-type}(\text{da}) \& is-\text{CHAR}\cdot \text{da-type}(\text{da-tg}) \quad \text{num-char-conv}(\text{da}, \text{v})
\]
\[
is-\text{NUM}\cdot \text{da-type}(\text{da}) \& is-\text{BIT}\cdot \text{da-type}(\text{da-tg}) \quad \text{num-bit-conv}(\text{da}, \text{v})
\]
\[
is-\text{CHAR}\cdot \text{da-type}(\text{da}) \& is-\text{NUM}\cdot \text{da-type}(\text{da-tg}) \quad \text{char-num-conv}(\text{v})
\]
\[
is-\text{CHAR}\cdot \text{da-type}(\text{da}) \& is-\text{BIT}\cdot \text{da-type}(\text{da-tg}) \quad \text{char-bit-conv}(\text{da-tg}, \text{v})
\]
\[
is-\text{BIT}\cdot \text{da-type}(\text{da}) \& is-\text{NUM}\cdot \text{da-type}(\text{da-tg}) \quad \text{PASS: bit-num-conv}(\text{v})
\]
\[
is-\text{BIT}\cdot \text{da-type}(\text{da}) \& is-\text{CHAR}\cdot \text{da-type}(\text{da-tg}) \quad \text{PASS: bit-char-conv}(\text{v})
\]
\[
T \quad \text{error}
\]
Ref.: char-num-conv 10-61(173)

(121) \[
\text{da-type}(\text{da}) =
\]
\[
is-\text{NUM}\cdot \text{da-type}(\text{da}) \quad \text{NUM}
\]
\[
is-\text{STRING}\cdot \text{da-type}(\text{da}) \quad \text{str-base}(\text{da})
\]
\[
T \quad \text{da}
\]
Ref.: is-\text{NUM}\cdot 8-23(92), is-\text{STRING}\cdot 8-23(93), str-base 8-39(156)

8.5.1.1 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. 8.6.3). If the source attribute is arithmetic, then again a picture attribute is constructed from the arithmetic attribute, but there are differences to the ordinary picture case regarding treatment of complex values and of scale factors.

(122) \[
\text{num-char-conv}(\text{da}, \text{v}) =
\]
\[
(is-\text{arithm} \& is-\text{CPLX}\cdot s\cdot \text{mode})(\text{da}) \quad \text{pass-cplx-string}(v-\text{re}, v-\text{im}) ;
\]
\[
v-\text{re}:\text{real-char-conv}(\text{da}\_\text{real}, \text{real}(v)),
\]
\[
v-\text{im}:\text{real-char-conv}(\text{da}\_\text{real}, \text{imag}(v))
\]
\[
is-\text{arithm} \& is-\text{REAL}\cdot s\cdot \text{mode})(\text{da}) \quad \text{real-char-conv}(\text{da}, \text{v})
\]
\[
is-\text{NUM}\cdot \text{pic}(\text{da}) \quad \text{num-pic-char-conv}(\text{da}, \text{v})
\]

8.5.1.1 cont'd
for: is-num-type(da) & is-num-val(v)

Ref.: is-num-pic 8-44(166), num-pic-char-conv 8-59(226)

(123) cplx-string(v-re,v-im) =

\[
\text{v-re} \text{ sign-and-left-adjust(v-im)} ^{\langle 1-\text{CHAR}\rangle}
\]

for: is-char-val-list(v-re) & is-char-val-list(v-im)

(124) sign-and-left-adjust(v) =

is-BLANK*elem(1,v) & (is-BLANK v is-PLUS v is-MINUS)*elem(2,v) \rightarrow

sign-and-left-adjust(tail(v)) ^ {\langle \text{BLANK}\rangle}

for: is-char-val-list(v)

Note: The function moves leading blanks to the end and, if no sign was present, signs the string with PLUS.

(125) real-char-conv(da,v) =

is-BIN*s-base(da) \rightarrow

real-char-conv(complete-tg(μ(AR-DA;<s-base:DEC>),da),v)

is-FIX*s-scale(da) \& Op_{da} \rightarrow

num-pic-char-conv(out-fix-pic(p_{da}+3,q_{da}),v)

is-FIX*s-scale(da) \rightarrow

pass-conv-scale-f(v-str,q_{da});

v-str: num-pic-char-conv(out-fix-pic(p_{da}+1,0),v.10q_{da})

is-FLT*s-scale(da) \rightarrow

num-pic-char-conv(out-flt-pic(p_{da}+2),v)

for: (is-arithm & is-REAL*s-mode)(da) & is-real-val(v)

Ref.: num-pic-char-conv 8-32(125), out-fix-pic 8-60(227), out-flt-pic 8-60(229)

complete-tg 8-37(143), AR-DA 8-13(56)

8.5.1.1
(126) \[ \text{conv-scale-f}(v, m) = \]
\[ v \cdot (\text{F-CHAR}) \cdot (\text{sign-char}(-m)) \cdot (\text{num-char-list}(\text{lgth}_m, \text{abs}(m))) \]

where: \( \text{lgth}_m = (i_1)(10^{i_1} \leq \text{abs}(m) < 10^{i_1}) \)

for: \( \text{is-char-val-list}(v) \)

Ref.: sign-char 8-53(203)

(127) \[ \text{num-char-list}(n, v) = \]
\[ n = 0 \rightarrow \]
\[ T \rightarrow \text{num-char-list}(n-1, \text{trunc}(v/10)) \cdot (\text{single-num-char}(\text{modulo}(v, 10))) \]
for: \( \text{is-intg-val}(v) \) & \( n \geq 0 \)

(128) \[ \text{single-num-char}(v) = \]
\[ v = 0 \rightarrow 0-\text{CHAR} \]
\[ v = 1 \rightarrow 1-\text{CHAR} \]
\[ v = 2 \rightarrow 2-\text{CHAR} \]
\[ v = 3 \rightarrow 3-\text{CHAR} \]
\[ v = 4 \rightarrow 4-\text{CHAR} \]
\[ v = 5 \rightarrow 5-\text{CHAR} \]
\[ v = 6 \rightarrow 6-\text{CHAR} \]
\[ v = 7 \rightarrow 7-\text{CHAR} \]
\[ v = 8 \rightarrow 8-\text{CHAR} \]
\[ v = 9 \rightarrow 9-\text{CHAR} \]
for: \( \text{is-intg-val}(v) \) & \( \text{osvs9} \)

8.5.1.2 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.
(129) \( \text{num-bit-conv}(\text{da},v) = \)

\[
l \text{lgth}_d < 0 \quad \text{PASS: <>}
\]

\[
T \rightarrow \text{bintg-bit-conv}(\text{lgth}_d,vr);
vr: \text{test-rep}(\text{bintg-da}(\text{lgth}_d),v)
\]

where: \( \text{lgth}_d = \text{bit-length}(\text{mk-arithm}(\text{da})) \)
for: \( \text{is-num-type}(\text{da}) \land \text{is-num-val}(v) \)

Ref.: test-rep 8-9(29), bintg-da 8-12(51), bit-length 8-39(154), mk-arithm 8-38(149)

(130) \( \text{bintg-bit-conv}(n,v_r) = \)

\[
\text{PASS: num-bit-list}(n,\text{abs-value}(\text{bintg-da}(n),v_r))
\]

Ref.: bintg-da 8-12(51)

(131) \( \text{num-bit-list}(n,v) = \)

\[
\begin{align*}
n & = 0 & \rightarrow & < > \\
T & \rightarrow \text{num-bit-list}(n-1,\text{trunc}(v/2)) \land < \text{single-num-bit}(\text{modulo}(v,2))>
\end{align*}
\]

for: \( \text{is-real-val}(v) \land n \geq 0 \)

(132) \( \text{single-num-bit}(v) = \)

\[
\begin{align*}
(v & = 0 \rightarrow 0\text{-BIT}, \quad v = 1 \rightarrow 1\text{-BIT})
\end{align*}
\]

for: \( v = 0 \lor v = 1 \)

8.5.1.3 Character to numeric conversion

The instruction \( \text{char-num-conv}(v) \) is defined in chapter 10, because it uses methods for parsing strings that are developed there (cf. 10-60(162)). The corresponding function for the case of a single character is \( \text{single-char-num}(v) \).

(133) \( \text{single-char-num}(v) = \)

\[
(ln)(v = \text{single-num-char}(n))
\]

for: \( \text{is-char-val}(v) \)

Ref.: single-num-char 8-33(128)
8.5.1.4 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.

\[(134) \text{char-bit-conv}(\text{da-tg}, v) =\]

\[
\begin{align*}
\text{PASS: LIST single-char-bit}(\text{elem}(i, v)) & \\
\text{T} \rightarrow \text{char-bit-conv}(\text{da-tg}, \text{corr-v}); & \quad \text{corr-v: call-conv-cond}(v, i, 0, 606)
\end{align*}
\]

where: \[\text{lgth}_{\text{tg}} = \min(\text{length}(v), \text{str-length}(\text{da-tg}))\]

\[i_0 = (i)(\text{is-0-CHAR } v \text{ is-1-CHAR} \cdot \text{elem}(i, v) \quad \forall j)(\text{is-0-CHAR } v \text{ is-1-CHAR} \cdot \text{elem}(j, v))\]

for: \((\text{is-string} \& \text{is-BIT-s-base}(\text{da-tg}) \& \text{is-char-val-list}(v))\)

Ref.: \text{call-conv-cond} 9-17(34), \text{str-length} 8-40(157)

\[(135) \text{single-char-bit}(v) =\]

\[(\text{is-0-CHAR}(v) \rightarrow 0, \text{is-1-CHAR}(v) \rightarrow 1)\]

for: \((\text{is-0-CHAR } v \text{ is-1-CHAR})(v)\)

8.5.1.5 Bit to numeric conversion

\[(136) \text{bit-num-conv}(v) = \sum_{i=1}^{\text{length}(v)} \text{single-bit-num}(\text{elem}(i, v)).2^i(\text{length}(v)-i)\]

for: \text{is-bit-val-list}(v)

\[(137) \text{single-bit-num}(v) =\]

\[(\text{is-0-BIT}(v) \rightarrow 0, \text{is-1-BIT}(v) \rightarrow 1)\]

for: \text{is-bit-val}(v)\]
8.5.1.6 Bit to character conversion

(138) \text{bit-char-conv}(v) = 
\[ \text{is-char-list}(\text{single-bit-char}(\text{elem}(i,v))) \]
for: \text{is-bit-val-list}(v)

(139) \text{single-bit-char}(v) = 
\[ (\text{is-0-bit}(v) \rightarrow \text{0-CHAR}, \text{is-1-BIT}(v) \rightarrow \text{1-CHAR}) \]
for: \text{is-bit-val}(v)

8.5.2 Completion of target attributes

The predicate \text{is-inc-da} characterizes a class of "incomplete data attributes" that differs from the class \text{is-val-da} (cf. 8-3(1)) in that any component of an arithmetic or string attribute may be replaced by *. The function \text{complete-tg}(\text{da-tg}, \text{da}) completes \text{da-tg} with the aid of \text{da}. If \text{da-tg} and \text{da} are of different types, then \text{da} is first transformed to the appropriate type by one of the functions \text{mk-arith}(\text{da}), \text{mk-string}(\text{da}), \text{mk-char-string}(\text{da}), \text{mk-bit-string}(\text{da}).

\[ \text{Metavariablen} \]
\[ \text{da-tg} \quad \text{is-inc-da} \quad \text{target data attribute} \]

(140) \text{is-inc-da} = \text{is-inc-arithm} \lor \text{is-inc-string} \lor \text{is-pic} \lor \text{is-elem-1-da} \lor \text{is-area}

Ref.: \text{is-elem-1-da} 8-3(3)

(141) \text{is-inc-arithm} = (\langle \text{s-node}: \text{is-REAL} \lor \text{is-CPLX} \lor \text{is-*}, \\
\langle \text{s-base}: \text{is-BIN} \lor \text{is-DEC} \lor \text{is-*}, \\
\langle \text{s-scale}: \text{is-FIX} \lor \text{is-FLT} \lor \text{is-*}, \\
\langle \text{s-orec}: \text{is-intg-val} \lor \text{is-O} \lor \text{is-*}, \\
\langle \text{s-scale-f}: \text{is-intg-val} \lor \text{is-O} \lor \text{is-*} \rangle)

(142) \text{is-inc-string} = (\langle \text{s-base}: \text{is-BIT} \lor \text{is-CHAR} \lor \text{is-*}, \\
\langle \text{s-length}: \text{is-intg-val} \lor \text{is-*}, \\
\langle \text{s-variable}: \text{is-opt} \lor \text{is-*} \rangle)
(143) \( \text{complete-tg} (\text{da-tg}, \text{da}) = \)
\[
\begin{align*}
\text{is-inc-arithm}(\text{da-tg}) &\quad \text{complete-ar}(\text{da-tg}, \text{mk-arithm}(\text{da})) \\
\text{is-inc-string}(\text{da-tg}) &\quad \text{complete-str}(\text{da-tg}, \text{(is-CHAR-s-base}(\text{da-tg}) &\quad \text{mk-char-string}(\text{da}) , \\
\text{is-BIT-s-base}(\text{da-tg}) &\quad \text{mk-bit-string}(\text{da}), \\
T &\quad \text{mk-string}(\text{da})) \)
\end{align*}
\]

8.5.2.1 Completion of arithmetic attributes

Metavariables

\[
\begin{align*}
\text{da-tg} &\quad \text{is-inc-arithm} \\
\text{da} &\quad \text{is-arithm}
\end{align*}
\]

(144) \( \text{complete-ar}(\text{da-tg}, \text{da}) = \)
\[
\text{complete-prec}(\text{complete-m-b-s}(\text{da-tg}, \text{da}), \text{da})
\]

(145) \( \text{complete-m-b-s}(\text{da-tg}, \text{da}) = \)
\[
\mu(\text{da-tg}, \{ \chi; \chi(\text{da}) > | \chi \in \{ \text{s-mode}, \text{s-base}, \text{s-scale} \} \& \text{is-***}(\text{da-tg}) \})
\]

(146) \( \text{complete-prec}(\text{da-tg}, \text{da}) = \)
\[
\begin{align*}
\text{is-**-s-prec}(\text{da-tg}) &\quad \text{is-FIX-s-scale}(\text{da-tg}) \\
\mu(\text{da-tg}, \{ s\text{-precmin}(T_{\text{tg}}, f_{\text{prec}} = 1 \rightarrow \text{p}_{\text{da}}, T \rightarrow 1 + \text{ceil}(\text{p}_{\text{da}} \cdot f_{\text{prec}})), \\
\text<s-scale-f:sign}(\text{q}_{\text{da}}), \text{ceil}(\text{abs}(\text{q}_{\text{da}}), \text{f}_{\text{prec}})) \}
\end{align*}
\]

(147) \( \text{is-**-s-prec}(\text{da-tg}) \& \text{is-FLT-s-scale}(\text{da-tg}) \\
\mu(\text{da-tg}, \{ s\text{-precmin}(T_{\text{tg}}, \text{ceil}(\text{q}_{\text{da}} \cdot f_{\text{prec}})) \})
\]

T \( \rightarrow \text{da-tg} \)

where: \( T_{\text{tg}} = \text{max-prec}(\text{da-tg}) \)
\( f_{\text{prec}} = \text{prec-fact}(\text{da-tg}, \text{da}) \)

Ref.: max-prec 8-13(62)
(147) \( \text{prec-fact}(\text{da-tg}, \text{da}) = \)
\[
\begin{align*}
\text{s-base}(\text{da}) &= \text{s-base}(\text{da-tg}) \quad \rightarrow \ l \\
\text{is-BIN}\cdot\text{s-base}(\text{da}) \& \text{is-DEC}\cdot\text{s-base}(\text{da-tg}) &= \rightarrow 1,3,32 \\
\text{is-DEC}\cdot\text{s-base}(\text{da}) \& \text{is-BIN}\cdot\text{s-base}(\text{da-tg}) &= \rightarrow 3,32
\end{align*}
\]

8.5.2.2 Completion of string attributes

(148) \( \text{complete-str}(\text{da-tg}, \text{da}) = \)
\[
\mu(\text{da-tg}; \{< \chi; \chi(\text{da}) > \mid \chi \in \{\text{s-base}, \text{s-length}, \text{s-varying}\} \& \text{is-}\star; \chi(\text{da-tg})}\})
\]
for: is-inc-string(\text{da-tg}) & is-string(\text{da})

8.5.2.3 Transformation into an arithmetic attribute

(149) \( \text{mk-arithm}(\text{da}) = \)
\[
\begin{align*}
\text{is-arithm}(\text{da}) &= \rightarrow \text{da} \\
\text{is-num-pic}(\text{da}) &= \rightarrow \text{mk-arithm-pic}(\text{da}) \\
\text{is-string-type}(\text{da}) &= \rightarrow \\
(\text{str-base}(\text{da}) &= \text{CHAR} \rightarrow \text{intg-da}(\text{max-prec}(\text{INTG-DA})), \\
\text{str-base}(\text{da}) &= \text{BIT} \rightarrow \text{bintg-da}(\text{max-prec}(\text{BINTG-DA})))
\end{align*}
\]
\( T \rightarrow \text{error} \)

Ref.: is-num-pic 8-44(166), mk-arithm-pic 8-64(239), is-string-type 8-23(93)
\( \text{str-base} 8-39(156), \text{intg-da} 8-12(50), \text{bintg-da} 8-12(51), \)
\( \text{max-prec} 8-13(62), \text{INTG-DA} 8-12(52), \text{BINTG-DA} 8-12(53) \)

8.5.2.4 Transformation into a string attribute

(150) \( \text{mk-string}(\text{da}) = \)
\[
\begin{align*}
\neg(\text{is-arithm} \lor \text{is-string} \lor \text{is-pic})(\text{da}) &= \rightarrow \text{error} \\
\text{str-base}(\text{da}) &= \text{CHAR} \rightarrow \text{mk-char-string}(\text{da}) \\
\text{str-base}(\text{da}) &= \text{BIT} \rightarrow \text{mk-bit-string}(\text{da})
\end{align*}
\]

(151) \( \text{mk-char-string}(\text{da}) = \)
\[
\begin{align*}
\text{is-arithm}(\text{da}) &= \mu_0(<\text{s-base}:\text{CHAR}, \text{s-length}:\text{char-length}(\text{da})>) \\
(\text{is-string} \lor \text{is-pic})(\text{da}) &= \mu(\text{mk-str-l}(\text{da}); <\text{s-base}:\text{CHAR}>) \\
T &= \rightarrow \text{error}
\end{align*}
\]

3.5.2.4
(152) char-length(da) =

\[
\text{is-BIN}\cdot s\cdot \text{base}(da) \rightarrow \text{char-length}(\text{complete-tg}(\mu(\text{AR-DA}; s\cdot \text{base}; \text{DEC})), da))
\]

\[
\text{is-CPLX}\cdot s\cdot \text{mode}(da) \rightarrow 2\cdot \text{char-length}(d_{\text{real}})
\]

\[
\text{is-PLT}\cdot s\cdot \text{scale}(da) \rightarrow P_{da} + 4 + \text{EXP}\cdot \text{SIZE}
\]

\[
\text{is-FIX}\cdot s\cdot \text{scale}(da) \rightarrow \\
(0_{s_{da}}^{P_{da}} \rightarrow P_{da} + 3, T \rightarrow P_{da} + 3 + \text{lgth}_{q})
\]

where: \(\text{lgth}_{q} = (i)(10^i(i - 1) < \text{abs}(\sigma_{da}) < 10^i)\)

for: \(\text{is-arithum}(da)\)

Ref.: complete-tg 8-37(143), AR-DA 8-13(56)

Note: cf. numeric to character conversion 8.5.1.1 and 8.6.3.5.

(153) mk-bit-string(da) =

\[
\text{is-num-type}(da) \rightarrow \mu_{q}(s\cdot \text{base}; \text{BIT}, s\cdot \text{length}; \text{bit-length}(\text{mk-arithum}(da)))
\]

\[
\text{is-string-type}(da) \rightarrow \mu(mk\cdot \text{str-1}(da); s\cdot \text{base}; \text{BIT})
\]

\(T \rightarrow \text{error}\)

Ref.: is-num-type 8-23(92), is-string-type 8-23(93), mk-arithum 8-38(149)

(154) bit-length(da) =

\[
\max(0, \min(1, \text{ceil}(\text{lgth}_{da}, f_{\text{prec}})))
\]

where: \(\text{lgth}_{da} = \max\cdot \text{prec}(\text{BINTG-DA})\)

\[
\text{lgth}_{da} = (\text{is-FIX}\cdot s\cdot \text{scale}(da) \rightarrow P_{da} - q_{da}, T \rightarrow P_{da})
\]

\(f_{\text{prec}} = \text{prec-fact}(\text{BINTG-DA}, da)\)

for: \(\text{is-arithum}(da)\)

Ref.: max-prec 8-13(62), prec-fact 8-38(147), BINTG-DA 8-12(53)

(155) mk-str-1(da) =

\[
\text{is-string}(da) \rightarrow da
\]

\[
\text{is-pic}(da) \rightarrow \mu_{q}(s\cdot \text{base}; \text{str-base}(da), s\cdot \text{length}; \text{str-length}(da))
\]

for: \(\text{is-string} \lor \text{is-pic}(da)\)

(156) str-base(da) =

\[
\text{is-string}(da) \rightarrow s\cdot \text{base}(da)
\]

\[
(\text{is-arithum} \& \text{is-BIN}\cdot s\cdot \text{base}(da) \lor \text{is-bin-pic}(da) \rightarrow \text{BIT}
\]

\(T \rightarrow \text{CHAR}\)

for: \(\text{is-arithum} \lor \text{is-string} \lor \text{is-pic}(da)\)
\[(157) \text{str-length}(\text{da}) = \]
\[\text{is-string}(\text{da}) \rightarrow \text{s-length}(\text{da})\]
\[\text{is-pic}(\text{da}) \rightarrow \text{pic-length}(\text{da})\]

for: \((\text{is-string} \lor \text{is-pic})(\text{da})\)

Ref.: \text{pic-length} 8-64(241)

### 3.6 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 8.6.1, the predicate is-proc-pic(\text{da}) in 8.6.2, the instructions ren-pic(\text{da}, \text{v}) and num-pic-char-conv(\text{da}, \text{v}) in 8.6.3, the function pic-val(\text{da}, \text{vr}) and the predicate is-pic-match(\text{da}, \text{v}) in 8.6.4, finally certain functions for the transformation of picture attributes into other attributes in 3.6.5.

The auxiliary functions and predicates defined in 8.6.1 are used in general without explicit reference.

**Metavariables**

- \text{da}: is-x-pic
- \text{sf} \in \{\text{sf}_{\text{mt}}, \text{sf}_{\text{exp}}\}
- \text{spec}: is-pic-spec
- \text{pv}: \text{is-int\-val\-list}
- \text{sgn}: \text{sgn} \in \{+1, -1\}
- \text{sg}: \text{sg} \in \{\text{PLUS}, \text{MINUS}\}

**Abbreviations**

- \text{sf}_{\text{mt}} = \text{s-mt-field}
- \text{sf}_{\text{exp}} = \text{s-exp-field}
- \text{lgth}_{\text{da}} = \text{pic-length}(\text{ppic}(\text{da}))
- \text{lgth}_{\text{sf}} = \text{length}(\text{sf}(\text{ppic}(\text{da}))

for \text{is-xnum-pic}(\text{da}):
\textbf{IBI-1 LAB VIENNA}

\[ \text{\texttt{lgth}}_\text{mt} = \text{length of mantissa subfield} \]
\[ \text{\texttt{u}}_\text{da} = \text{unit position of mantissa subfield} \]
\[ \text{\texttt{q}}_\text{da} = \text{explicit scale-factor of da} \]
\[ \text{\texttt{T}} \longrightarrow \text{s-scale-f(}\text{da}\text{)} \]
\[ \text{\texttt{P}}_\text{da} = \text{precision of da} \]
\[ \text{\texttt{q}}_\text{da} = \text{number of digit positions behind binary or decimal point} \]
\[ \text{\texttt{T}} \longrightarrow \text{n-of-d(}\text{da, sf, \text{lgth}}_\text{mt}, \text{u}_\text{da}\text{)} \]
\[ \text{\texttt{N}}_\text{da} = \text{maximum precision associated with da} \]
\[ \text{\texttt{nd}}_\text{sf} = \text{number of digit positions in a subfield} \]
\[ \text{\texttt{nd}}_\text{mt} = \text{number of digit positions in mantissa subfield} \]
\[ \text{\texttt{base}}_\text{da} = \text{number base of da} \]
\[ \text{\texttt{T}} \longrightarrow \text{n-of-d-subf(}\text{da, sf, \text{nd}}_\text{mt}\text{)} \]

for \texttt{is-xnum-pic(d)} & \texttt{is-CPLX-s-mode(d)}:

\[ \text{\texttt{da}}_{\text{real}} = \mu(\text{da}; \langle \text{s-mode}; \text{REAL} \rangle) \]

real data attribute corresponding to a complex da

for \texttt{(is-xdec-pic v is-xsterling-pic)(da)}:

\[ \text{\texttt{d}}_i = \text{i-th digit position of a subfield} \]
\[ \text{\texttt{d}}_{i+1} = \text{(i+1)-th digit position of a subfield} \]
\[ \text{\texttt{d}}_{i-1} = \text{(i-1)-th digit position of a subfield} \]
\[ \text{\texttt{d}}_1 = \text{first digit position of a subfield} \]
\[ \text{\texttt{d}}_{\text{nd}} = \text{last digit position of a subfield} \]
\[ \text{\texttt{i}}_u = \text{index of unit digit position} \]
\[ \text{\texttt{beg}}_{\text{dr}} = \text{drift begin for a subfield} \]
\[ \text{\texttt{end}}_{\text{dr}} = \text{drift end for a subfield} \]
\[ \text{\texttt{spec}}_{\text{dr}} = \text{drifting specification for a subfield} \]

for \texttt{is-x-sterling-pic(d)}:

\[ \text{\texttt{end}}_{\text{st}} = \text{end of static part} \]
\[ \text{\texttt{end}}_{\text{pd}} = \text{end of pounds field} \]
\[ \text{\texttt{end}}_{\text{sh}} = \text{end of shillings field} \]
\[ \text{\texttt{end}}_{\text{ipc}} = \text{end of integral pence field} \]
\[ \text{\texttt{nd}}_{\text{pd}} = \text{number of digits in pounds field} \]
8.6.1 Auxiliary definitions

The class of "explicit" picture attributes is introduced in 8.6.1.1. Predicates for the classification of picture attributes and picture specifications (i.e., characters in the picture attribute) are listed in 8.6.1.2.

8.6.1.1 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 specification) 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.

8.6.1.1.1 Syntax of explicit picture attributes

The syntax of explicit picture attributes is defined by the predicate is-xpic.

Metavariable

x arbitrary object

(158) is-xpic(x) =

is-pic(x) v (is-dec-pic v is-sterling-pic) opic(x) &

\[ \bigvee_{s \in \text{Subfields}(x)} (\text{is-int-val}s-dr-beg \& \text{is-int-val}s-dr-end \& \text{is-dr-spec}s-dr-spec) \& sf(x) \]

Ref.: is-dr-spec 8-45(179)
8.6.1.1.2 Transformation of an explicit picture into a picture

The picture attribute (or "implicit" picture attribute) corresponding to an explicit picture attribute da is obtained by the function mpic(da).

\[
\text{(162) } \text{mpic}(\text{da}) =
\begin{align*}
&\text{(is-char-pic } \vee \text{ is-bin-pic)}(\text{da}) \rightarrow \text{da} \\
&\text{T} \rightarrow \text{mpic-1(}\text{da})
\end{align*}
\]

\[
\text{(163) } \text{mpic-1}(\text{da}) =
\begin{align*}
&\text{is-fix-pic}(\text{da}) \rightarrow \text{msubf}(\text{da}, \text{s}_{\text{f}}_{\text{mt}}) \\
&\text{is-flt-pic}(\text{da}) \rightarrow \text{msubf}(\text{msubf}(\text{da}, \text{s}_{\text{f}}_{\text{mt}}), \text{s}_{\text{f}}_{\text{exp}})
\end{align*}
\]

for: (is-xdec-pic } \vee \text{ is-xsterling-pic}(\text{da})

Ref.: is-fix-pic 8-44(170), is-flt-pic 8-44(171)

\[
\text{(164) } \text{msubf}(\text{da}, \text{s}) =
\begin{align*}
&\text{is-Ω}(\text{spec}_{\text{dr}}) \rightarrow \text{da} \\
&T \rightarrow \mu(\text{ppic}(\text{da}); \langle (\text{colm}(d_{i}))_{\text{s}}{\text{f}}; \text{spec}_{\text{dr}} \mid 1 \text{isend}_{\text{s}}{\text{f}} \& \text{beg}_{\text{dr}} \cdot d_{i} \text{send}_{\text{dr}} \& \neg \text{is-Y-CHAR} \cdot \text{colm}(d_{i}, \text{s}_{\text{f}}(\text{da}))})
\end{align*}
\]

for: (is-xdec-pic } \vee \text{is-xsterling-pic}(\text{da})

Note: The digit positions of the subfield between \text{beg}_{\text{dr}} and \text{end}_{\text{dr}} that are different from \text{Y-CHAR} are replaced by \text{spec}_{\text{dr}}, and the explicit drifting information is deleted.
8.6.1.1.3 Transformation of a picture into an explicit picture

The explicit picture attribute corresponding to a picture attribute da is obtained by the function xpic(da). The definition, in all but the trivial cases, is implicit and interferes with that of is-prop-pic-da (cf. 8.6.2).

(165) \[
\text{xpic}(\text{da}) = (\text{is-char-pic } \lor \text{is-bin-pic}) (\text{da}) \quad \text{for: } \text{is-prop-pic}(\text{da})
\]

(166) \[
\text{Ref.: is-prop-pic-l 8-47(188)}
\]

8.6.1.2 Classifications 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.

8.6.1.2.1 Predicates classifying pictures

(166) \[
\text{is-num-pic} = \text{is-bin-pic } \lor \text{is-dec-pic } \lor \text{is-sterling-pic}
\]

(167) \[
\text{is-xnum-pic} = \text{is-bin-pic } \lor \text{is-xdec-pic } \lor \text{is-xsterling-pic}
\]

(168) \[
\text{is-xdec-pic}(\text{da}) = \text{is-dec-pic}(\text{ppic}(\text{da}))
\]

(169) \[
\text{is-xsterling-pic}(\text{da}) = \text{is-sterling-pic}(\text{ppic}(\text{da}))
\]

(170) \[
\text{is-fix-pic}(\text{da}) = \neg \text{is-}\bar{Q}\cdot s_{\exp}(\text{da}) \land \text{is-}\bar{Q}\cdot s_{\exp-sep}(\text{da}) \quad \text{for: } \text{is-xnum-pic}(\text{da})
\]

(171) \[
\text{is-flt-pic}(\text{da}) = \neg \text{is-}\bar{Q}\cdot s_{\exp}(\text{da}) \land \text{is-}\bar{Q}\cdot s_{\scale}(\text{da}) \quad \text{for: } \text{is-xnum-pic}(\text{da})
\]
8.6.1.2.2 Predicates classifying picture specifications

The following predicates serve to classify characters occurring within picture attributes. Where the classification is context dependent, the arguments to these predicates have to be positions, i.e. selectors, rather than characters.

Metavariable

\[ da \quad \text{is-xnum-pic} \quad \text{explicit numeric picture attribute} \]

(172) \[ \text{is-pic-spec} = \text{is-char-spec} \lor \text{is-bin-spec} \lor \text{is-sterling-spec} \]

(173) \[ \text{is-dig-spec}(da,sf,i) = \]
\[ \text{is-dec-dig-spec}(da,sf,i) \lor (\text{is-3-CHAR} \lor \text{is-7-CHAR} \lor \text{is-6-CHAR}) \cdot \text{elem}(i,sf,da) \]

(174) \[ \text{is-dec-dig-spec}(da,sf,i) = \]
\[ \text{is-9-CHAR} \lor \text{is-Y-CHAR} \lor \text{is-T-CHAR} \lor \text{is-I-CHAR}) \cdot \text{elem}(i,sf,da) \lor \text{is-r-char}(da,sf,i) \]

(175) \[ \text{is-r-char}(da,sf,i) = \]
\[ \text{is-R-CHAR} \cdot \text{elem}(i,sf(da)) \land \neg \text{is-C-CHAR} \cdot \text{elem}(i-1,sf(da)) \]

(176) \[ \text{is-sign-spec}(da,sf,i) = \]
\[ (\text{is-SIGN} \lor \text{is-PLUS} \lor \text{is-MINUS} \lor \text{is-T-CHAR} \lor \text{is-I-CHAR}) \cdot \text{elem}(i,sf(da)) \lor \text{is-r-char}(da,sf,i) \]

(177) \[ \text{is-trail-sign}(da,sf,i) = \]
\[ \text{is-C-CHAR} \cdot \text{elem}(i,sf(da)) \land \text{is-R-CHAR} \cdot \text{elem}(i+1,sf(da)) \lor \text{is-D-CHAR} \cdot \text{elem}(i,sf(da)) \land \text{is-B-CHAR} \cdot \text{elem}(i+1,sf(da)) \]

(178) \[ \text{is-sign-pos}(da,sf,\lambda) = \]
\[ (\exists i)(\lambda = (\text{elem}(i)) \cdot sf \land \text{is-sign-spec}(da,sf,i)) \lor \]
\[ sf = \text{sf}^{\text{mt}} \land (\exists sf-1,i)(\lambda = (\text{elem}(i)) \cdot sf-1 \land \text{is-trail-sign}(da,sf-1,i)) \]

Note: A trailing sign CR or DB is interpreted as a sign position of the mantissa subfield, even if it appears within the exponent subfield.

(179) \[ \text{is-dr-spec} = \]
\[ \text{is-Z-CHAR} \lor \text{is-ASTER} \lor \text{is-dr-ed-spec} \]
(180) \text{is-dr-ed-spec} =
   \text{is-SIGN} \lor \text{is-PLUS} \lor \text{is-MINUS} \lor \text{is-DOLLAR}

(181) \text{is-ed-spec(da, sf, i)} =
   (\text{is-BLANK} \lor \text{is-POINT} \lor \text{is-COMMA} \lor \text{is-SLASH} \lor \text{is-S-CHAR}) \cdot \text{elem}(i, sf(da)) \lor
   \text{is-d-char(da, sf, i)}

(182) \text{is-d-char(da, sf, i)} =
   \text{is-D-CHAR} \cdot \text{elem}(i, sf(da)) \land \neg \text{is-B-CHAR} \cdot \text{elem}(i+1, sf(da))

8.6.1.2.3 Functions computing and counting digit positions

\text{dig-pos(da, sf, i)} \text{ yields } d_i \text{, the } i\text{-th digit position of the subfield } sf(da)\text{; }\text{n-of-d-subf(da, sf)} \text{ yields the number of digit positions in the subfield } sf(da)\text{; }\text{n-of-d(da, sf, m, n)} \text{ the number of those positions between } m \text{ and } n; \text{dig-index(da, sf, n)} \text{ yields the index of the rightmost digit position that precedes or equals position } n.

(183) \text{dig-pos(da, sf, i)} =
   i = 0 \rightarrow 0
   \begin{align*}
   T &\quad (\forall n)(n > d_i - 1 \land (n = \text{length}_{sf} + 1 \lor \text{is-dig-spec(da, sf, n)})) \land
      (\forall j)(d_i - 1 < j < n \lor \neg \text{is-dig-spec(da, sf, j)})
   \end{align*}
   \text{for: (is-xdec-pic } \lor \text{is-xsterling-pic(da) } \land \text{is-xdec-pic(da) })

(184) \text{n-of-d-subf(da, sf)} =
   \begin{align*}
   \text{is-bin-pic(da) } &\rightarrow \text{is-sign-mag-pic(da) } \land \neg \text{is-SIGN-elem(1, sf(da)) } \rightarrow \text{length}_{sf}
   \end{align*}
   \begin{align*}
   T &\rightarrow \text{length}_{sf} - 1
   T &\rightarrow \text{n-of-d(da, sf, i, length}_{sf})
   \end{align*}
   \text{for: is-xnum-pic(da) }
   \text{Ref.: is-sign-mag-pic 8-48(189)}
8.6.2 Checking of picture attributes

The class is-pic defined by the abstract syntax is much wider than the class of actually permitted picture attributes; the latter is characterized by the predicate is-prop-pic.

(187) is-prop-pic(da) =

\[ \text{is-char-pic}(da) \& \text{lgth}_{da} > 0 \lor \]
\[ \text{is-bin-pic}(da) \& (\text{is-sign-mag-pic} \lor \text{is-2-compl-pic} \lor \text{is-1-compl-pic})(da) \lor \]
\[ \text{is-prop-pic-l}(da) \lor \]
\[ \text{(is-dec-pic} \lor \text{is-sterling-pic})(da) \& (\exists da-1)(\text{is-prop-pic-l}(da-1) \& \]
\[ da = \text{mpic-1}(da-1)) \]

for: is-pic(da)

(188) is-prop-pic-l(da) =

\[ (\text{is-fix-pic} \lor \text{is-flt-pic})(da) \& (\text{is-\ensuremath{\mathcal{Q}}(u_{da})} \lor \text{u}_{da} \leq \text{lgth}_{mt}) \& \]
\[ \exists s_{f} \exists s_{f-1}(da) \]

for: is-xnum-pic(\*da)
8.6.2.1 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

da is-bin-pic

(189) is-sign-mag-pic(da) =

\[ \bigvee_{s \in sf(set(da))} \text{is-1-CHAR-list} \land \text{is-SIGN-head} \land \text{is-1-CHAR-list-tail} \cdot sf(da) \]

(190) is-2-compl-pic(da) = \[ \bigvee_{s \in sf(set(da))} \text{is-2-CHAR-list} \cdot sf(da) \]

(191) is-1-compl-pic(da) = \[ \bigvee_{s \in sf(set(da))} \text{is-3-CHAR-list} \cdot sf(da) \]

8.6.2.2 Proper subfields

(192) is-prop-subf(da,sf) =

\[ nd_{sf} > 0 \land ((\text{is-xdec-pic} \lor \text{is-xsterling-pic})(da) \lor \begin{align*}
& (\forall i)(\neg (\text{is-2-CHAR} \lor \text{is-ASTER}) \cdot \text{elem}(i,sf(da))) \land \\
& (\exists x-1,x-2)(x-1 \neq x-2 \land \text{is-sign-pos}(da,sf,x-1) \land \\
& \text{is-sign-pos}(da,sf,x-2)) \land \\
& \neg (\exists i,j)(i \neq j \land \text{is-DOLLAR} \cdot \text{elem}(i,sf(da)) \land \text{is-DOLLAR} \cdot \text{elem}(j,sf(da))) \land \\
& (\forall i)(\text{is-trail-sign}(da,sf,i) \supset i > d_{nd} \land (\text{is-flt-pic}(da) \lor sf = sf_{exp}))) \land \\
& (\forall i)(\text{is-dr-ed-spec} \cdot \text{elem}(i,sf(da)) \supset \neg (d_1 < i < d_{nd})) \land \\
& (\text{is-xdec-pic}(da) \supset \\
& (\forall i)((\text{is-C-CHAR} \lor \text{is-D-CHAR}) \cdot \text{elem}(i,sf(da)) \lor \\
& \text{is-B-CHAR} \cdot \text{elem}(i+1,sf(da)) \supset \text{is-trail-sign}(da,sf,i))) \land \\
& (\text{is-xsterling-pic}(da) \supset \text{is-prop-sterling-subf}(da,sf)) \land \\
& (\neg \text{is-\text{Q}(spec_{dr}} \supset \text{is-prop-drift}(da,sf)))
\]

for: is-xnum-pic(da)

Ref.: is-prop-sterling-subf 8-49(193), is-prop-drift 8-50(195)
8.6.2.3 Additional restrictions for sterling subfields

**Metavariabes**

da         is-x-stelring-nic  
sf         sf = sf_{mt}  

sterling pictures have no exponent field

(193) is-prop-stelring-subf(da,sf) =

\[
\begin{align*}
0 \leq \text{end} &< \text{end}_{pd} < \text{end}_{sh} < \text{end}_{ipc} \leq \lgh_{sf} \& \text{nd}_{pd} > 0 \& \\
\text{end}_{pdo} &\in \{i \in \{1, \ldots, n\}^{\text{end}_{pdo} + 1} \\
&\text{is-blank} \lor \text{is-point} \lor \text{is-slash} \lor \text{is-s-char} \}\cdot \text{elem}(i, sf(da)) \& \\
&\text{is-prop-sh-pc}(da, sf, n, \text{end}_{sh} ) \& \\
(\exists n) (\text{end}_{pd} < n < \text{end}_{ipc} \& \\
\text{end}_{ipc} < \text{end}_{nd} \& \\
\text{is-q}(u_{da}) \& \text{end}_{ipc} \in \{i \in \{1, \ldots, n\}^{\text{end}_{ipc} + 1} \\
&\text{is-dec-dig-spec}(da, sf, i) \lor \\
&\text{is-dec-dig-spec}(da, sf, n) \& \\
&\text{is-prop-sh-pc}(da, sf, n, \text{end}_{sh} ) \& \\
&\text{is-trail-sign}(da, sf, i) \lor \text{is-trail-sign}(da, sf, i-1) )
\end{align*}
\]

(194) is-prop-sh-pc(da,sf,n,n) =

\[
\begin{align*}
m = n \& (n = \text{end} \& \text{is-s-char}(n, sf(da)) \lor \\
&\text{end}_{sh} \& \text{is-s-char}(n, sf(da))) \lor \\
m = \text{end}_{ipc} \& (n = \text{end} \& \text{is-s-char}(n, sf(da))) \lor \\
m = n-1 \& \text{is-dec-dig-spec}(da, sf, m) \& \text{is-dec-dig-spec}(da, sf, n) \& \\
&\text{is-s-char}(n, sf(da)) \lor \text{is-s-char}(m, sf(da))
\end{align*}
\]
8.6.2.4 Proper drifting

The following rules are expressed:

a) Drifting must not begin after the first digit position; if drifting extends beyond \( u_{da} \), 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.

c) In the case of a sterling picture, if the drifting specification is a drifting editing specification or an asterisk, drifting must not extend beyond \( end_{pd} \).

d) Trailing editing specifications and Y-CHAR belong to the drifting part of the subfield.

e) Only certain kinds of specifications may be contained in the drifting part.

\[
(195) \text{is-prop-drift}(da,sf) = \\
\begin{array}{c}
\begin{align*}
1 \leq \text{beg}_{dr} \leq d_i \leq \text{end}_{dr} \leq \text{lgth}_{sf} & \\
(sf=sf_{mt} \land \neg \text{is-Q}(u_{da}) \land \text{end}_{dr} > u_{da} \land \text{end}_{dr} \geq d_{nd}) & \\
\text{(is-dr-ed-spec}(\text{spec}_{dr}) \supset \text{beg}_{dr} < \text{end}_{dr}) & \\
\text{(is-xsterling-pic}(da) \land (\text{is-dr-ed-spec} \lor \text{is-ASTER})(\text{spec}_{dr}) & \\
\text{end}_{dr} \leq \text{end}_{pd} & \\
\text{is-ed-spec}(sf,da,\text{beg}_{dr}) & \\
\text{(is-xsterling-pic}(da) \land (\text{is-dr-ed-spec} \lor \text{is-ASTER})(\text{spec}_{dr}) & \\
\text{end}_{dr} = \text{end}_{pd} & \\
\neg \text{is-ed-spec}(da,sf,\text{end}_{dr}+1) \land \neg \text{is-Y-CHAR} \text{elem}(\text{end}_{dr}+1),sf(da)) & \\
\end{align*}
\end{array}
\]

\[
(\forall i)(\text{beg}_{dr} \leq i \leq \text{end}_{dr} \supset \\
\text{(is-dr-ed-spec}(\text{spec}_{dr}) \land i = \text{beg}_{dr} \rightarrow \text{elem}(i,sf(da)) = \text{spec}_{dr}' & \\
T \rightarrow \text{is-ed-spec}(da,sf,i) \lor (\text{is-9-CHAR} \lor \text{is-Y-CHAR}) \text{elem}(i,sf(da)))
\]

for: \( (\text{is-xdec-pic} \lor \text{is-xsterling-pic})(da) \)
8.6.3 Representation in pictured form

In this chapter, the instruction \( \text{rep-pic}(\text{da},v) \) is defined which represents a value \( v \) with a picture attribute \( \text{da} \). Numeric pictures are treated in 8.6.3.1 to 8.6.3.3, character pictures in 8.6.3.4. The instruction \( \text{num-pic-char-conv}(\text{da},v) \) for numeric picture to character conversion, which is closely related to \( \text{rep-pic} \), is defined in 8.6.3.5.

**Pictured values.** For all kinds of pictures, \( \text{rep-pic} \) 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 stening 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, 9-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.

\[
(196) \quad \text{rep-pic}(\text{da},v) = \begin{array}{l}
\text{is-prop-pic}(\text{da}) \rightarrow \text{pass-rep-list}(\text{str-base}(\text{da}),v-1); \\
v-1: \text{rep-pic-1}(\text{da},v) \rightarrow \text{error}
\end{array}
\]

for: \( \text{is-num-pic}(\text{da}) \& \text{is-num-val}(v) \) \& \( \text{is-char-pic}(\text{da}) \& \text{is-char-val-list}(v) \)

Ref.: is-prop-pic 8-47(187), rep-list 8-10(34), str-base 8-39(156)

Note: The instruction is defined only for proper picture attributes \( \text{da} \).

\[
(197) \quad \text{rep-pic-1}(\text{da},v) = \begin{array}{l}
\text{is-char-pic}(\text{da}) \rightarrow \text{test-char-pic}(\text{da},\text{adjust-string}(\text{CHAR},\text{lgth}_\text{da}',v)) \\
\text{T} \rightarrow \text{rep-num-pic-1}(\text{xpic}(\text{da}),v)
\end{array}
\]

for: \( \text{is-num-pic}(\text{da}) \& \text{is-num-val}(v) \) \& \( \text{is-char-pic}(\text{da}) \& \text{is-char-val-list}(v) \)

Ref.: adjust-string 8-10(36)

Note: In case of a numeric picture, \( \text{da} \) is transformed into explicit form. This and all the following formulas of this chapter will be used only for proper \( \text{da} \), though this will no longer be mentioned explicitly.
8.6.3.1 Representation of numeric pictures

(198) \( \text{rep-num-pic-1}(da,v) = \)

\[ \text{is-CPLXs-mode}(da) \rightarrow \text{conc}(v\text{-re}, v\text{-im}); \]
\[ v\text{-re} : \text{rep-real-pic-1}(da\text{real}, \text{real}(v)), \quad v\text{-im} : \text{rep-real-pic-1}(da\text{real}, \text{imag}(v)) \]
\[ T \rightarrow \text{rep-real-pic-1}(da, \text{real}(v)) \]

for: is-xnum-pic(da) & is-num-val(v)

Ref.: conc 9-16(30)

(199) \( \text{rep-real-pic-1}(da,v) = \)

\[ \text{is-fix-pic}(da) \rightarrow \]
\[ \text{pass-lin-fix}(da, v-1); \]
\[ v-1 : \text{rep-subf}(da, sf_{\text{mt}}, \text{trunc}(v\text{.base}_{\text{da}} \uparrow (q_{\text{da}} - q_{\text{da}}))) \]
\[ \text{is-flt-pic}(da) \rightarrow \]
\[ \text{pass-lin-flt}(da, v-1, v-2); \]
\[ v-1 : \text{rep-subf}(da, sf_{\text{mt}}, \text{trunc}(v\text{.base}_{\text{da}} \uparrow (q_{\text{da}} - e_{\text{da}}))), \quad v-2 : \text{rep-subf}(da, sf_{\text{exp}}, e_{\text{da}}) \]

where: \( e_{\text{da}} = ((n)(\text{base}_{\text{da}} \uparrow (p_{\text{da}} - 1)) \text{abs}(v\text{.base}_{\text{da}} \uparrow (q_{\text{da}} - n)) \text{base}_{\text{da}} \uparrow p_{\text{da}}) \)

for: is-xnum-pic(da) & is-real-val(v)

Ref.: rep-subf 8-53(200), lin-fix 8-58(218), lin-flt 8-58(219)

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.

8.6.3.2 Representation of subfields

Metavariables

\( da \quad \text{is-xnum-pic} \)
\( v \quad \text{is-intg-val} \)
(200) \( \text{rep-subf}(da, sf, v) = \)

\[\text{is-subf-size-cond}(da, sf, v) \rightarrow \text{error}; \]
\[\text{call-cond}(\text{SIZE}, 607) \]
\[T \rightarrow \text{PASS: rep-valid-subf}(da, sf, v) \]

Ref.: \text{call-cond} 9-11(18)

Note: The SIZE condition is raised if \( v \) is too large to be held by the subfield.

(201) \( \text{is-subf-size-cond}(da, sf, v) = \)

\[\text{is-xsterling-pic}(da) \rightarrow \text{abs}(v) > 240.10(\text{nd}_{pd} + \text{q}_{da}) \]
\[T \rightarrow \text{abs}(v) > \text{base}_{da} \uparrow \text{nd}_{sf} \& (\text{is-2-compl-pic}(da) \cup v \neq -(\text{base}_{da} \uparrow \text{nd}_{sf})) \]

Ref.: \text{is-2-compl-pic} 8-48(190)

Note: Except for sterling pictures, \( \text{base}_{da} \uparrow \text{nd}_{sf} \) measures the capacity of the subfield. For two-complement representation, the negative limit value is representable.

(202) \( \text{rep-valid-subf}(da, sf, v) = \)

\[\text{is-bin-pic}(da) \& \text{is-sign-mag-pic}(da) \rightarrow \]
\[\text{rep-bin-sign}(da, sf, \text{sign}(v)) \cap \text{num-bit-list}(\text{nd}_{sf}, \text{abs}(v)) \]
\[\text{is-bin-pic}(da) \rightarrow \text{num-bit-list}(\text{lgth}_{sf}, \text{compl-val}(da, sf, v)) \]
\[T \rightarrow \text{drift-pv}(da, sf, i_{s}, \text{rep-sign}(da, sf, s_{v}, \text{rep-num-list}(da, sf, s_{v}, n_{lv}))) \]

where: \( s_{v} = \text{sign-char}(v) \)
\( n_{lv} = \text{mk-num-list}(da, sf, \text{abs}(v)) \)
\( i_{s} = (i)(i = \text{nd}_{sf} + 1 \& \text{lsisnd}_{sf} \& \text{elem}(i, n_{lv}) \neq 0) \& \)
\[\forall j)(1 \leq j \leq i \rightarrow \text{elem}(j, n_{lv}) = 0) \]

Note: \( i_{s} \) is the index of the first non-zero element of \( n_{lv} \).

Ref.: \text{is-sign-mag-pic} 8-48(189), \text{num-bit-list} 8-34(131)

(203) \( \text{sign-char}(v) = (v > 0 \rightarrow \text{PLUS}, T \rightarrow \text{MINUS}) \)
8.6.3.2.1 Binary subfields

Metavariable

\[ da \text{ is-bin-pic} \]

(204) \( \text{rep-bin-sign}(da,sf,sgn) = \)

\[ \text{is-SIGN-head.sf}(da) \rightarrow \) \( T \rightarrow (sgn = 1 \rightarrow \langle 0-BIT\rangle, sgn = -1 \rightarrow \langle 1-BIT\rangle) \]

for: \( \text{is-sign-mag-pic}(da) \)

(205) \( \text{compl-val}(da,sf,v) = \)

\[ v \geq 0 \rightarrow v \]

\[ \text{is-2-compl-pic}(da) \rightarrow 2^{\text{lgth}_{sf}} - \text{abs}(v) \]

\[ \text{is-1-compl-pic}(da) \rightarrow (2^{\text{lgth}_{sf}} - 1) - \text{abs}(v) \]

for: \( (\text{is-2-compl-pic} v \text{ is-1-compl-pic})(da) \)

Ref.: is-2-compl-pic 8-48(190), is-1-compl-pic 8-48(191)

8.6.3.2.2 Decimal and sterling subfields

The function mk-num-list(da,sf,v) decomposes the number v into a number list. The function rep-num-list(da,sf,sg,nl) replaces the digit positions of the subfield sf(da) by characters representing the elements of the number list nl. These representing characters are computed by rep-digit(spec, sg,n); the sign-character sg is needed if spec specifies overpunching; the subfunctions overpunch(sg,n) and rep-sterling(spec,n) are implementation-defined, but obey suitable axioms guaranteeing their one-to-one-ness. The function rep-sign(da,sf,sg,pv) inserts a sign character. The function drift-pv(da, sf,i,pv) performs zero-suppressing and drifting on a subfield of a pictured value, depending on the index i of the first significant digit position.

Metavariable

\[ da \text{ is-xdec-pic v is-xsterling-pic} \]
8.6.3.2.2.1 Decomposition into a number list

(206) mk-num-list(da,sf,v) =

\[\begin{align*}
\text{is-xdec-pic}(da) & \rightarrow \text{mk-num-list-1}(nd_{sf},v) \\
\text{is-xsterling-pic}(da) & \rightarrow \\
& \text{mk-num-list-1}(nd_{pd},\text{trunc}(v/240)) \\
& \text{mk-sterl-sUBL}(da,sf,\text{end}_{sh},\text{modulo}(\text{trunc}(v/12),20)) \\
& \text{mk-sterl-sUBL}(da,sf,\text{end}_{ipc},\text{modulo}(v/12)) \\
& \text{mk-num-list-1}(q_{da},\text{modulo}(v,10q_{da}))
\end{align*}\]

where: \( v_{1} = \text{trunc}(v/10q_{da}) \)

(207) mk-num-list-1(n,v) =

\[\begin{align*}
n & = 0 \rightarrow < > \\
T & \rightarrow \text{mk-num-list-1}(n-1,\text{trunc}(v/10)) <\text{modulo}(v,10)>
\end{align*}\]

for: \( n \geq 0 \)

(208) mk-sterl-sUBL(da,sf,n,v) =

\[\begin{align*}
(\text{is-6-CHAR} \lor \text{is-7-CHAR} \lor \text{is-8-CHAR}) & \cdot \text{elem}(n,sf(da)) \rightarrow <v> \\
T & \rightarrow \text{mk-num-list-1}(2,v)
\end{align*}\]

for: \( \text{is-xsterling-pic}(da) \land n \in \{\text{end}_{sh},\text{end}_{ipc}\} \land sf = sf_{mt} \)

8.6.3.2.2.2 Representation of number lists

(209) rep-num-list(da,sf,sg,nl) =

\[\begin{align*}
\mu(da;\langle(\text{elem}(d_{i}))\cdot sf;\text{rep-digit}(\text{elem}(d_{i},sf(da)),sg,\text{elem}(i,nl))\rangle |\text{lisisend}_{sf})
\end{align*}\]

(210) rep-digit(spec,sg,n) =

\[\begin{align*}
\text{is-Y-CHAR}(spec) & \land n = 0 \rightarrow \text{BLANK} \\
\text{is-T-CHAR}(spec) \lor \text{is-I-CHAR}(spec) \land \text{is-PLUS}(sg) \lor \text{is-R-CHAR}(spec) \land \\
\text{is-MINUS}(sg) & \rightarrow \text{overpunch}(sg,n) \\
(\text{is-6-CHAR} \lor \text{is-7-CHAR} \lor \text{is-8-CHAR})(spec) & \rightarrow \text{rep-sterl-dig}(spec,n)
\end{align*}\]
The functions

\[(211) \text{overpunch}(s_g,n)\]

\[(212) \text{rep-sterl-dig}(\text{spec},n)\]

are implementation-defined and are characterized by the axioms \(8-56(\text{213}), 8-56(\text{214})\) respectively.

\[(213) (\forall s_g-1,n-1,s_g-2,n-2) (0 \leq n-1 \leq 9 \& 0 \leq n-2 \leq 9 \Rightarrow
\text{is-char-val} \cdot \text{overpunch}(s_g-1,n-1) \& \text{overpunch}(s_g-1,n-1) \neq \text{single-num-char}(n-2) \&
(s_g-1 \neq s_g-2 \vee n-1 \neq n-2 \Rightarrow \text{overpunch}(s_g-1,n-1) \neq \text{overpunch}(s_g-2,n-2)))\]

Ref.: single-num-char 8-33(128), is-char-val 8-4(9).

\[(214) (\forall \text{spec},n-1,n-2)
(\text{is-sterl-dig-match}(\text{spec},n-1) \& \text{is-sterl-dig-match}(\text{spec},n-2) \Rightarrow
\text{is-char-val} \cdot \text{rep-sterl-dig}(\text{spec},n-1) \&
(n-1 \neq n-2 \Rightarrow \text{rep-sterl-dig}(\text{spec},n-1) \neq \text{rep-sterl-dig}(\text{spec},n-2)))\]

Ref.: is-char-val 8-4(9).

\[(215) \text{is-sterl-dig-match}(\text{spec},n) =
\text{is-8-CHAR}(\text{spec}) \& \text{Osns19} \lor (\text{is-7-CHAR} \lor \text{is-6-CHAR})(\text{spec}) \& \text{Osns11}\]

8.6.3.2.2.3 Representation of the sign

\[(216) \text{rep-sign}(\text{da},sf,s_g,pv) =
(\exists \chi_i)(\text{is-sign-pos}(\text{da},sf,\chi_i)) \rightarrow pv
\]

\[(\exists i)((\chi_s = (\text{elem}(i)) \cdot sf \& \text{is-sign-spec}(\text{da},sf,i)) \rightarrow
(\text{is-SIGN} \cdot \chi_s(\text{da}) \rightarrow \mu(pv;\chi_s:s_g),
(\text{is-PLUS} \lor \text{is-MINUS}) \cdot \chi_s(\text{da}) \& \chi_s(\text{da}) \neq s_g \rightarrow \mu(pv;\chi_s:\text{BLANK}),
T \rightarrow pv)\]

\[(\text{is-PLUS}(s_g) \rightarrow \mu(pv;\langle(\text{elem}(j)) \cdot sf_s:\text{BLANK}\rangle \mid j \in \{i_s,i_s+1\}),
T \rightarrow pv)\]

cont'd
where: \( s = (\lambda)\) is-sign-pos(da,sf,\( x \))

\( sf = (\lambda sf-1) i (x = (elem(i)) sf-1) \)

\( is = (\lambda (x = (elem(i)) sf) \)

Note: The third alternative treats the trailing signs R and DB (cf. is-sign-pos 8-45(178)). The case of the sign specifications T-CHAR, I-CHAR, R-CHAR (overpunching) has already been treated by rep-digit (cf. 8-55(210)).

### 8.6.3.2.2.4 Zero suppression and drifting

(217) drift-pv(da,sf,i,pv) =

\[
\begin{align*}
is-\mathcal{Q}(\text{spec}_{\text{dr}}) & \rightarrow pv \\
T & \rightarrow \mu(pv;\{\langle elem(j)\rangle sf; dr-\text{spec}_j \} \mid \text{beg}_{\text{dr}}; j; \text{dyn-end}_{\text{dr}} \& \neg is-\text{sterling-ed}_{\text{da}, sf, j})
\end{align*}
\]

where: dyn-end_{\text{dr}} =

\[
\begin{align*}
\text{sf} & = sf_{mt} \& \neg \text{is-}\mathcal{Q}(u_{da}) \& u_{da} < d_{i-1} < \text{end}_{\text{dr}} \rightarrow u_{da} \\
T & \rightarrow \min(d_{i-1}, \text{end}_{\text{dr}})
\end{align*}
\]

\( \text{dr-\text{spec}_j} = \)

\[
\begin{align*}
is-\text{ASTER}(\text{spec}_{\text{dr}}) & \rightarrow \text{ASTER} \\
is-\text{ed}\text{-\text{ed}\text{-spec}}(\text{spec}_{\text{dr}}) \& j = \text{dyn-end}_{\text{dr}} \rightarrow \text{elem} (\text{beg}_{\text{dr}}, \text{sf}(pv)) \\
T & \rightarrow \text{BLANK}
\end{align*}
\]

\( \text{is-}\text{sterling-ed}_{\text{da}, sf, j} = \)

\[
\begin{align*}
is-\text{xsterling-pic}(da) \& \text{end}_{p} < j < \text{end}_{inc} \& \neg is-\mathcal{Q}(da) \\
is-\text{ed}\text{-\text{spec}}(da, sf, j)
\end{align*}
\]

Note: Drifting positions of pv are replaced by BLANK or ASTER, except the last drifting position in the case that the drifting specification is a drifting editing specification. Since i is the index of the first non-zero digit, \( d_{i-1} \) is an upper limit for positions that may be suppressed; another upper limit is \( \text{end}_{\text{dr}} \); a third is \( u_{da} \) except when all digit positions are zero. Certain editing positions of sterling pictures are never suppressed.
8.6.3.3 Composition of subfields and linearization

Metavariable

da is-xnum-pic

(218) lin-fix(da,pv) =

\[
\text{is-bin-pic}(da) \rightarrow pv \\
T \rightarrow \text{sf}_{mt} \circ \text{ppic}(pv)
\]

for: is-fix-pic(da)

(219) lin-flt(da,pv-1,pv-2) =

\[
\text{is-bin-pic}(da) \rightarrow pv-1 \cap pv-2 \\
T \rightarrow \text{lin-flt-merge-subf}(da,\text{ppic}(pv-1),\text{ppic}(pv-2))
\]

for: is-flt-pic(da)

(220) merge-subf(da,pv-1,pv-2) =

\[
\mu(Ev-1; \langle (\text{elem}(i)) \circ \text{sf}_{exp}; \text{elem}(i,\text{sf}_{exp}(pv-2)) \mid \text{is-length}\circ \text{sf}_{exp}(pv-2) \rangle & \text{is-trail-sign}(da,\text{sf}_{exp},i) \rangle & \text{is-trail-sign}(da,\text{sf}_{exp},i-1))}
\]

Note: Cf. the note to is-sign-pos 8-45(178)

(221) lin-flt-1(pv) = \text{sf}_{mt}(pv) \cap \text{exp-sep}(pv) \cap \text{sf}_{exp}(pv)

(222) exp-sep(pv) =

\[
\text{is-}Q \circ \text{s-exp-sep}(pv) \rightarrow \langle > \\
T \rightarrow \langle s\text{-exp-sep}(pv) >
\]

8.6.3.4 Representation of character pictures

The instruction test-char-pic(da,v) checks the character string value v against the character picture attribute da. If v matches da, the unchanged value is returned; otherwise, the CONVERSION condition is raised and the representation is retried.
(223) test-char-pic(da,v) =
\[\text{is-char-pic-match}(da,v) \rightarrow \text{PASS}:v\]
\[T \rightarrow \text{rep-pic-1}(da,\text{corr-v});\]
\[\text{corr-v:call-conv-cond}(v,i_0,608)\]

where: \(i_0 =\)
\[(\forall i)(1 \leq i \leq \text{length}_da \Rightarrow \text{is-valid-single-char}(\text{elem}(i,\text{s-field}(da)),\text{elem}(i,v)))\]
for: \(\text{is-char-pic}(da) \& \text{is-char-val-list}(v)\)

Ref.: rep-pic-1 8-51(197), call-conv-cond 9-17(34)

(224) is-char-pic-match(da,v) =
\[\text{is-valid-single-char}(\text{elem}(i,\text{s-field}(da)),\text{elem}(i,v))\]
for: \(\text{is-char-pic}(da) \& \text{is-char-val-list}(v)\)

(225) is-valid-single-char(spec,v) =
\[\text{is-X-CHAR}(\text{spec}) \& \text{is-A-CHAR}(\text{spec}) \& (\text{is-letter} \& \text{is-BLANK})(v) \&\]
\[\text{is-9-CHAR}(\text{spec}) \& (\text{is-digit} \& \text{is-BLANK})(v)\]
for: \(\text{is-char-spec}(\text{spec}) \& \text{is-char-val}(v)\)

Ref.: is-letter 8-4(11), is-digit 8-5(12)

8.6.3.5 Conversion from numeric picture to character string

The definitions collected here are used in numeric to character conversion (cf. 8.5.1.1).

(226) num-pic-char-conv(da,v) =
\[\text{is-bin-pic}(da) \rightarrow \text{pass-bit-char-conv}(v-1);\]
\[v-1: \text{rep-pic-1}(da,v)\]
\[T \rightarrow \text{rep-pic-1}(da,v)\]
for: \(\text{is-num-pic}(da) \& \text{is-num-val}(v)\)

Ref.: rep-pic-1 8-51(197), bit-char-conv 8-36(138)

8.6.3.5
(227) \text{out-fix-pic}(m,n) = \text{mpic} \times \text{out-fix-pic}(m,n)

Ref.: mpic 8-43(162)

(228) \text{out-fix-pic}(m,n) =
\mu_0(<\text{s-mode:REAL}>,
\text{spec-list}_{\text{mt}}:<\text{s-dr-beg}:1>,
\text{spec-list}_{\text{exp}}>)
\times
\mu(<\text{s-dr-end}: (n=0 \rightarrow m-1, T \rightarrow m-n-2)>,
<s-dr-spec:MINUS>),
<s-dr-unit: (n=0 \rightarrow Q, T \rightarrow m-n-1)>)

where: \text{spec-list}_{\text{mt}} = \text{LIST} (i=1 \rightarrow \text{MINUS}, n=0 \& i=m-n \rightarrow \text{POINT},
T \rightarrow 9\text{-CHAR})

Note: \text{out-fix-pic}(m,n) \text{ is a fixed-point picture } da
with \text{lgth}_{\text{mt}} = m, \text{qda} = n, \text{ and a drifting MINUS.}

(229) \text{out-flt-pic}(m) =
\mu_0(<\text{s-mode:REAL}>,
\text{spec-list}_{\text{mt}}>)
\times
\mu(<\text{s-exp-sep:E-CHAR}>,
\text{spec-list}_{\text{exp}}>)
\times
\mu(<\text{s-dr-spec:MINUS}>,
\text{spec-list}_{\text{mt}} = \text{LIST} (i=1 \rightarrow \text{MINUS}, i=3 \rightarrow \text{POINT}, T \rightarrow 9\text{-CHAR})
\text{spec-list}_{\text{exp}} = \text{LIST} (i=1 \rightarrow \text{SIGN}, T \rightarrow 9\text{-CHAR})
\text{Note: out-flt-pic}(m) \text{ is a floating-point picture } da \text{ with}
\text{lgth}_{\text{mt}} = m, \text{lgth}_{\text{exp}} = \text{EXP-SIZE} + 1, \text{qda} = \text{pda} - 1.

The elementary object \text{EXP-SIZE} is an implementation-defined integer
value.

8.6.4 Value of a picture

The function \text{pic-val}(da,vr) \text{ yields a numeric value for numeric pictures,}
a character string value for character pictures. It performs the inverse
steps of \text{rep-pic}, and the greater part of its definition is given implicitly.
In order that \text{pic-val}(da,vr) \text{ be defined, } da \text{ and the string value represented}
by vr must match, i. e. they must satisfy the predicate \text{is-pic-match}(da,v)
defined in 8.6.4.2.
Metavariable

\( v \) is-char-val-list \( \lor \) is-bit-val-list

Abbreviations

for \((\text{is-xnum-pic} \& \text{is-CPLX\_s-mode})(\text{da})\):

\[
\begin{align*}
\text{v}_{\text{real}} &= \text{LIST}_{i=1}^{\text{lgth}_{\text{da}}/2} \text{elem}(i,\text{v}) \quad \text{first half of string representing a complex value} \\
\text{v}_{\text{imag}} &= \text{LIST}_{i=\text{lgth}_{\text{da}}/2+1}^{\text{lgth}_{\text{da}}} \text{elem}(i,\text{v}) \quad \text{second half of string representing a complex value}
\end{align*}
\]

(230) \( \text{pic-val}(\text{da, vr}) = \)

\[
\begin{align*}
\text{is-prop-pic}(\text{da}) &\longrightarrow \text{pic-val-1}(\text{da, val-list(str-base(\text{da, vr}))}) \\
T &\longrightarrow \text{error}
\end{align*}
\]

for: \( \text{is-pic}(\text{da}) \& (\text{is-char-vr-list} \lor \text{is-bit-vr-list})(\text{vr}) \)

Ref.: is-prop-pic 8-47(187), val-list 8-8(27), str-base 8-39(156)

(231) \( \text{pic-val-1}(\text{da, v}) = \)

\[
\begin{align*}
\text{is-char-pic}(\text{da}) &\longrightarrow \\
(\text{is-char-pic-match}(\text{da, v}) &\longrightarrow \text{v, } T &\longrightarrow \text{error}) \\
T &\longrightarrow \text{num-pic-val-1(xpic(da), v})
\end{align*}
\]

for: \( \text{is-prop-pic}(\text{da}) \)

Ref.: is-char-pic-match 8-59(224)

Note: in case of numeric picture, \( \text{da} \) is transformed into explicit form.

Again, the remaining formulas of this chapter will be restricted to \( \text{is-prop-pic}(\text{da}) \) without explicit mentioning.

8.6.4.1 Value of a numeric picture

Metavariables

\( \text{da} \) is-xnum-pic
\( x \) is-real-val
(232) num-pic-val-1(da,v) =
  is-CPLX-s-mode(da) →
  cplx(num-pic-val-1(da_real,v_real),num-pic-val-1(da_real,v_imag))

is-fix-pic(da) →
  (lx)(∃m)(is-fix-match(da,m,v) & is-norm-subf(da,sf_mt,m) & x = m.base^da_t(gda_qda))

is-flt-pic(da) →
  (lx)(∃m,n)(is-flt-match(da,m,n,v) & is-norm-subf(da,sf_mt,m) & is-norm-subf(da,sf_exp,n) & x = m.base^da_t(n_qda))

Note: Cf. the definitions of rep-num-pic-1 8-52(198) and rep-real-pic-1 8-52(199). The conditions is-fix-match and is-flt-match guarantee that x is an inverse of v, i.e. that v represents x. The conditions is-norm-subf additionally guarantee that x is unique.

(233) is-fix-match(da,m,v) =
  ¬is-subf-size-cond(da,sf_mt,m) & v = lin-fix(da,rep-valid-subf(da,sf_mt,m))

for: is-fix-pic(da)

Ref.: is-subf-size-cond 8-53(201), rep-valid-subf 8-53(202),
      lin-fix 8-58(218)

(234) is-flt-match(da,m,n,v) =
  ¬is-subf-size-cond(da,sf_mt,m) & ¬is-subf-size-cond(da,sf_exp,n) &
  v = lin-flt(da,rep-valid-subf(da,sf_mt,m),rep-valid-subf(da,sf_exp,n))

for: is-flt-pic(da)

Ref.: is-subf-size-cond 8-53(201), rep-valid-subf 8-53(202),
      lin-flt 8-58(219)

(235) is-norm-subf(da,sf,m) =
  ¬is-signed-subf(da,sf) ⊃ m ≥ 0
8.6.4.2 Testing of string values against picture attributes

The predicate is-pic-match(da,v) can be applied to arbitrary pictures is-pic(da). It combines the predicates is-char-pic-match, is-fix-match, is-flt-match used by the function pic-val. The predicate will be used only for is-prop-pic(da).

(237) is-pic-match(da,v) =

\[
\text{is-char-pic(da)} \land \text{is-char-pic-match(da,v)} \\
T \land \text{is-num-pic-match(xpic(da),v)}
\]

for: is-pic(da)

Ref.: is-char-pic-match 8-59(224)

(238) is-num-pic-match(da,v) =

\[
\text{is-CPLX.s-mode(da)} \land \text{is-num-pic-match(da,real,v,real)} \land \text{is-num-pic-match(da,real,v,imag)} \\
\text{is-fix-pic(da)} \lor (\exists m)(\text{is-fix-match(da,m,v)}) \\
\text{is-flt-pic(da)} \lor (\exists m,n)(\text{is-flt-match(da,m,n,v)})
\]

for: is-xnum-pic(da)

Ref.: is-fix-match 8-62(233), is-flt-match 8-62(234)

8.6.5 Transformation of picture attributes

The function mk-arithm-pic(da) transforms a numeric picture attribute da into an arithmetic attribute. The function mk-str-1(da) (cf. 8-39(155)) transforms a string class attribute into a string attribute, and only its subfunction pic-length(da) remains to be defined.
(239) \( \text{mk-arithm-pic(da)} = \)
\[
\mu_0(<s\text{-mode}:s\text{-mode(da)}>,
\begin{align*}
&s\text{-base}:(\text{is-bin-pic(da)} \rightarrow \text{BIN},\text{T} \rightarrow \text{DEC}), \\
&s\text{-scale}:(\text{is-flt-pic(da)} \rightarrow \text{FLT},\text{T} \rightarrow \text{FIX}), \\
&s\text{-prec:pic-prec\times pic(da)}, \\
&s\text{-scale-f}:(\text{is-flt-pic(da)} \rightarrow \Omega,\text{T} \rightarrow q_{da} - g_{da})
\end{align*}
\]

\text{for: is-num-pic(da)}

(240) \( \text{pic-prec(da)} = \)
\[
\text{is-xsterling-pic(da)} \rightarrow nd_{pd} + 3 + q_{da}
\]
\text{T} \rightarrow nd_{mt}

\text{for: is-xnum-pic(da)}

Note: For sterling pictures, the shillings field and integral pence field are counted as if containing together three digit positions. Hence, in general, \( P_{da} \times nd_{mt} \) for sterling pictures.

(241) \( \text{pic-length(da)} = \)
\[
\text{is-char-pic(da)} \rightarrow \text{length\times field(da)} \\
\text{is-CPLX\times s-mode(da)} \rightarrow 2 \cdot \text{pic-length(da}_{\text{real}}) \\
\text{is-fix-pic(da)} \rightarrow \text{length\times sf_{mt}(da)} \\
\text{is-flt-pic(da)} \rightarrow \text{length\times lin-flt-1(da)}
\]

\text{for: is-pic(da)}

Ref.: \text{lin-flt-1 8-58(221)}
9. CONDITIONS

This chapter defines the parts of PL/I which are common in dealing with on-conditions. Due to the heterogeneous purposes served by the various types of conditions - computational, input/output, program-checkout, list-processing, programmer-named and system-action - 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. The evaluation of condition prefixes of the program takes place on block entry, procedure interpretation and statement interpretation. A major subpart of the Condition Status CS contains the current condition enabling status during the interpretation of program text.

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 for each condition can be modified by executing on and revert statements. The current specifications of condition actions are contained in a major subpart of CS.

The raising and execution of a condition action has some analogy to the execution of a call to a parameterless procedure. The raising either is the result of a specific condition situation in the interpretation of a program or the result of the execution of a signal statement for a condition. A program which due to a condition situation executes a condition action, on return from the condition action will become undefined for several of the PL/I on-conditions. Only an abnormal return via a goto or exit continues the program interpretation.

Section 9.1 of this chapter summarizes the treatment of condition prefixes, 9.2 defines the condition system action and the establishment of condition actions by on and revert statements. Section 9.3 defines the updating of state parts for condition - builtin functions by condition actions. Section 9.4 defines the activation of a condition action and the auxiliary call instructions used for the condition activation in other parts of the document and the signal statement.
**Metavariabes**

<table>
<thead>
<tr>
<th>Metavariable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>at</td>
<td>is-at</td>
</tr>
<tr>
<td>ca</td>
<td>is-cond-act</td>
</tr>
<tr>
<td>cbif</td>
<td>is-cbif</td>
</tr>
<tr>
<td>cond</td>
<td>is-cond</td>
</tr>
<tr>
<td></td>
<td>is-f-io-cond</td>
</tr>
<tr>
<td>env</td>
<td>is-e</td>
</tr>
<tr>
<td>reflist</td>
<td>is-ref-list</td>
</tr>
<tr>
<td>t</td>
<td>is-p-on-st</td>
</tr>
<tr>
<td></td>
<td>is-revert-st</td>
</tr>
<tr>
<td>type</td>
<td>is-intg-val</td>
</tr>
</tbody>
</table>

9.1 **Condition Enabling**

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 Status $CS$, the block prefix part and the statement prefix part (cf. 2.5.2).

Condition prefixes heading a begin statement or a procedure statement have the scope of the respective blocks. They are interpreted by the instruction `update-cs` in the block interpretation in `int-block` (5-3(1)) and in the procedure interpretation in `int-proc-body` (5-50(90)). The instruction `update-cs` merges the set of prefixes which is valid in the interpretation of the static environment of a block or procedure with the explicitly specified block or procedure condition prefixes.

At the beginning of the program interpretation a standard enabling status exists in the initial state of $CS$ (cf. 3.1).
During the interpretation of PL/I statements the enabling status as defined for the block can be modified by explicit statement condition prefixes. The statement enabling status is kept in the statement prefix part of CS which is updated by update-st-cs during the interpretation of a statement by int-st (5-17(25)). It is not necessary to stack the statement prefix part during interpretation of compound statements according to the prefix scope rules of the language.

9.2 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 on and revert statements. If an on statement specifies a statement as the condition action, this is kept with some additional information in the condition-action-part of CS. On statements can specify as an option that the first action of the specified condition action is an implementation defined snap-action.

9.2.1 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.

The system action for most conditions raises the error condition. The system action for the check condition distinguishes between various data types and requires an environment for achieving the type distinction. The conversion condition makes the source of the conversion error available in the action entered due to the conversion condition and in the error action specified as the system action. The source of the conversion error is accessible through the onsource condition builtin function which requires the condition-bif argument in the condition call (cf. 9.4.1) for later reference to ONSOURCE.
(1) \( syst\text{-}cond\text{-}exec(\text{cond}, \text{env}, \text{cbif}, \text{type}) = \)
\[
\text{is-UFL}(\text{cond}) \lor \text{is-STRG}(\text{cond}) \lor \text{is-NAME}\text{-}\text{s-io-cond}(\text{cond}) \lor \text{is-progr\text{-}named\text{-}cond}(\text{cond}) \\
\text{comment}(\text{cond}, \text{env}, \text{cbif}) \\
\text{is-ENDPAGE}\text{-}\text{s-io-cond}(\text{cond}) \rightarrow syst\text{-}endpage\text{-}exec(f_{\text{den}}) \\
\text{is-CONV}(\text{cond}) \lor \text{is-named\text{-}io-cond}(\text{cond}) \lor \text{is-f\text{-}io-cond}(\text{cond}) \\
\{\text{call-cond-1(\text{ERROR}, \text{env}, \text{cbif}, \text{type})}; \text{comment}(\text{cond}, \text{env}, \text{cbif})\} \\
\text{is-FOFL}(\text{cond}) \lor \text{is-OFL}(\text{cond}) \lor \text{is-SIZE}(\text{cond}) \lor \text{is-ZDIV}(\text{cond}) \lor \\
\text{is-SUBRG}(\text{cond}) \lor \text{is-AREA}(\text{cond}) \rightarrow \\
\{\text{call-cond(\text{ERROR}, \text{type})}; \text{comment}(\text{cond}, \text{env}, \text{cbif})\} \\
\text{is-check}(\text{cond}) \rightarrow \text{int\text{-}stand\text{-}check}(\text{cond}, \text{env}) \\
\text{is-FINISH}(\text{cond}) \rightarrow \text{null} \\
\text{is-ERROR}(\text{cond}) \rightarrow syst\text{-}error\text{-}exec \\
\text{where}: f_{\text{den}} = (\text{is-cond}(\text{cond}) \rightarrow \text{sel}(\text{s\text{-}file}(\text{cond}))(\text{env})(\text{DN}), \\
\text{is-f\text{-}io-cond}(\text{cond}) \rightarrow \text{s\text{-}f\text{-}den}(\text{cond}))
\]

Ref.: \text{call-cond} \: 9-11(18), \: \text{call-cond-1} \: 9-12(19), \: syst\text{-}endpage\text{-}exec \: 10-102(279), \: \text{is-f\text{-}io-cond} \: 2-32(106)

(2) \text{comment}(\text{cond}, \text{env}, \text{cbif}) = \]
\[
\mu(M; <\text{s\text{-}comment}; \text{s\text{-}comment}(M) \circ \text{comment-f}(\text{cond}, \text{env}, \text{cbif}, \xi)>)
\]

(3) \text{comment-f}(\text{cond}, \text{env}, \text{cbif}, \xi) = \]

Note: Implementation defined function whose values satisfy the predicate \text{is-comment}.

(4) \((\forall \text{cond}, \text{env}, \text{cbif}, \xi) (\text{is-comment}(\text{comment-f}(\text{cond}, \text{env}, \text{cbif}, \xi)))
\]

for: \text{is\text{-}state}(\xi)

Ref.: is\text{-}comment \: 2-44(145)
9.2.2 Interpretation of on statement

The interpretation of an on statement for a condition establishes the on-unit as the new action for that specific condition in the current condition status CS. An on statement for a check-condition establishes as many condition actions as the check reference possesses base elements for a structure reference or alternative elements for a cell reference.

9.2.2.1 Interpretation of on statement

The interpretation of an on statement for a condition establishes the on-unit as the new action for that specific condition in the current condition status CS. An on statement for a check-condition establishes as many condition actions as the check reference possesses base elements for a structure reference or alternative elements for a cell reference.

9.2.2.2 Interpretation of on statement

The interpretation of an on statement for a condition establishes the on-unit as the new action for that specific condition in the current condition status CS. An on statement for a check-condition establishes as many condition actions as the check reference possesses base elements for a structure reference or alternative elements for a cell reference.
(8) \( \text{is-corr-check}(c_{dl}, \text{env}, \text{at}) = \)
\[
(\forall i)(\text{is-check}(\text{elem}(i, c_{dl}) \Rightarrow (\neg \text{is-PARAM-scope}(\text{attr}_{idl}) \&
\begin{align*}
&\text{(is-prop-var}(\text{attr}_{idl}) \& \text{is-entry}(\text{attr}_{idl}) \& \text{is-LABEL}(\text{attr}_{idl})))))
\]
where: \( \text{attr}_{idl} = \text{s-attr}(\text{sel-s-id-list}(\text{elem}(i, c_{dl}))(\text{env})(\text{at})) \)
for: \( \text{is-cond-list}(c_{dl}) \)

(9) \( \text{is-corr-cond}(c_{dl}, \text{env}, \text{at}) = \)
\[
\text{length}(c_{dl}) = 1 \&
(\text{is-named-io-cond}(\text{head}(c_{dl})) \&
\begin{align*}
&\text{is-FILE-s-attr}(\text{sel}(\langle \text{s-file}(\text{head}(c_{dl})) \rangle)(\text{env})(\text{at})) \&
\text{is-progr-named-cond}(\text{head}(c_{dl})) \&
\text{is-COND-s-attr}(\text{sel}(\langle \text{head}(c_{dl}) \rangle)(\text{env})(\text{at})) \&
\text{is-cond}(\text{head}(c_{dl})) \& \neg \text{is-check}(\text{head}(c_{dl}))
\]
\]
for: \( \text{is-cond-list}(c_{dl}) \)

(10) \( \text{on-establish-1}(\text{erl}, \text{t}) = \)
\[
\text{null};
\begin{align*}
&\text{on-establish}(\text{elem}(i, \text{erl}), \text{t}) \mid \text{is-length}(\text{erl})
\]
for: \( \text{is-ref-list}(\text{erl}), \text{is-p-on-st}(\text{t}) \)

(11) \( \text{on-establish}(\text{cond}, \text{t}) = \)
\[
\text{s-cs}: \mu(\text{CS}(<\text{cond-dyn-sel}(\text{cond}, \text{CS}, \text{DN}, \text{AT}) \cdot \text{s-cap}:
\begin{align*}
&\mu_{o}(\langle \text{s-snap}: \text{s-snap}(\text{t}) \rangle, \langle \text{s-cond}: \text{cond} \rangle, \langle \text{s-e}: \text{E} \rangle, \langle \text{s-on-unit}: \text{s-on-unit}(\text{t}) \rangle, \langle \text{s-bpp}: \text{s-bpp}(\text{CS}) \rangle)
\end{align*}
\]
for: \( \text{is-p-on-st}(\text{t}), \text{is-cond}(\text{cond}) \)
Ref.: cond-dyn-sel 2-30(100)
9.2.3 Interpretation of revert statement

The interpretation of a revert statement in a block or procedure for a condition establishes the condition action which was valid before the execution of the first on statement in that block or procedure for the condition. Due to the dynamic inheritance rules the condition action as part of $C$ is stacked in the dump $D$, whenever a new block or procedure is interpreted. On execution of a revert statement the stacked condition action is reinstalled. Again the check condition expands to base elements and alternative elements as for the on statement.

(12) \[
\text{int-revert-st}(t) = \begin{cases} 
\text{is-corr-check}(\text{s-on-cond}(t), E, AT) & \rightarrow \\
\text{revert-on-l}(\text{erl}, t); \\
\text{erl} : \text{expand-reflist}(\text{s-on-cond}(t), E) & \rightarrow \\
\text{is-corr-cond}(\text{s-on-cond}(t), E, AT) & \rightarrow \\
\text{revert-on}(\text{head}(\text{s-on-cond}(t)), t) \\
T & \rightarrow \text{error} \\
\end{cases}
\]

for: \text{is-revert-st}(t)

Ref.: \text{expand-reflist} 9-15(29), \text{is-corr-check} 9-6(8), \text{is-corr-cond} 9-6(9)

(13) \[
\text{revert-on-l}(\text{erl}, t) = 
\begin{cases} 
null; & \rightarrow \\
\{ \text{revert-on}(\text{elem}(1, \text{erl}), t) \mid \text{lsis-length}(\text{erl}) \} & \rightarrow \\
\end{cases}
\]

for: \text{is-revert-st}(t), \text{is-ref-list}(\text{erl})

(14) \[
\text{revert-on}(\text{cond}, t) = 
\begin{cases} 
\text{s-cs} : \mu(CS : \text{cond-dyn-sel}(\text{cond}, E, DN, AT) \cdot \text{s-cap} : \text{cond-dyn-sel}(\text{cond}, E, DN, AT) \cdot \text{s-cap} \cdot \text{s-cs}(D) ) & \rightarrow \\
\end{cases}
\]

for: \text{is-revert-st}(t), \text{is-cond}(\text{cond})

Ref.: \text{cond-dyn-sel} 2-30(100)

Note: The execution of \text{revert-on} in the dynamically first block activation, i.e., the initial external procedure, deletes the appropriate condition action part from $s\text{-cap}(CS)$ because $s\text{-cs}(D)$ yields $\varnothing$. The execution of \text{revert-on} as the only statement of an on-unit has no effect due to the activation of a new block for the interpretation of the on-unit (9-13(24)).
9.3 Condition Builtin Function Status

Condition built-in functions change the value they return as a consequence to condition raising. The new value obtained remains unchanged in the dynamic descendence of the condition raising, i.e., in all blocks entered from an on-unit executed as consequence of the condition raising. If a condition is raised and only the system action executed, no change of the condition-bif-values is required. The updating of condition-bif-values is defined by the instruction `update-c bif` which is executed as part of the interpretation of a condition action by `int-cond` (9-13(24)). The value of a condition built-in function, where the corresponding component of the condition bif part is \( \emptyset \), is described in chapter 11 with the builtin function.

\[(15) \quad \text{update-c bif}(\text{cond}, \text{cbif}, \text{type}) = \]

\[
\text{is-SIGNAL} \cdot \text{st} \cdot \text{s-c bif}(\text{CS}) \land \text{is-N AME} \cdot \text{s-io-cond}(\text{cond}) \rightarrow
\]

\[
\text{s-cs} \cdot \mu(\text{CS}) \cdot \text{s-c bif}(\text{CS}) \cdot <\text{s-onfile} : \text{s-onfile}(\text{cbif})>,
\]

\[
<s-datafield: \emptyset>,
\]

\[
<s-oncode: \text{code}(\text{cond}, \text{type})>,
\]

\[
<s-onloc: \text{entry} \cdot \text{s-c bif}(\text{CS})>,
\]

\[
<s-ten: \text{s-ten}(\text{cbif})>)
\]

\[
\text{is-N AME} \cdot \text{s-io-cond}(\text{cond}) \rightarrow
\]

\[
\text{s-cs} \cdot \mu(\text{CS}) \cdot \text{s-c bif}(\text{CS}) \cdot <\text{s-onsource} : \emptyset>,
\]

\[
<s-\text{onchar} : \emptyset>,
\]

\[
<s-oncode: \text{code}(\text{cond}, \text{type})>,
\]

\[
<s-onloc: \text{entry} \cdot \text{s-c bif}(\text{CS})>,
\]

\[
<s-ten : \text{s-ten}(\text{cbif})>)
\]

\[
\text{is-SIGNAL} \cdot \text{s-st} \cdot \text{s-c bif}(\text{CS}) \land \text{is-CON V}(\text{cond}) \rightarrow
\]

\[
\text{s-cs} \cdot \mu(\text{CS}) \cdot \text{s-c bif}(\text{CS}) \cdot <\text{s-onsource} : \emptyset>,
\]

\[
<s-\text{onchar} : \emptyset>,
\]

\[
<s-oncode: \text{code}(\text{cond}, \text{type})>,
\]

\[
<s-onloc: \text{entry} \cdot \text{s-c bif}(\text{CS})>,
\]

\[
<s-ten : \text{s-ten}(\text{cbif})>)
\]

\[
\text{is-CON V}(\text{cond}) \land \neg \text{is-} \emptyset \cdot \text{s-curr-} \text{file} \cdot \text{s-c bif}(\text{CS}) \land \neg \text{is-} \emptyset \cdot \text{s-onkey}(\text{cbif}) \rightarrow
\]

\[
\text{s-cs} \cdot \mu(\text{CS}) \cdot \text{s-c bif}(\text{CS}) \cdot <\text{s-onfile} : \text{s-curr-} \text{file} \cdot \text{s-c bif}(\text{CS})>,
\]

\[
<s-\text{onsource} : \text{s-} \text{onsource}(\text{cbif})>,
\]

\[
<s-\text{onchar} : \text{onchar}(\text{cbif})>,
\]

\[
<s-oncode: \text{code}(\text{cond}, \text{type})>,
\]

\[
<s-onloc: \text{entry} \cdot \text{s-c bif}(\text{CS})>,
\]

\[
<s-ten : \text{s-ten}(\text{cbif})>)
\]

cont'd
is-CONV(cond) & \neg is-\bigcirc s\text{-curr-file}\text{-s-cbf}(CS) →

\[ s_{-cs} : \mu(CS); s_{-cbif} : \mu(s_{-cbif}(CS)); s\text{-onfile} : s\text{-curr-file}\text{-s-cbf}(CS) >,
\]

\[ s\text{-onsource} : s\text{-onsource}(cbif) >,
\]

\[ s\text{-onchar} : s\text{-onchar}(cbif) >,
\]

\[ s\text{-oncode} : code(\text{cond, type}) >,
\]

\[ s\text{-onloc} : s\text{-entry}\text{-s-cbf}(CS) >,
\]

\[ s\text{-ten} : s\text{-ten}(cbif) >)
\]

is-CONV(cond) →

\[ s_{-cs} : \mu(CS); s_{-cbif} : \mu(s_{-cbif}(CS)); s\text{-onfile} : s\text{-onfile}(cbif) >,
\]

\[ s\text{-onkey} : \emptyset >,
\]

\[ s\text{-oncode} : code(\text{cond, type}) >,
\]

\[ s\text{-oncount} : \emptyset >,
\]

\[ s\text{-onloc} : s\text{-entry}\text{-s-cbf}(CS) >,
\]

\[ s\text{-ten} : s\text{-ten}(cbif) >)
\]

is-SIGNAL\text{-s-st}\text{-s-cbf}(CS) & (\neg is-UNDF\text{-s-io-cond}(\text{cond}) \lor \neg is-\text{ENDPAGE}\text{-s-io-cond}(\text{cond})) →

\[ s_{-cs} : \mu(CS); s_{-cbif} : \mu(s_{-cbif}(CS)); s\text{-onfile} : s\text{-onfile}(cbif) >,
\]

\[ s\text{-onkey} : s\text{-onkey}(cbif) >,
\]

\[ s\text{-oncode} : s\text{-oncode}(cbif) >,
\]

\[ s\text{-oncount} : s\text{-oncount}(cbif) >,
\]

\[ s\text{-onloc} : s\text{-entry}\text{-s-cbf}(CS) >,
\]

\[ s\text{-ten} : s\text{-ten}(cbif) >)
\]

is-UNDF\text{-s-io-cond}(\text{cond}) \lor is-\text{ENDPAGE}\text{-s-io-cond}(\text{cond}) →

\[ s_{-cs} : \mu(CS); s_{-cbif} : \mu(s_{-cbif}(CS)); s\text{-onfile} : s\text{-onfile}(cbif) >,
\]

\[ s\text{-oncode} : code(\text{cond, type}) >,
\]

\[ s\text{-onloc} : s\text{-entry}\text{-s-cbf}(CS) >,
\]

\[ s\text{-ten} : s\text{-ten}(cbif) >)
\]

is-RECORD\text{-s-io-cond}(\text{cond}) & \neg is-\bigcirc s\text{-onkey}(cbif) →

\[ s_{-cs} : \mu(CS); s_{-cbif} : \mu(s_{-cbif}(CS)); s\text{-onfile} : s\text{-onfile}(cbif) >,
\]

\[ s\text{-onkey} : s\text{-onkey}(cbif) >,
\]

\[ s\text{-oncode} : s\text{-oncode}(cbif) >,
\]

\[ s\text{-oncount} : s\text{-oncount}(cbif) >,
\]

\[ s\text{-onloc} : s\text{-entry}\text{-s-cbf}(CS) >,
\]

\[ s\text{-ten} : s\text{-ten}(cbif) >)
\]

cont'd
is-io-cond • s-io-cond(cond) & • is-\textbackslash R\textbackslash s-onkey(cbif) —

\[ s-cs: \mu(CS; <s-cbif; \mu(s-cbif(CS)); <s-onfile; s-onfile(cbif)>), \]
\[ <s-onkey; s-onkey(cbif)>, \]
\[ <s-oncode; code(cond, type)>, \]
\[ <s-oncount; s-oncount(cbif)>, \]
\[ <s-onloc; s-entry; s-cbif(CS)>, \]
\[ <s-ten; s-ten(cbif)>) > \]

is-RECORD • s-io-cond(conds) —

\[ s-cs: \mu(CS; <s-cbif; \mu(s-cbif(CS)); <s-onfile; s-onfile(cbif)>), \]
\[ <s-oncode; s-oncode(cbif)>, \]
\[ <s-oncount; s-oncount(cbif)>, \]
\[ <s-onloc; s-entry; s-cbif(CS)>, \]
\[ <s-ten; s-ten(cbif)>) > \]

is-io-cond • s-io-cond(cons) —

\[ s-cs: \mu(CS; <s-cbif; \mu(s-cbif(CS)); <s-onfile; s-onfile(cbif)>), \]
\[ <s-oncode; code(cond, type)>, \]
\[ <s-oncount; s-oncount(cbif)>, \]
\[ <s-onloc; s-entry; s-cbif(CS)>, \]
\[ <s-ten; s-ten(cbif)>) > \]

T —

\[ s-cs: \mu(CS; <s-cbif; \mu(s-cbif(CS)); <s-oncode; code(cond, type)>, \]
\[ <s-onloc; s-entry; s-cbif(CS)>, \]
\[ <s-ten; s-ten(cbif)>) > \]

(16) \text{code}(\text{cond, type}) =

Note: Implementation defined function, whose values satisfy the axiom 17.
The argument type characterizes the instruction in the ULD, where the
interrupt occurred.

(17) (\forall \text{cond, type})(\text{is-intg-val} (\text{code}(\text{cond, type})))
9.4 Condition Activation

If the interpretation of a program arrives in certain situations the language specifies that a condition is to be raised. The actual execution of a condition action specified for the specific condition may take place at a later point in the interpretation because PL/I does not specify asynchronous interrupts of a program. 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 the auxiliary instructions used for the condition activation.

9.4.1 Interpretation of the condition call

The instruction \texttt{call-cond-1} is the general form of a condition call which is used if the condition name contains an identifier requiring an environment env for the name resolution, or if the condition call requires an argument cbif for updating the condition builtin function part of \texttt{CS}. Where none of those requirements exists, as e.g. for some computational conditions, the instruction \texttt{call-cond} is used.

The instruction \texttt{call-cond-1} distinguishes between prefix controlled and uncontrolled conditions. Furthermore, it distinguishes conditions which on normal return permit further interpretation, from conditions which on normal return arrive at an error situation. Another distinction is required for disabled conditions, which when raised result in no action.

The instruction \texttt{call-cond-1} inspects the condition status for an appropriate condition action. If none is present or the on-unit is SYSTEM \texttt{syst-cond-exec} is used to execute the system action, otherwise \texttt{int-cond} is used to execute the condition action. The snap action is executed if so specified by the condition action.

(18) \texttt{call-cond\{cond, type\} =

\texttt{is-FOFL(cond) \lor is-OFL(cond) \lor is-SIZE(cond) \lor is-STRG(cond) \lor
is-SUBRG(cond) \lor is-UFL(cond) \lor is-ZDIV(cond) \lor is-AREA(cond) \lor
is-ERROR(cond) \lor is-FINISH(cond) \rightarrow call-cond-1\{cond, \Omega, \Omega, type\}

T \rightarrow error}
(19) \( \text{call-cond-1}(\text{cond},\text{env},\text{cbif},\text{type}) = \)
\[
(\neg \text{is-}\emptyset(\text{cond-dyn-sel}(\text{cond},\text{env},\text{DN,AT}) \cdot \text{s-spp}(\text{CS})) \land \\
(\text{is-CONV}(\text{cond}) \lor \text{is-STRG}(\text{cond}) \lor \text{is-UFL}(\text{cond}) \lor \text{is-check}(\text{cond}) \lor \\
\text{is-SIGNAL}(\text{cond}) \lor \text{is-AREA}(\text{cond}) \lor \\
\text{is-named-io-cond}(\text{cond}) \lor \text{is-f-io-cond}(\text{cond}) \lor \\
\text{is-progr-named-cond}(\text{cond}) \lor \text{is-ERROR}(\text{cond}) \lor \text{is-FINISH}(\text{cond}) \\
\rightarrow \\
\text{call-cond-2}(\text{cond},\text{env},\text{cbif},\text{type}) \\
\neg \text{is-}\emptyset(\text{cond-dyn-sel}(\text{cond},\text{env},\text{DN,AT}) \cdot \text{s-spp}(\text{CS})) \land \\
(\text{is-UFL}(\text{cond}) \lor \text{is-check}(\text{cond}) \rightarrow \text{null}) \\
\rightarrow \text{error} \\
\text{call-cond-2}(\text{cond},\text{env},\text{cbif},\text{type}) \\
\neg \text{is-}\emptyset(\text{cond-dyn-sel}(\text{cond},\text{env},\text{DN,AT}) \cdot \text{s-spp}(\text{CS})) \land \\
(\text{is-UFL}(\text{cond}) \lor \text{is-check}(\text{cond}) \rightarrow \text{null}) \\
\rightarrow \text{error}
\]

Ref.: \text{cond-dyn-sel \ 2-30(100)}, \quad \text{is-f-io-cond \ 2-32(106)}

(20) \( \text{is-cbif} = \langle \text{oncode} : \text{is-intg-val} \lor \text{is-}\emptyset, \\
\text{oncount} : \text{is-intg-val} \lor \text{is-}\emptyset, \\
\text{onfile} : \text{is-id} \lor \text{is-}\emptyset, \\
\text{onkey} : \text{is-char-val-list} \lor \text{is-}\emptyset, \\
\text{datafield} : \text{is-char-val-list} \lor \text{is-}\emptyset, \\
\text{onchar} : \text{is-intg-val} \lor \text{is-}\emptyset, \\
\text{onsource} : \text{is-n} \lor \text{is-}\emptyset, \\
\text{onlloc} : \text{is-id} \lor \text{is-}\emptyset, \\
\text{onten} : \text{is-n} \lor \text{is-}\emptyset \rangle \)

Ref.: \text{is-char-val-list \ 8-4(9)}, \quad \text{is-n \ 2-17(35)}

(21) \( \text{call-cond-2}(\text{cond},\text{env},\text{cbif},\text{type}) = \)
\[
\text{is-p-st} \cdot \text{on-unit}(\text{cond-dyn-sel}(\text{cond},\text{env},\text{DN,AT}) \cdot \text{s-cap}(\text{CS})) \\
\rightarrow \text{int-cond}(\text{cond-dyn-sel}(\text{cond},\text{env},\text{DN,AT}) \cdot \text{s-cap}(\text{CS}),\text{cbif},\text{type});
\]
\[
\text{int-snap}(\text{cond},\text{env})
\]
\[
\rightarrow \text{syst-cond-exec}(\text{cond},\text{env},\text{cbif},\text{type});
\]
\[
\text{int-snap}(\text{cond},\text{env})
\]
\[
\text{cont'd}
\]
9.4.2 Interpretation of condition action

Condition actions are interpreted in close analogy to parameterless procedures (5-50829)). A new block activation is established and after updating the condition bif part, the statement of the on-unit is interpreted. When this is finished the block activation is terminated by the instruction epilogue (5-14(20)), which makes use of the epilogue information installed in EI at block activation.

(24) \[ \text{int-cond}(ca, cbif, type) = \]
\[ (\text{s-e} ca) \]
\[ (\text{s-ei}) \]
\[ \quad \quad \quad (\text{s-block-action}, \]
\[ \quad \quad \quad \quad \text{s-free-set:[]} >, \]
\[ \quad \quad \quad \quad \text{s-task-set:[]} >) \]
\[ \quad \quad \quad \mu \quad \quad \quad \quad \text{s-bpp:s-bpp}(ca) \]
\[ \quad \quad \quad \text{s-da stack}() \]
\[ \quad \quad \quad \text{s-c epilogue;} \]
\[ \quad \quad \quad \quad \text{int-st}(s-on-unit(ca)); \]
\[ \quad \quad \quad \quad \text{update-bif}(s-cond(ca), cbif, type)); \]
\[ \quad \quad \quad \quad \text{test-on-unit}(s-on-unit(ca)) \]

Ref.: int-st 5-17(25), epilogue 5-14(20),
update-bif 9-8(15), stack 5-4(2)
9.4.3 The signal statement

The execution of a signal statement for a condition causes the condition to be immediately raised. As in the case of on and revert statements a check condition for a structure or cell name in a signal statement is expanded to a check condition for all base elements of a structure or alternative elements of a cell.

(25) \( \text{test-on-unit}(\text{st}) = \)
\[
\text{is-<>s-label-list}(\text{st}) \land \\
\neg (\text{is-return-st}(\text{st}_{\text{pr}}) \lor \text{is-p-if-st}(\text{st}_{\text{pr}}) \lor \\
\text{is-p-on-st}(\text{st}_{\text{pr}})) \lor \\
\text{is-p-group}(\text{st}_{\text{pr}}) \lor \text{is-p-proc-st-list}(\text{st}_{\text{pr}}) \\
\text{null} \\
\text{T} \rightarrow \text{error}
\]

---

where: \( \text{st}_{\text{pr}} = \text{s-prop-st}(\text{st}) \)
for: \( \text{is-p-st}(\text{st}) \)
Ref.: \( \text{is-p} 3-10(17) \)

(26) \( \text{int-signal-st}(\text{t}) = \)
\[
\text{is-corr-check}(\text{s-on-cond}(\text{t}),E,\text{AT}) \\
\text{call-check-cond}(\text{s-on-cond}(\text{t}),E,701) \\
\text{is-corr-cond}(\text{s-on-cond}(\text{t}),E,\text{AT}) \land \text{is-named-io-cond}(\text{s-on-cond}(\text{t})) \\
\text{call-cond-1}(\text{head}(\text{s-on-cond}(\text{t})),E,\text{<s-onfile:id>},701) \\
\text{is-corr-cond}(\text{s-on-cond}(\text{t}),E,\text{AT}) \\
\text{call-cond-1}(\text{head}(\text{s-on-cond}(\text{t})),E,\text{\textbackslash },701) \\
\text{T} \rightarrow \text{error}
\]

where: \( \text{id} = (\text{is-id}(\text{s-file}+\text{head}(\text{s-on-cond}(\text{t})))) \\
\text{s-title}+\text{attr}(\text{sel}<\text{s-file}+\text{head}(\text{s-on-cond}(\text{t}))>\{E\}(\text{AT})) \\
\text{T} \rightarrow \text{\textbackslash})
\]

for: \( \text{is-signal-st}(\text{t}) \)
Ref.: \( \text{is-corr-check} 9-6(8), \text{is-corr-cond} 9-6(9), \text{call-check-cond} 9-15(27), \text{call-cond-1} 9-12(19) \)
9.4.4 Special condition activations

9.4.4.1 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 list establishes an iterated condition call to an expanded reference list which contains references to all base elements of structures referenced in `reflist`.

\[(\text{27})\quad \text{call-check-cond}(\text{re}f\text{l}ist, \text{env}, \text{type}) = \]
\[
\begin{align*}
\text{is-<> (re}f\text{l}ist) \land \text{is-} \varnothing (\text{re}f\text{l}ist) & \rightarrow \text{null} \\
T & \rightarrow \\
\text{iterate-call-cond}(\text{exp-re}f\text{l}ist, \text{env}, \text{type}); \\
\text{exp-re}f\text{l}ist & \text{expand-re}f\text{l}ist(\text{re}f\text{l}ist, \text{env})
\end{align*}
\]

\[(\text{28})\quad \text{iterate-call-cond}(\text{e}r\text{l}, \text{env}, \text{type}) = \]
\[
\begin{align*}
\text{is-<> (e}r\text{l}) & \rightarrow \text{null} \\
T & \rightarrow \\
\text{iterate-call-cond}(\text{tail(e}r\text{l}), \text{env}, \text{type}); \\
\text{call-cond-1}(\text{head(e}r\text{l}), \text{env}, \varnothing, \text{type})
\end{align*}
\]

For: \text{is-ref-list(e}r\text{l)}

Ref.: \text{call-cond-1 9-12(19)}

\[(\text{29})\quad \text{expand-re}f\text{l}ist(\text{re}f\text{l}ist, \text{env}) = \]
\[
\begin{align*}
\text{is-<> (re}f\text{l}ist) & \rightarrow \text{pass(<<)} \\
T & \rightarrow \\
\text{conc}(\text{list-1, list-2}); \\
\text{list-1} & \text{expand-ref(head(re}f\text{l}ist), env), \\
\text{list-2} & \text{expand-re}f\text{l}ist(\text{tail(re}f\text{l}ist), env)
\end{align*}
\]
(30) \( \text{conc}(a, b) = \)  
\[ \text{PASS}: a \setminus b \]
for: \( \text{is-list}(a) \) & \( \text{is-list}(b) \)

(31) \( \text{expand-ref}(\text{ref}, \text{env}) = \)  
\[ \text{is.ptr-qual-ref}(\text{ref}) \rightarrow \text{pass}(<>) \]
\[ \text{is-BUILTIN}(\text{attr}_\text{ref}) \) & \( \text{mk-id}(\text{SUBSTR}) = \text{head}\cdot\text{is-id-list}(\text{ref}) \rightarrow \]
\[ \text{expand-ref}(\text{head}\cdot\text{is-arg-list}(\text{ref}), \text{env}) \]
\[ \text{is-BUILTIN}(\text{attr}_\text{ref}) \rightarrow \text{expand-reflist}(\text{is-arg-list}(\text{ref}), \text{env}) \]
\[ (\text{is-entry}(\text{attr}_\text{ref}) \) \cup \text{is-LABEL}(\text{attr}_\text{ref}) \) & \( \neg \text{is-PARAMs-scope}(\text{attr}_\text{ref}) \rightarrow \]
\[ \text{pass}(\text{ref}) \]
\[ \text{is-prop-var}(\text{attr}_\text{ref}) \) & \( \neg \text{is-PARAMs-scope}(\text{attr}_\text{ref}) \rightarrow \]
\[ \text{mk-ref-list}((\text{idl-list}) ; \]
\[ \text{idl-list}: \text{pass-expand-idl}(\text{is-id-list}(\text{ref}), \]
\[ \text{sub-da}(\text{da}_\text{ref}, <>, \text{eval-ql}(\text{tail}(\text{idl-list}(\text{ref})), \text{da}_\text{ref}))) \]
\[ \text{T} \rightarrow \text{pass}(<>) \]

where: \( \text{attr}_\text{ref} = \text{s-attr}(\text{sel}\cdot\text{id-list}(\text{ref})(\text{env})(\text{AT})) \)
\[ \text{da}_\text{ref} = \text{s-da}(\text{attr}_\text{ref}) \]
for: \( \text{is-ref}(\text{ref}) \)

Ref.: \( \text{sub-da} 7-34(89), \text{eval-ql} 7-27(66), \text{mk-id} 10-76(214) \)

(32) \( \text{expand-idl}(\text{idl}, \text{da}) = \)  
\[ \text{is-array}(\text{da}) \rightarrow \text{expand-idl}(\text{idl}, \text{s-elem}(\text{da})) \]
\[ \text{is-cell}(\text{da}) \cup \text{is-struct}(\text{da}) \rightarrow \]
\[ \text{order}(\text{da}) \]
\[ \text{CONC} \text{expand-idl}(\text{idl} \setminus <\text{s-id.elem}(\text{i}, \text{da})), \text{s-da.elem}(\text{i}, \text{da}) \]
\[ \text{is-scalar}(\text{da}) \rightarrow <\text{idl}> \]
for: \( \text{is-id-list}(\text{idl}), \text{is-da}(\text{da}) \)

Ref.: \( \text{order} 12-19(170) \)
(33) \texttt{mk-ref-list(idl-list)} =
\texttt{length(idl-list)}
\texttt{PASS: CUNC}_{i}^{n}(<s-id-list:elem(i, idl-list)>,
\texttt{<s-arg-list:<>>})
\texttt{for: is-id-list-list(idl-list)}

Note: This instruction constructs a reference list from a list of identifier lists by inserting empty argument lists.

9.4.4.2 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 allocates a dummy for the onsource condition builtin function, passes the name of the dummy and the integer required for interpreting the onchar condition builtin function to the cbif-argument of a \texttt{call-cond-1}, and after interpreting the call frees the storage of the dummy, returning the value of the dummy, i.e. the value of the character string whose conversion resulted in the condition situation, possibly modified through pseudovariables.

(34) \texttt{call-conv-cond(v, i, type)} =
\texttt{pass-op-val(op)}
\texttt{op: return-free(b);}
\texttt{call-cond-1(CONV, \varnothing , cbif, type);} \texttt{cbif: pass}_{i}^{n}(<s-onsource:b>,
\texttt{<s-onchar:i>>) \texttt{assign}(gen, val-op(ONSOURCE-DA,v));}
\texttt{gen: allocate-l(b, ONSOURCE-DA, PACKED);} \texttt{b: un-name}
\texttt{for: is-char-val-list(v), is-intg-val(i)}

Ref.: op-val 8-11(40), return-free 7-20(50), call-cond-1 9-12(19), assign 7-9(20), allocate-1 5-44(80), un-name 2-17(36), val-op 8-11(41)

(35) \texttt{ONSOURCE-DA =}
\texttt{\mu }_{i}^{n}(<s-base:CHAR>,
\texttt{<s-varying:T>,}
\texttt{<s-length:MAX-LENGTH>})

(36) \texttt{is-intg-val(MAX-LENGTH)}

Note: Implementation defined maximum length of varying character string
10. INPUT AND OUTPUT

This chapter contains the definition of all kind of data transmission to and from the external storage $ES$, and to and from the message part $M$. Transmission involving $ES$ is done by a file while the transmission to and from $M$ is either by display statements or by the comments of standard system actions for on-conditions.

The chapter is divided into six major parts. Section 10.1 defines the complete sets of file attributes which are the most important characterization of a file. The access to individual data sets in $ES$ by the title of a file and the basic accessing instructions are defined in section 10.2. The proper creation of a file by opening and the deletion of a file by closing is contained in section 10.3 which defines the open and close statement and all the cases of indirect opening and closing. Transmission by files having the record attribute is defined in section 10.4, including the read-, write-, rewrite-, locate-, delete- and unlock statement. Stream transmission by a get or put statement either by a file having the stream attribute or to or from a scalar character string variable is defined in section 10.5. This section also includes the definition of the standard system action of the check on-condition and the conversion from character string to arithmetic type. The definition of the display statement is given in section 10.6.

Apart from the relations to chapters 1, 2 and 12, chapter 10 is highly related with chapters 7 and 9. There is no immediate connection with chapter 3; from chapter 4 especially the instruction delete-event and the function ev-VR is used in 10.4 and 10.6; from chapter 5 the instruction int-contr-do is used in 10.5; from chapter 6 the instructions based-allocate and eval-set-gen are used in 10.4; from chapter 11 the functions new and pref are used in 10.5.

The above summary of the connections with other chapters is incomplete with intent to stress the major relations.
10.1. Attributes of a File

Any file identifier may have declared attributes and attributes specified in an open statement. The completion of the set of attributes occurs at opening. The function which yields the complete set of attributes from any set of attributes is defined in this section.

Metavariables

- `cسا` (csa) - a complete set of (file) attributes
- `fاس` (fas) - a set of file attributes
- `i`, `j` - integer values

(1) \[ \text{CSA-LIST} = \]

\[
\langle \text{STREAM, IN} \rangle, \\
\langle \text{STREAM, OUT} \rangle, \\
\langle \text{STREAM, OUT, PRINT} \rangle, \\
\langle \text{RECORD, IN, SEQ, BUF} \rangle, \\
\langle \text{RECORD, IN, SEQ, BUF, BACK} \rangle, \\
\langle \text{RECORD, IN, SEQ, BUF, KEYED} \rangle, \\
\langle \text{RECORD, IN, SEQ, BUF, KEYED, BACK} \rangle, \\
\langle \text{RECORD, OUT, SEQ, BUF} \rangle, \\
\langle \text{RECORD, OUT, SEQ, BUF, KEYED} \rangle, \\
\langle \text{RECORD, UPD, SEQ, BUF} \rangle, \\
\langle \text{RECORD, UPD, SEQ, BUF, KEYED} \rangle, \\
\langle \text{RECORD, IN, SEQ, UNBVF} \rangle, \\
\langle \text{RECORD, IN, SEQ, UNBVF, BACK} \rangle, \\
\langle \text{RECORD, IN, SEQ, UNBVF, KEYED} \rangle, \\
\langle \text{RECORD, IN, SEQ, UNBVF, KEYED, BACK} \rangle, \\
\langle \text{RECORD, OUT, SEQ, UNBVF} \rangle, \\
\langle \text{RECORD, OUT, SEQ, UNBVF, KEYED} \rangle, \\
\langle \text{RECORD, UPD, SEQ, UNBVF} \rangle, \\
\langle \text{RECORD, UPD, SEQ, UNBVF, KEYED} \rangle, \\
\langle \text{RECORD, IN, DIR, KEYED} \rangle, \\
\langle \text{RECORD, OUT, DIR, KEYED} \rangle, \\
\langle \text{RECORD, UPD, DIR, KEYED} \rangle, \\
\langle \text{RECORD, UPD, DIR, KEYED, EXCL} \rangle. \\
\]

(2) \[ \text{is-csa(fas)} = \]

\[ (\exists i)(\text{fas} = \text{elem}(i, \text{CSA-LIST}) \land \neg \text{is-}\varnothing(\text{fas})) \]

10.1
(3) \( \text{assoc-csa}(\text{fas}) = \)

\[
\exists t \quad \neg (\text{fas} \subseteq \text{csa}) \quad \Rightarrow \quad \emptyset \\
\text{is-csa}(\text{csa}) \\
T \quad \Rightarrow \quad \text{elem}(i_0, \text{CSA-LIST})
\]

where: \( \text{csa}_i = \text{elem}(i, \text{CSA-LIST}) \)
\( \text{csa}_j = \text{elem}(j, \text{CSA-LIST}) \)
\( i_0 = (\forall i)(i \neq 1 \quad \& \quad \text{fas} \subseteq \text{csa}_i \quad \& \quad (\forall j)(1 \leq j < i \quad \Rightarrow \quad \neg (\text{fas} \subseteq \text{csa}_j))) \)

Note: The function yields the complete set of attributes associated with the set of attributes \( \text{fas} \). If there is no such set the result is \( \emptyset \).

10.2 Access to the External Storage

10.2.1 Access to a data set by a title

Any member out of the set of titled data sets which are contained in \( \text{ES} \) is accessed by one of its identifying titles. If a given title does not uniquely identify a titled data set this must not lead to undefinedness since the undefinedfile on-condition caters for this case.

Metavariables

\begin{align*}
\text{title} & \quad \text{is-char-val-list} & \quad \text{a title which is evaluated during opening of a file} \\
\text{es-title} & \quad \text{is-title} & \quad \text{a title which identifies a titled data set} \\
\text{es} & \quad \text{is-es} & \quad \text{the external storage}
\end{align*}

(4) \( \text{is-title-match}(\text{title}, \text{es-title}) = \)

Note: This predicate is implementation defined.

The predicate is true if the title \( \text{es-title} \) of a titled data set matches the title "title" evaluated during opening.
10.2.2 Basic accessing instructions

This section contains the instructions for updating (i.e., accessing and changing) and inspecting (i.e., accessing without changing) a data set in external storage. Updating is defined by the instruction `upd-dataset` and its descendants, inspecting by the instruction `tk-data set`. The definitions apply to all files independent from stream or record transmission.
Abbreviations

fu-elem_u = (is-fu-elem*fu(FU) → u(FU), T → error)
Ref.: is-fu-elem 2-35(113)

fu-csa_u = s-csa(fu-elem_u)

fu-io-env_u = s-io-env(fu-elem_u)

fu-f_u = s-f(fu-elem_u)

fu-title_u = s-title(fu-elem_u)

fu-pos_u = s-position(fu-elem_u)

es-titled-dataset_u = es-titled-dataset(fu-title_u,ES)
Ref.: es-titled-dataset 10-4(7)

es-dataset_u = es-dataset(fu-title_u,ES)
Ref.: es-dataset 10-4(8)

es-ds-label_u = es-ds-label(fu-title_u,s,ES)
Ref.: es-ds-label 10-4(9)

es-descr_u = es-descr(fu-title_u,ES)
Ref.: es-descr 10-4(10)

Metavariables

u        is-n               a file union name
key      is-char-val-list_v a key of record data or Ω
ids      is-inner-dataset   a proper data set in inner representation
ds-pt    is-inner-dataset_v the data part of a data set or the label part in inner or outer representation
s        is-char-val-list_v a selector to a part of a data set
          s = s-data_v
          s = s-header_v
          s = s-trailer
opt      is-opt            truth or Ω
s-pt  see is-fu-elem  a selector applicable to an object which satisfies is-fu-elem
fu-elem-pt see is-fu-elem  an object selected by s-pt
pos  is-intg-val v is-∅  the position of a data set
f  is-f  a file name

(11)  \text{upd-data-set}(u,\text{key},\text{id}s) =
       \text{upd-es-transmit}(u,\text{key},\text{cipher}(\text{fu-csa}_u,\text{fu-io-env}_u,\text{es-descr}_u)(\text{id}s),s-\text{data})

Ref.: cipher 2-41(136)

(12)  \text{upd-es-transmit}(u,\text{key},\text{ds-pt},s) =
      \text{upd-fu-elem}(u,\text{s-transmit},\text{opt});
      \text{opt: transmit}(\text{fu-pos}_u,\text{key});
      \text{upd-es}(u,\text{ds-pt},s)

(13)  \text{upd-fu-elem}(u,\text{s-pt},\text{fu-elem-pt}) =
      s-fu:_μ(\text{FU},<\text{s-pt}:u;\text{fu-elem-pt}>)

(14)  \text{upd-es}(u,\text{ds-pt},s) =
      s-es:\{\text{es-titled-dataset}_u\} \cup
      \{μ(\text{es-titled-dataset}_u,<s:s-dataset:ds-pt>)\}

Note: This is the only instruction which updates external storage.

(15)  \text{transmit}(\text{pos},\text{key}) =

Note: This instruction depends additionally on extralingual parameters.

The value is either $\top$ or $\bot$. Any implementation returns the value $\top$
if and only if a transmission error has occurred.
(16) \( tk\text{-dataset}(u, \text{key}) = \)
\[ tk\text{-es-transmit}(u, \text{key}, s\text{-data}) \]

(17) \( tk\text{-es-transmit}(u, \text{key}, s) = \)
\[ \text{pass}(ds\text{-pt}); \]
\[ \text{upd-fu-elem}(u, s\text{-transmit}, \text{opt}); \]
\[ \text{opt\text{-transmit}}(fu\text{-pos}, \text{key}); \]
\[ ds\text{-pt}; tk\text{-es}(u, s) \]

(18) \( tk\text{-es}(u, s) = \)
\[ s = s\text{-data} \]
\[ \text{PASS: decipher}(fu\text{-csa}, fu\text{-io-env}, es\text{-descr})(es\text{-dataset}_u) \]
\[ s = s\text{-header} \]
\[ s = s\text{-trailer} \]
\[ \text{PASS: es-ds-label}_u \]
Ref.: decipher 2-41(135)
Note: This is the only instruction which inspects external storage.

(19) \( \text{verify-transmit}(u, \text{type}) = \)
\[ \text{raise-transmit-cond}(u, \bigcup, \text{type}) \]
for: \( \text{is\text{-intg-val}(type)} \)
Ref.: \( \text{raise-transmit-cond} \)[10-30(84)]
Note: This instruction inspects whether a transmission error has occurred previously for which no transmit on-condition call has been executed so far.

10.3 Opening and Closing of a File

Opening designates all actions which create a file while closing is the general term for all actions which delete a file. Dependent on how opening is achieved one can distinguish between explicit opening (by an open statement) and implicit opening (by an I/O statement other than open and close naming a file or by other statements if they access the standard system print file). Dependent on what is achieved by opening one can discern between proper opening (creation of a file by making new entries in FD and FU) and improper opening which in turn can be separated into erroneous opening (either undefined or involving a call to the undefined file on-condition) and opening of an already open file.

Closing is either by a close statement and proper or improper (which in turn is undefined or is closing of an already closed file) or is closing at task termination which is very similar to closing by a close statement.
The opening performed by an open statement is defined in section 10.3.1, implicit opening in section 10.3.2. The actions common for explicit and implicit opening are contained in section 10.3.3.

Closing is described in section 10.3.4.

Abbreviations

\[ e_{\cdot} = (i-s(t) \rightarrow \varnothing, \quad is-id\cdot s-file(t) \rightarrow sel(<s-file(t)>)(E)) \]

\[ ln-f_{\cdot} = (i-s(t) \rightarrow s\cdot stand\cdot print, \quad is-n(e_{\cdot}) \rightarrow e_{\cdot}(DN)) \]

\[ at-attr_{\cdot} = (i-file\cdot s\cdot attr\cdot e_{\cdot}(AT) \rightarrow s\cdot attr\cdot e_{\cdot}(AT)) \]

Note: In this abbreviation and in the sequel the notion of a single alternative is used only to define the range of the abbreviation.

\[ at-title_{\cdot} = (i-s(t) \rightarrow chl\cdot mk-id(SYSPRINT), \quad is-id\cdot s\cdot title(at-atr_{\cdot}) \rightarrow chl\cdot s\cdot title(at-atr_{\cdot})) \]

Ref. : chl 10-76(215) \quad mk-id 10-76(214)

\[ at-io-env_{\cdot} = (i-s(t) \rightarrow DEF\cdot ENVIRONMENT, \quad is-io-opt\cdot s\cdot io\cdot env(at-atr_{\cdot}) \rightarrow s\cdot io\cdot env(at-atr_{\cdot})) \]

Note: DEF\cdot ENVIRONMENT is the default environment attribute of the standard system print file (is-io-opt(DEF\cdot ENVIRONMENT))

\[ at-file-atr_{\cdot} = (i-file\cdot attr-set\cdot s\cdot file\cdot attr(at-atr_{\cdot}) \rightarrow s\cdot file\cdot attr(at-atr_{\cdot})) \]

\[ fd-elem_{\cdot} = (i-s\cdot fd\cdot elem\cdot dn-f_{\cdot}(FD) \vee is-\varnothing \cdot dn-f_{\cdot}(FD) \rightarrow dn-f_{\cdot}(FD)) \]

Ref. : is-fd-elem 2-34(110)

\[ fd-fu_{\cdot} = (i-s\cdot s\cdot fu(fd-elem_{\cdot}) \rightarrow s\cdot fu(fd-elem_{\cdot}), \quad is-\cdot stand\cdot print\cdot s\cdot fu(fd-elem_{\cdot}) \& is-n\cdot s\cdot fu\cdot s\cdot stand\cdot print(fd) \rightarrow s\cdot fu\cdot s\cdot stand\cdot print(fd)) \]

Ref. : is-n 2-17(35)

\[ fd-orig_{\cdot} = (i-own\cdot s\cdot orig(fd-elem_{\cdot}) \vee is-\cdot orig(fd-elem_{\cdot}) \rightarrow s\cdot orig(fd-elem_{\cdot})) \]

\[ fu-csa_{\cdot} = (i-csa\cdot s\cdot csa\cdot fd-fu_{\cdot}(FU) \rightarrow s\cdot csa\cdot fd-fu_{\cdot}(FU), \quad T \rightarrow error) \]

Ref. : is-csa 10-2(2)
fu-mu = (is-fu-mu FU) \rightarrow u(FU),
    \text{T} \rightarrow \text{error}
\text{Ref.: is-fu-mu 2-35(113)}

fu-csa = s-csa(fu-mu)

es-titled-dataset \text{title} = es-titled-dataset(title, ES)
\text{Ref.: es-titled-dataset 10-4(7)}

es-dataset \text{title} = es-dataset(title, ES)
\text{Ref.: es-dataset 10-4(8)}

es-desc \text{title} = es-desc(title, ES)
\text{Ref.: es-desc 10-4(10)}

is-ext-sysprint = (is-\emptyset(t) \rightarrow F,
    \text{T} \rightarrow \text{EXT-scope(at-attr)} & \text{at-title} =
    \text{chl} \cdot \text{mk-id(SYSPRINT)})
\text{Ref.: chl 10-76(215), mk-id 10-76(214)}

open-fas = (is-open(t) \rightarrow s-open-atr(t),
    \text{T} \rightarrow \text{default-attr-set(t)})

merged-csa = assoc-csa(at-file-atr \cup open-fas)
\text{Ref.: assoc-csa 10-3(3)}

new-csa = (is-ext-sysprint & \{ \text{STREAM}, \text{OUT} \} \subseteq \text{merged-csa} \rightarrow
    \text{merged-csa} \cup \{ \text{PRINT} \},
    \text{T} \rightarrow \text{merged-csa})

\textbf{Metavariables}

alt \quad \text{is-T } \lor \text{is-F}
\text{an alternative between truth or falsity indicating undefined file condition to be raised or not}

alt \text{ is-alt-list}
\text{a list of alternatives used similar as alt}

is-alt = \text{is-T } \lor \text{is-F}
**10.3.1 Explicit opening**

This section together with 10.3.3 defines the interpretation of the open statement.

**Metavariables**

- `to` is-open-st the text of an open statement
- `t` is-open the text of an element of an open statement
- `tl` is-open-list the text which is a list of elements of an open statement
(20) \texttt{int-open-st(t, to) =}
\begin{align*}
\text{int-undef(head(tl_0),alt,tail(tl_0),altl);} \\
\{\text{alt:\ init-expl-open(head(tl_0))}\} \cup \\
\{\text{elem(i)(altl):init-expl-open(elem(i,tail(tl_0))) | 1 \leq l \leq \text{length\:tail(tl_0)}\}
\end{align*}

where: \( tl_0 = s\text{-open-list}(t, to) \)

(21) \texttt{init-expl-open(t) =}
\begin{align*}
\text{int-expl-open(t);} \\
\text{verify-file-name(<s-file(t)>)}
\end{align*}

(22) \texttt{verify-file-name(idl) =}
\begin{align*}
\text{is-n(sel(idl)(E)) \& is-file s\text{-attr(sel(idl)(E)(A\tilde{I})) \&} \\
\text{is-f(sel(idl)(E)(DN))} \rightarrow \text{null} \\
\text{T} \rightarrow \text{error}
\end{align*}

Ref.: is-n 2-17(35), is-f 2-35(lll)

(23) \texttt{int-undef(t,alt,tl,altl) =}
\begin{align*}
is-<>(tl) \& alt \rightarrow \text{call-undef-cond(dn-f_t,at-title_t,801)} \\
is-<>(tl) \rightarrow \text{null} \\
alt \rightarrow \\
\text{int-undef(head(tl),head(altl),tail(tl),tail(altl));} \\
\text{call-undef-cond(dn-f_t,at-title_t,801)} \\
\text{T} \rightarrow \text{int-undef(head(tl),head(altl),tail(tl),tail(altl))}
\end{align*}

(24) \texttt{call-undef-cond(f,at-title,\text{type}) =}
\begin{align*}
\text{call-cond-1}(\mu_0(<s\text{-io-cond:UNDF}>,<s\text{-f\text{-}den:}\text{f}>),\Omega, \\
\mu_0(<s\text{-onfile:mk-id(at-title)>},\text{type})
\end{align*}

for.: is-n(f) \& is-intg-val(type)

Ref.: call-cond-1 y-1\alpha(1y), mk-id 10-70(214)

Note: The undefinedfile on-condition must be handled separately from all other input and output on-conditions (see \texttt{call-\text{io-cond}}) since it is not related with an open file.
(25) \[ \text{int-expl-open}(t) = \]
\[ \begin{align*}
&\quad \text{open-associate}(t,\text{title},\text{new-csa}_t,\text{expl-opt}); \\
&\quad \text{s-ident}(\text{expl-opt}) :\text{eval-ident}(\text{s-ident}(t),\text{new-csa}_t), \\
&\quad \text{s-pagesize}(\text{expl-opt}) :\text{eval-intg-expr}(\text{s-pagesize}(t),E), \\
&\quad \text{s-linesize}(\text{expl-opt}) :\text{eval-intg-expr}(\text{s-linesize}(t),E), \\
&\quad \text{title} :\text{select-title}(\text{s-title}(t),\text{at-title}_t) \\
\end{align*} \]

Ref.: \text{eval-intg-expr} 7-21(52), \text{open-associate} 10-17(39)

Note: Specified options are evaluated without being compared with the complete set of attributes and without to contribute to the attributes.

(26) \[ \text{is-expl-opt} = (\langle \text{s-ident}:\text{is-ds-label} \rangle, \\
\quad \langle \text{s-pagesize}:\text{is-intg-val} \vee \text{is-}Q \rangle, \\
\quad \langle \text{s-linesize}:\text{is-intg-val} \vee \text{is-}Q \rangle) \]

(27) \[ \text{is-ds-label} = \text{is-gen} \vee \text{is-ps-gen} \vee \text{is-char-val-list} \vee \text{is-}Q \]

Ref.: \text{is-gen} 2-27(96), \text{is-ps-gen} 7-11(26), \text{is-char-val} 8-4(9)

(28) \[ \text{eval-ident}(\text{id},\text{csa}) = \]
\[ \begin{align*}
&\quad \text{is-}Q(\text{id}) \vee \text{OUT} \notin \text{csa} \& \text{is-ref}(\text{id}) \quad \text{eval-opt-gen}(\text{id}) \\
&\quad \text{OUT} \notin \text{csa} \quad \text{conv-char-string}(\text{id}) \\
&\quad \text{T} \quad \text{error} \\
\end{align*} \]

Note: The ident option is evaluated in any case where it is specified. If it is a reference and the attributes are either update or input, only the generation of the reference is evaluated.

(29) \[ \text{eval-opt-gen}(\text{ref}) = \]
\[ \begin{align*}
&\quad \text{is-}Q(\text{ref}) \quad \text{null} \\
&\quad \text{T} \quad \text{eval-lp}(\text{ref}) \\
\end{align*} \]

Ref.: \text{eval-lp} 7-4(5)

(30) \[ \text{conv-char-string}(\text{expr}) = \]
\[ \begin{align*}
&\quad \text{is-}Q(\text{expr}) \quad \text{null} \\
&\quad \text{T} \quad \text{pass-op-val}(\text{op}-1); \\
&\quad \quad \text{op}-1:\text{convert}(\text{CHAR-DA},\text{op}); \\
&\quad \quad \text{op}:\text{eval-expr}(\text{expr},E) \\
\end{align*} \]

Ref.: \text{eval-expr} 7-18(43), \text{convert} 8-30(118), \text{op-val} 8-11(40), \text{CHAR-DA} 8-13(58)
(31) \[
\text{select-title(opt-title,at-title)} = \\
\text{is-Ω(opt-title) \rightarrow PASS:at-title} \\
\text{is-expr(opt-title) \rightarrow conv-char-string(opt-title)}
\]

Note: Depending on whether the title option has been specified or not the title is returned which is used to identify the data set in external storage.

10.3.2 Implicit opening

10.3.2.1 Opening with a file identifier

Implicit opening as it occurs in the interpretation of I/O statements naming a file is defined in this section. Note that all get and put statements are considered to name a file if they do not name a string.

Metavariabes

\( t \)
- \( \text{is-delete-st(t) v is-file-get(t) v the text of a statement} \)
- \( \text{is-locate-st(t) v is-file-put(t) v causing implicit} \)
- \( \text{is-read-st(t) v is-rewrite-st(t) v opening} \)
- \( \text{is-unlock-st(t) v is-write-st(t)} \)

(32) \[
\text{int-impl-open(t)} = \\
\text{verify-impl-open(t);} \\
\text{open-associate}(t, \text{at-title}_t, \text{new-csa}_t, \text{Ω}); \\
\text{verify-file-name}(<s-file(t)>)
\]

Ref.: open-associate 10-17(39), verify-file-name 10-11(22)

(33) \[
\text{default-attr-set(t)} = \\
\text{is-file-get(t)} \rightarrow \{\text{STREAM, IN}\} \\
\text{is-file-put(t)} \rightarrow \{\text{STREAM, OUT}\} \\
\text{is-read-st(t)} \rightarrow \{\text{EXCL \_ at-file-attr}_t \rightarrow \{\text{RECORD}, \text{OUT}\}, \text{UPD \_ at-file-attr}_t \rightarrow \{\text{RECORD}\}, \text{T} \rightarrow \{\text{RECORD, IN}\}\}
\]
is-rewrite-st(t) \lor is-delete-st(t) \rightarrow \{UPD\}
is-unlock-st(t) \rightarrow \{EXCL\}
is-locate-st(t) \rightarrow \{OUT, BUF\}
is-write-st(t) \rightarrow 
\begin{align*}
(EXCL \in \text{at-file-attr}_t \lor \\
UPD \in \text{at-file-attr}_t) \rightarrow \{\text{RECORD}\},
\end{align*}
T \rightarrow \{\text{RECORD, OUT}\}

Note: The default attributes do not depend on specific options of the statement causing implicit opening. This can cause inconsistency between the derived complete set of attributes and the statement. For this and all other reasons which may cause inconsistency, implicit opening ends normally with a consistency test by \texttt{verify-impl-open}(t).

(34) \texttt{verify-impl-open}(t) = 
\begin{align*}
\texttt{is-\neg \exists (fd-elem_t)} \rightarrow \\
\texttt{error; } \\
\texttt{call-cond(\text{ERROR, 802})} \\
\texttt{STREAM} \in \texttt{fu-csa}_t \land \texttt{is-consistent-stream-file}(t, \texttt{fu-csa}_t) \lor \\
\texttt{RECORD} \in \texttt{fu-csa}_t \land \texttt{is-consistent-record-file-1}(t, \texttt{fu-csa}_t) \land \\
\texttt{is-consistent-record-file-2}(t, \texttt{fu-csa}_t) \rightarrow \texttt{PASS: fd-fu}_t
\end{align*}
T \rightarrow \texttt{error}

Ref.: \texttt{call-cond} 9-11(18)

Note: The test on record files is performed in two steps only to achieve higher transparency. In the non-erroneous case the instruction returns the file union name.

(35) \texttt{is-consistent-stream-file}(t, \texttt{csa}) =
\begin{align*}
\texttt{is-file-get}(t) \rightarrow \texttt{IN} \in \texttt{csa} \land \\
(is-<\leftrightarrow s\texttt{-spec}(t) \lor \neg is-\neg \exists s\texttt{-skip}(t))
\end{align*}
is-file-put(t) \rightarrow 
\begin{align*}
\texttt{OUT} \in \texttt{csa} \land \\
(PRINT \notin \texttt{csa} \lor \neg is-\neg \exists s\texttt{-skip}(t) \lor \\
is-\neg \exists s\texttt{-page}(t) \land is-\neg \exists s\texttt{-line}(t) \land \\
is-\neg \exists s\texttt{-spec(t)} \lor \\
\neg is-\neg \exists s\texttt{-page}(t) \lor \neg is-\neg \exists s\texttt{-line}(t) \lor \neg is-\neg \exists s\texttt{-skip}(t)
\end{align*}

10.3.2.1
(36) \( \text{is-consistent-record-file-1}(t, \text{csa}) = \)

\[
\begin{align*}
\text{(KEYED} & \text{csa} \supset \\
\text{is-} & \text{Q}^\circ \text{s-key}(t) \text{ & is-} \text{Q}^\circ \text{s-keyto}(t) \text{ & is-} \text{Q}^\circ \text{s-keyfrom}(t)), \& \\
(\text{BUF} & \text{csa} \supset \text{is-} \text{Q}^\circ \text{s-event}(t)), \& \\
(\neg \text{is-} & \text{Q}^\circ \text{s-key}(t) \supset \text{is-} \text{Q}^\circ \text{s-keyto}(t)) \& \\
(\text{is-} & \text{Q}^\circ \text{s-key}(t) \supset \text{is-} \text{Q}^\circ \text{s-nolock}(t))
\end{align*}
\]

Note: This test does not depend on specific statements; it deals with options which are generally conflicting and with options which are generally in conflict with the presence or absence of certain attributes.

(37) \( \text{is-consistent-record-file-2}(t, \text{csa}) = \)

\[
\begin{align*}
\text{is-delete-st}(t) & \rightarrow \text{UPD} \text{csa} \& (\text{DIR} \text{csa} = \text{is-} \text{Q}^\circ \text{s-key}(t)) \\
\text{is-locate-st}(t) & \rightarrow \text{OUT} \text{csa} \& \text{BUF} \text{csa} \& \\
\text{(KEYED} & \text{csa} \supset \neg \text{is-} \text{Q}^\circ \text{s-keyfrom}(t)) \\
\text{is-into-read}(t) & \rightarrow \text{OUT} \text{csa} \& (\text{SEQ} \text{csa} \supset \text{is-} \text{Q}^\circ \text{s-keyto}(t)) \& \\
\text{(EXCL} & \text{csa} \supset \text{is-} \text{Q}^\circ \text{s-nolock}(t)) \\
\text{is-set-read}(t) & \rightarrow \text{OUT} \text{csa} \& \text{BUF} \text{csa} \\
\text{is-ignore}(t) & \rightarrow \text{OUT} \text{csa} \& \text{DIR} \text{csa} \\
\text{is-rewrite-st}(t) & \rightarrow \text{UPD} \text{csa} \& (\text{DIR} \text{csa} = \text{is-} \text{Q}^\circ \text{s-key}(t)) \& \\
\text{(BUF} & \text{csa} \supset \neg \text{is-} \text{Q}^\circ \text{s-from}(t)) \\
\text{is-unlock-st}(t) & \rightarrow \text{EXCL} \text{csa} \\
\text{is-write-st}(t) & \rightarrow \text{IN} \text{csa} \& (\text{SEQ} \text{csa} \supset \text{UPD} \text{csa}) \& \\
\text{(KEYED} & \text{csa} \supset \neg \text{is-} \text{Q}^\circ \text{s-keyfrom}(t))
\end{align*}
\]

Note: This predicate should be considered together with the predicate \( \text{is-consistent-record-file-1}(t, \text{csa}) \).
10.3.2.2 Opening of the standard system print file

(38) \text{int-stand-print-open} =
\text{open-associate(\omega, \text{stand}_t, [STREAM,OUT,PRINT], \omega)}

where: \text{stand}_t = \text{chl}\cdot \text{mk-id}(\text{SYSPRINT})

Ref.: \text{open-associate 10-17(39), chl 10-76(215), mk-id 10-76(214)}

Note: This instruction opens the standard system print file as a consequence of the copy option of a get statement or of the standard system action of the check on-condition. Opening of the standard system print file may arise from explicit opening or implicit opening with a file name also. The standard system input file has no special treatment (in the interpreter).

10.3.3 Central part of opening

This section contains the definition of the actions common for explicit and implicit opening. It also contains the updating of data set labels, which also occurs in connection with closing.

\textbf{Metavariables}

\begin{align*}
\text{t} & \quad \text{is-open}(t) \lor \text{is-delete-st}(t) \lor \\
& \quad \text{is-file-get}(t) \lor \text{is-locato-st}(t) \lor \\
& \quad \text{is-file-put}(t) \lor \text{is-read-st}(t) \lor \\
& \quad \text{is-rewrite-st}(t) \lor \text{is-unlock-st}(t) \lor \\
& \quad \text{is-write-st}(t) \lor \text{is-\Omega}(t) \\
\text{i} & \quad \text{is-intg-val}(i) \lor \text{is-\Omega}(i)
\end{align*}

\text{text of a statement or element of an open statement which causes opening; } \Omega \text{ means opening of the standard system print file an integer or } \Omega \text{ representing linesize or pagesize}
(39) \( \text{open-associate}(t, \text{title}, \text{new-csa}, \text{expl-opt}) = \)

\[
\neg \text{is-}Q(\text{new-csa}) \lor \neg \text{is-}Q(\text{es-titled-dataset title}) \lor \\
\neg \text{is-ds-descr(} \text{es-descr title}) \lor \neg \text{is-outer-dataset(} \text{es-dataset title}) \lor \\
\neg \text{is-proper-dataset(} \text{new-csa}) \\
(\text{decipher(} \text{new-csa, at-io-env}_{t, \text{es-descr title}})(\text{es-dataset title})) \rightarrow \\
\text{invalid-open}(t) \\
\neg \text{is-Q(} \text{fd-element}_{t}) \land \neg \text{is-OWN(} \text{fd-orig}_{t}) \lor \text{DIR} \in \text{fu-csa}_{t} \land \text{DIR} \in \text{new-csa}) \rightarrow \\
\text{PASS} \lor \\
\neg \text{is-Q(} \text{fd-element}_{t} \rightarrow \text{error} \\
\text{is-Q(} \text{fd-element}_{t} \rightarrow \text{open}(u, t, \text{title}, \text{new-csa}, \text{expl-opt}); \\
\text{u:un-name} \\
\text{Ref.: un-name 2-17(36), is-proper-dataset 2-40(128), is-ds-descr 2-39(125),} \\
\text{is-outer-dataset 2-39(126), decipher 2-41(135)} \\
\text{Note: The four alternatives are invalid opening (which normally leads to an} \\
\text{undefined file on-condition call), opening of an already open file for} \\
\text{direct access and/or in the task owning the file, opening of an inherited file for stream or sequential transmission, and proper opening} \\
\text{of a file (which may be the standard system print file).} \\
\]

(40) \( \text{invalid-open}(t) = \)

\[
\text{is-open}(t) \rightarrow \text{PASS} \lor \\
\text{is-delete-st}(t) \lor \text{is-locate-st}(t) \lor \text{is-read-st}(t) \lor \text{is-rewrite-st}(t) \lor \\
\text{is-unlock-st}(t) \land \text{is-write-st}(t) \lor \text{is-file-get}(t) \lor \text{is-file-put}(t) \rightarrow \\
\text{call-undf-cond}(dn-f_{t}, \text{at-title}_{t}, 803) \\
\text{is-Q}(t) \rightarrow \text{error} \\
\text{Ref.: call-undf-cond 10-11(24)} \\
\text{Note: The three alternatives concern explicit opening, implicit opening with} \\
a \text{file identifier and implicit opening of the standard system print file.} \\
\]

(41) \( \text{open}(u, t, \text{title}, \text{new-csa}, \text{expl-opt}) = \)

\[
\text{pass}(F); \\
\text{verify-transmit}(u, 804); \\
\text{und-ds-label(OPEN, u, s-ident(t), s-ident(expl-opt))}; \\
\text{open-fd-l}(u, t, \text{new-csa}), \\
\text{open-fu}(u, t, \text{title}, \text{new-csa}, \text{expl-opt}) \\
\text{Ref.: verify-transmit 10-7(19)} \\
\text{cont'd} \]
Note: The transmission of a file mark for output files during opening can be regarded as a property of external storage mapping.

(42) \(\text{open-fd-1}(u, t, \text{new-csa}) = \)

\[- (\text{is-ext-sysprint}_t \& \text{PRINT} \in \text{new-csa}) \rightarrow \text{open-fd-2}(u, dn-f_t)\]

\(\text{is} = \Phi(u_p) \rightarrow \)

null;

\(\text{open-fd-2}(u, s-\text{stand-print})\),

\(\text{open-fd-2}(\text{STAND-PRINT}, dn-f_t)\)

\(- \text{is} = \Phi(u_p) \rightarrow \text{error}\)

where: \(u_p = s-fu \cdot s-\text{stand-print}(FD)\)

Note: The standard system print file must have been opened by the second alternative of the instruction if it should be accessible by subsequent put statements.

(43) \(\text{open-fd-2}(u, f) = \)

\(s-fd: \mu(FD, f)(<s-fu; u>, <s-\text{orig: OWN}>))\)

for: \(\text{is-n}(u) \lor \text{is-\text{STAND-PRINT}}(u)\)

(44) \(\text{open-fu}(u, t, \text{title}, \text{new-csa}, \text{expl-opt}) = \)

\((\exists u-1)(- \text{is} = \Phi, s-\text{title} = u-1(FU) \&\)

\((\text{es-titled-dataset}(\text{title} = \text{es-titled-dataset}(s-\text{title} = u-1(FU), ES) \rightarrow\)

\(\text{DIR} \in fu-csa_u \& \text{DIR} \in \text{new-csa})\) \rightarrow \)

\(s-fu: \mu(FU; u: \mu(fu-elem(t, title, new-csa, expl-opt);\)

\(<s-\text{csa: new-csa}, >\)

\(<s-\text{at-title: at-title}_t, >\)

\(<s-\text{io-env: at-\text{io-env}_t}, >\)

\(<s-f: dn-f_t>, >\)

\(<s-\text{title: title}>)\))

\(T \rightarrow \text{error}\)

Ref.: es-titled-dataset 10-4(7)

Note: Access to one and the same data set by more than one file is erroneous except when all files are direct.
(45) \( \text{fu-elem}(t, \text{title}, \text{new-csa}, \text{expl-opt}) = \)

\[
\text{STREAM } \epsilon \text{ new-csa} \rightarrow \\
\mu_0\left(\langle \text{s-position:fu-pos}(t, \text{title}, \text{new-csa})>, \langle \text{s-column:} >, \langle \text{s-linesize:fu-lsz}(\text{s-linesize}(<\text{expl-opt}), \text{new-csa}>)>, \langle \text{s-line:line_1}>, \langle \text{s-pag esize:fu-psz}(\text{s-pagesize}(<\text{expl-opt}), \text{new-csa}>)\rangle\right)
\]

\[
T \rightarrow \\
\mu_0\left(\langle \text{s-position:fu-pos}(t, \text{title}, \text{new-csa})>, \langle \text{s-io:ev:ev_1} >\right)
\]

where: \( \text{line}_1 = (\text{PRINT } \epsilon \text{ new-csa} \rightarrow \text{bintg-op}(1), T \rightarrow \emptyset) \)  
\( \text{ev}_1 = (\text{BUF } \epsilon \text{ new-csa} \rightarrow \{ \}, T \rightarrow \emptyset) \)

Ref.: \( \text{bintg-op } 8-11(43) \)

Note: The current line (\( \text{line}_1 \)) has the form of an operand instead of a value since a built-in function can access it.

(16) \( \text{fu-pos}(t, \text{title}, \text{new-csa}) = \)

\[
\text{DIR } \epsilon \text{ new-csa } \rightarrow \emptyset \\
\text{BACK } \epsilon \text{ new-csa } \\
\text{length} \times (\text{decipher}(\text{new-csa}, \text{atio-env}_t, \text{es-descr}_t, \text{es-desctitle}))(\text{es-dataset}_ttitle) \\
\text{OUT } \epsilon \text{ new-csa } \\
\text{open-intg-test}(\text{pos}_i, \text{out-pos}(\text{new-csa}, \text{atio-env}_t, \text{es-titled-dataset}_ttitle))
\]

\[
T \rightarrow \text{pos}_i
\]

where: \( \text{pos}_i = (\text{RECORD } \epsilon \text{ new-csa } \rightarrow 0, T \rightarrow 1) \)

Ref.: \( \text{decipher } 2-41(135) \)

Note: The function \( \text{out-pos} \) is implementation-defined and yields the positioning for non-direct output data sets.

(47) \( \text{open-intg-test}(i, k) = \)

\[
\text{is-intg-val}(k) \& i \leq k \rightarrow k \\
T \rightarrow \text{error}
\]
(48) fu-lsz(i,new-csa) =

\[
\begin{align*}
\text{OUT} &\notin \text{new-csa} \land \text{STREAM} \notin \text{new-csa} \longrightarrow \Omega \\
T &\longrightarrow \text{open-intg-test}(1,lsz_1)
\end{align*}
\]

where: \( lsz_1 = (\text{is-}Ω(1) \longrightarrow \text{DEF-LINESIZE}, T \longrightarrow 1) \)

Note: DEF-LINESIZE is the implementation-defined default linesize.

(49) fu-psz(i,new-csa) =

\[
\begin{align*}
\text{PRINT} &\notin \text{new-csa} \longrightarrow \Omega \\
T &\longrightarrow \text{open-intg-test}(1,psz_1)
\end{align*}
\]

where: \( psz_1 = (\text{is-}Ω(i) \longrightarrow \text{DEF-PAGESIZE}, T \longrightarrow i) \)

Note: DEF-PAGESIZE is the implementation-defined default pagesize.

(50) upd-ds-label(st,u,ident,ds-label) =

\[
\begin{align*}
is-Ω(\text{ident}) &\rightarrow \text{null} \\
\text{OUT} &\notin \text{fu-csa}_u \land \text{is-CHAR-gen-base}(ds-label) \longrightarrow \\
\text{call-check-cond}(\langle \text{ident},E,305\rangle); \\
\text{convert-assign}(ds-label,op); \\
\text{op:pass-val-op}(da_1,cl); \\
\text{cl:tk-es-transmit}(u,Ω,s_u) \\
\text{OUT} &\notin \text{fu-csa}_u \\
\text{upd-es-transmit}(u,Ω,ds-label,s_u) \\
T &\rightarrow \text{error}
\end{align*}
\]

where: \( da_1 = \text{char-da\_length}(cl) \)

\( s_u = (\text{BACK} ε \text{fu-csa}_u \equiv \text{is-CLOSE}(st) \longrightarrow \text{s-header}, T \longrightarrow \text{s-trailer}) \)

Ref.: \text{convert-assign 7-9(19), call-check-cond 9-15(27),}
\text{tk-es-transmit 10-6(17), upd-es-transmit 10-6(12), char-da 8-13(60),}
\text{val-op 8-11(41)}

Note: This instruction is used for opening and closing.
(51) \[ \text{gen-base}(g) = \]
\[
is\text{-string-type}\cdot s\text{-da}(g) \rightarrow \text{str-base}\cdot s\text{-da}(g) \]
\[
s\text{-id}(g) = \text{mk-id}(\text{SUBSTR}) \rightarrow \text{gen-base}\cdot \text{elem}(1, s\text{-arg-list}(g)) \]
\[
s\text{-id}(g) = \text{mk-id}(\text{UNSPEC}) \rightarrow \text{BIT} \]
\[
s\text{-id}(g) = \text{mk-id}(\text{ONSOURCE}) \lor s\text{-id}(g) = \text{mk-id}(\text{ONCHAR}) \rightarrow \text{CHAR} \]
\[
T \rightarrow Q \]

for \( \text{is-gen}(g) \lor \text{is-ps-gen}(g) \lor \text{is-q}(g) \)

Ref.: is-string-type 8-23(93), str-base 8-39(156), mk-id 10-76(214),
\text{gen-base} 10-21(51)

Note: This function yields the elementary objects \text{CHAR, BIT, or } Q \text{ whether the generation or pseudo-generation is of character, bit or other type, respectively.}

10.3.4 Closing

Closing by a close statement is defined by \text{int-close-st} and its
descendants while closing at task termination is defined by \text{close}.

Abstractions

\[
\text{fd-fu}_f = s\text{-fu}\cdot f(FD) \]
\[
\text{fu-elem}_f = (is\text{-fu}\cdot \text{elem}\cdot \text{fd-fu}_f(FU) \rightarrow \text{fd-fu}_f(FU), \]
\[
T \rightarrow \text{error} \]

Ref.: is-fu\cdot\text{elem} 2-35(113)
\[
\text{fu-ev}_f = s\text{-io}\cdot\text{ev}(\text{fu-elem}_f) \]

Metavariables

\( tc \) \hspace{1cm} \text{is-close-st} \hspace{1cm} \text{the text of a close statement}
\( t \) \hspace{1cm} \text{is-close} \hspace{1cm} \text{the text of an element of a close statement}

(52) \[ \text{int-close-st} (tc) = \]
\[
\neg is\leftrightarrow(tc_0) \rightarrow \null; \]
\[
\text{init-expl-close}(\text{elem}(1, tc_0)); \]
\[
\text{verify-file-name}( <s\text{-file}\cdot\text{elem}(1, tc_0)> | 1 \leq \text{length}(tc_0) ) \}
\[
T \rightarrow \text{error} \]

where: \( tc_0 = s\text{-close-list}(tc) \)

Ref.: \text{verify-file-name} 10-11(22)
(53) $\text{init-expl-close}(t) =$
\[
\begin{align*}
&\,\text{is-}\emptyset (fd-\text{elem}_t) \rightarrow \text{null} \\
&\,T \rightarrow \text{close-dissociate}(fd-\text{fu}_t, f, fd-\text{orig}_t, s-\text{ident}(t))
\end{align*}
\]

(54) $\text{close-dissociate}(u, f, \text{orig}, \text{ident}) =$
\[
\begin{align*}
&\,\text{is-OWN}(\text{orig}) \rightarrow \\
&\,\text{close}(f); \\
&\,\text{verify-transmit}(u, 806); \\
&\,\text{upd-ds-label}(\text{CLOSE}, u, \text{ident}, \text{ds-label}); \\
&\,\text{ds-label: eval-ident}(\text{ident}, fu-\text{csa}_u) \\
&\,\text{is-INH}(\text{orig}) \rightarrow \text{error}
\end{align*}
\]

Ref.: $\text{verify-transmit}$ 10-7(19), $\text{upd-ds-label}$ 10-20(50), $\text{eval-ident}$ 10-12(28)

Note: The transmission of a filemark for output files during closing can be regarded as a property of external storage mapping.

(55) $\text{close}(f) =$
\[
\begin{align*}
&\,\text{is-STAND-PRINT}(fd-\text{fu}_f) \rightarrow \\
&\,s-\text{fd}: \delta(\text{FD}; s-\text{stand-print}, f) \\
&\,s-\text{fu}: \delta(\text{FU}; u_p) \\
&\,\text{is-n}(fd-\text{fu}_f) \rightarrow \\
&\,\text{close-fd-fu}(fd-\text{fu}_f, f); \\
&\,\text{delete-buffer-event}(fd-\text{fu}_f, fu-\text{ev}_f);
\end{align*}
\]

where: $u = s-\text{fu} \cdot s-\text{stand-print}(\text{FD})$

Note: At termination of a specific task the instruction is executed for all files owned by this task which are still open.

(56) $\text{delete-buffer-event}(u, \text{ev}) =$
\[
\begin{align*}
&\,\text{is-}\emptyset (\text{ev}) \rightarrow \text{execute-locate}(u) \\
&\,T \rightarrow \\
&\,\text{execute-locate}(u); \\
&\{\text{delete-event}(\text{ten}, \text{ABNL}) \mid \text{ten} \in \text{ev} \& \neg \text{is-}\emptyset \cdot \text{ten}(\text{PA})\}
\end{align*}
\]

for: $\text{is-}\emptyset (\text{ev}) \lor \text{is-n-set}(\text{ev})$

Ref.: $\text{execute-locate}$ 10-42(112), $\text{delete-event}$ 4-20(49)

(57) $\text{close-fd-fu}(u, f) =$
\[
\begin{align*}
&\,s-\text{fd}: \delta(\text{FD}; f) \\
&\,s-\text{fu}: \delta(\text{FU}; u)
\end{align*}
\]

10.3.4
10.4 Record Input and Output

This chapter contains the interpretation of all record input and output statements. All special topics of record input and output occur in more than one single statement and therefore they are separated and treated in single chapters (10.4.1 through 10.4.6). The interpretation of the statements (except the unlock statement, which is treated in 10.4.6) are given in 10.4.7. The read statement is subdivided into three syntactically recognizable categories (read(into), read(set), read(ignore)).

Metavariables

t
is-read-st \lor is-write-st \lor
is-rewrite-st \lor is-locate-st \lor
is-delete-st \lor is-unlock-st

opt
is-rec-opt

u
is-n

fu-elem
is-fu-file \lor \}\nids
is-inner-dataset

key
is-char-val-list \lor \}\n
ref-list
is-ref-list

Abbreviations

fu-elem_u = (is-fu-elem_u(FU) \rightarrow u(FU),
T \rightarrow error)

Ref.: is-fu-elem 2-36(113)

fu-csa_u = s-csa(fu-elem_u)

fu-pos_u = s-position(fu-elem_u)

fu-endf_u = s-endfile(fu-elem_u)

fu-bup_u = s-bufferpointer(fu-elem_u)

fu-key_u = s-key(fu-elem_u)
10.4.1 Positioning of record data sets

A data set suited for record input and output either is a list (for a sequential file) or a set of record data (for a direct file). Any access to record data is preceded by positioning. Dependent on the type of access, i.e. with a key in the case of all record files or without key in the case of sequential files only, positioning proceeds differently.

The access without a key causes updating of the current position fu-pos_u in the entry of FU (see new-position-seq, new-position, new-position-delete). Access with a key is either like access without a key (but including a test on the uniqueness of the key), or consists only in a test on the key, or causes raising of the key on-condition (see new-position-direct).

(58) new-position-seq(u) =
BACK e fu-csa_u new-position-delete(u)
T new-position(u)

(59) new-position(u) =
    s-fu:µ(FU;<s-position_u:fu-pos_u + 1>)

(60) new-position-delete(u) =
    s-fu:µ(FU;<s-position_u:fu-pos_u - 1>)

(61) new-position-direct(u,opt,ids) =
    is-key-match(s-key(opt),ids) & SEQ e fu-csa_u s-fu:µ(FU;<s-position_u:(li)(is-key-to-key-match(s-key-elem(i,ids), s-key(opt)))>)
    is-key-match(s-key(opt),ids) null
    T call-io-cond(u,opt,KEY,Ø,807)

Ref.: call-io-cond 10-30 (86)

(62) is-key-to-key-match(key-1,key-2) =

Note: This predicate is implementation-defined; it is satisfied if the two arguments match.
10.4.2 Unbuffered record input and output

The basic transmission from storage consists in the insertion of (keyed or unkeyed) record data in the data set (see `from-direct`, `from-seq`).

Basic transmission into storage tests whether the target storage is part of any storage associated with a task variable which is again associated with an active task, or with an event variable which is again associated with an active input and output event. If the test fails assignment into storage is made without any conversion (see `direct-in`, `seq-in`). Assignment of a key is performed including conversion (see `seq-keyto`).

(64) \[ \text{from-direct}(\text{opt,ids}) = \]
\[
\text{PASS:} \text{ids} \cup \left\{ \mu_{s} (<\text{s-key:} s\text{-key(opt)}, <\text{s-data:} el\text{-rf}(s\text{-pp:s-from(opt)})(\$) >, <\text{s-size:} \text{size(s-pp:s-from(opt)})(\$) >) \right\}
\]
Ref.: el-rf 2-22(68), size 2-21(59)

(65) \[ \text{from-seq}(u,\text{opt,ids}) = \]
\[
\text{PASS:} \mu(\text{ids}; <\text{elem(fu-pos}_{u})\mu_{s} (<\text{s-key:} s\text{-key(opt)}, <\text{s-data:} el\text{-rf}(s\text{-pp:s-from(opt)})(\$) >, <\text{s-size:} \text{size(s-pp:s-from(opt)})(\$) >) >)
\]
Ref.: el-rf 2-22(68), size 2-21(59)
(66) \[ \text{direct-into}(u, \text{opt}, \text{ids}) = \]
\[ \neg \text{is-active}(s-\text{into}(\text{opt}), \text{PA}) \rightarrow \]
\[ s-s: \text{el-ass}(s-p \cdot s-\text{into}(\text{opt}), s-\text{data}((\text{rec})(\text{is-key-to-key-match}
\text{(s-key(rec), s-key(opt)}) \& \text{rec} \in \text{ids})), S) \]
\[ \text{T} \rightarrow \text{error} \]

Ref.: is-active 4-7(11), is-key-to-key-match 10-24(62), el-ass 2-22(70)

(67) \[ \text{sequ-keyto}(u, \text{opt}, \text{ids}) = \]
\[ \neg \text{is-} \Omega \circ \text{s-keyto(opt)} \rightarrow \]
\[ \text{keyto-assign}(u, \text{opt}, \text{s-key\cdot elem}(\text{fu-pos}_u, \text{ids})) \]
\[ \text{T} \rightarrow \text{null} \]

(68) \[ \text{keyto-assign}(u, \text{opt}, \text{key}) = \]
\[ \neg \text{is-} \Omega (\text{key}_1) \& \text{is-correct-key(key, fu-elem}_u, \text{es-titled-dataset}_u) \rightarrow \]
\[ \text{convert-assign}(s-\text{keyto(opt)}, \text{val-op(char-data}_\text{length(key}_1), \text{key}_1) \]
\[ \text{T} \rightarrow \text{call-io-cond}(u, \text{opt}, \text{KEY}, \Omega, 808) \]

where: \text{es-titled-dataset}_u = \text{es-titled-dataset}(s-\text{title}(\text{fu-elem}_u), ES)
\text{key}_1 = \text{adjust-str}(s-\text{keyto(opt)}, \text{key})

Ref.: es-titled-dataset 10-4(7), val-op 8-11(41), char-data 8-13(60), convert-assign 7-9(19), call-io-cond 10-30(86)

Note: The adjustment of the key by adjust-str caters for all cases where convert-assign might call on-conditions.

(69) \[ \text{is-correct-key(key, fu-elem, es-titled-dataset)} = \]

for: \text{is-titled-dataset(es-titled-dataset)}

Note: This predicate is implementation-defined. It is satisfied if the key (key) is correct according to the entry in \text{FU} (fu-elem) and the data set accessed (es-titled-dataset).
(70) \( \text{adjust-str}(g, \text{key}) = \)
\[ \text{is-} \emptyset \text{(key)} \rightarrow \emptyset \]
\[ \text{is-char-pic}\circ s\text{-da}(g) \rightarrow \text{adjust-char-pic}(s\text{-da}(g), \text{key}) \]
\[ s\text{-id}(g) = \text{mk-id(SUBSTR)} \rightarrow \text{adjust-substr}(\text{elem}(2, s\text{-arg-list}(g)),\]
\[ \text{elem}(3, s\text{-arg-list}(g)), \text{key}) \]
\[ T \rightarrow \text{key} \]
for \( \text{is-gen}(g) \lor \text{is-ps-gen}(g) \)
Ref.: \( \text{mk-id } 10-76(214) \)

(71) \( \text{adjust-char-pic}(da, \text{key}) = \)
\[ \text{is-prop-pic}(da) \land \text{is-char-pic-match}(da, \text{key}) \rightarrow \text{key} \]
\[ T \rightarrow \emptyset \]
for \( \text{is-char-pic}(da) \)
Ref.: \( \text{is-prop-pic } 8-47(187), \text{is-char-pic-match } 8-59(224) \)

(72) \( \text{adjust-substr}(i, j, \text{key}) = \)
\[ \text{is-} \emptyset (j) \land \text{lsis-length}(\text{key}) \rightarrow \text{LIST} \text{elem}(n, \text{key}) \]
\[ \neg \text{is-} \emptyset (j) \land \text{lsis-length}(\text{key}) \land i+j-1\text{lsis-length}(\text{key}) \rightarrow \text{LIST} \text{elem}(n, \text{key}) \]
\[ T \rightarrow \emptyset \]
for \( \text{is-intg-val}(i) \land (\text{is-intg-val}(j) \lor \text{is-} \emptyset (j)) \)

(73) \( \text{sequ-into}(u, \text{opt}, \text{ids}) = \)
\[ \neg \text{is-active}(s\text{-into}(\text{opt}), \text{PA}) \rightarrow \]
\[ s\text{-s:el-ass}(s\text{-pp}\text{-s-into}(\text{opt}), s\text{-data}\text{elem}(\text{fu-pos}_u, \text{ids}), \emptyset) \]
\[ T \rightarrow \text{error} \]
Ref.: \( \text{is-active } 4-7(11), \text{ el-ass } 2-22(70) \)
10.4.3 Buffered record input and output

The pointer of the current buffer and the key are part of the entry in FU (see note-buffer, note-key). The buffer itself is part of storage and its allocation is done by giving the record size instead of evaluated data attributes and density (see buffer-allocate). Deletion of the buffer is done by erase-buffer.

Basic transmission into storage is made in two steps, namely by sequ-buffer and buffer-into. The transmission instructions are similar to those of sub-chapter 10.4.2 in the sequential case.

(74) \text{note-buffer}(u,p) = \newline  \text{s-fu:μ}(\text{FU};<\text{s-bufferpointer}·u;p>) \newline \text{for : is-ptr-val}(p)

(75) \text{note-key}(u,\text{key}) = \newline  \text{s-fu:μ}(\text{FU};<\text{s-key}·u;\text{key}>)

(76) \text{buffer-allocate}(\text{size}) = \newline  \text{PASS:buffer-space(size,}$\$) \newline  \text{s-s:el-alloc(buffer-space(size,}$\$),I,}$\$
\text{for : is-size(size)} \newline \text{Ref.: buffer-space 6-13(25), el-alloc 2-23(75)}

(77) \text{erase-buffer}(u) = \newline  \text{¬is-Ω}(\text{fu-bup}_u) \rightarrow \newline  \text{s-fu:δ}(\text{FU};\text{s-bufferpointer}·u,\text{s-key}·u) \newline  \text{s-s:el-free}\{\text{fu-bup}_u,I,}$\$
\text{T} \rightarrow null \newline \text{Ref.: el-free 2-23(76)}

(78) \text{buffer-seq}(u,\text{ids}) = \newline  \text{PASS:μ}(\text{ids};<\text{elem(fu-pos}_u);μ_0(<\text{s-key:fu-key}_u>, \newline  <\text{s-data:el-rf:fu-bup}_u(}$\$
\text{,}<\text{s-size:size}·\text{fu-bup}_u(}$\$
\text{)>)> \newline \text{Ref.: el-rf 2-22(68), size 2-21(59)}
10.4.4 Record input and output on-conditions

There are four types of on-conditions which may occur in record input and output: the key, record, transmit, and endfile on-condition. The conditions which yield a call of the key on-condition are comparatively complicated, and are given in the relevant sub-chapters. The conditions for a call of the remaining on-conditions are treated in the sequel.

Metavariable

\textbf{type} \quad \textit{is-intg-val} \quad \text{a characterization of the type of interrupt}

(82) \textbf{raise-record-cond(u,opt,ids,type)} =

\begin{align*}
\text{size}_r &= \text{size}_s \rightarrow \text{null} \\
T &\rightarrow \text{call-io-cond(u,opt,RECORD},(\text{s-code:rec-code(size}_r,\text{size}_s)\text{), } \\
\quad &\text{where: size}_r = (\text{is- } &\text{s-key(opt)}} \rightarrow \text{s-sizeelem(fu-pos,ids),} \\
&\text{is-key-match(s-key(opt),ids)} \rightarrow \text{s-size}((\text{rec}(\text{is-key-to-key-match(s-key(rec),s-key(opt)}}))))
\end{align*}
\[ \text{size}_s = (\text{is-}(\text{opt}) \rightarrow \text{size}(\text{fu-bup}_u)(S), \]
\[ \quad \text{is-gen} \cdot \text{s-into}(\text{opt}) \rightarrow \text{size}(\text{s-pp} \cdot \text{s-into}(\text{opt}))(S), \]
\[ \quad \text{is-gen} \cdot \text{s-from}(\text{opt}) \rightarrow \text{size}(\text{s-pp} \cdot \text{s-from}(\text{opt}))(S) ) \]

Ref.: \text{call-\text{io-\text{cond} 10-30(86), is-key-match 10-25(63),}}
\[ \text{is-key-to-key-match 10-24(62), size 2-21(59), is-gen 2-27(96)} \]

(83) \text{rec-oncode} (\text{size-1, size-2} ) =

Note: This function is implementation-defined. It yields an integer value
(is-intg-val) as deduced from the size of the record (size-1) and the
size of storage (size-2).

(84) \text{raise-transmit-cond}(u, opt, type) =
\[ \text{is-}(\text{opt}) \rightarrow \text{null} \]
\[ \text{T} \rightarrow \]
\[ \text{call-\text{io-\text{cond} (u, opt, TRANSMIT, Q, type) ;}} \]
\[ \text{upd-fu-elem(u, s-transmit, Q) } \]

where: \text{fu-tmt}_u = \text{s-transmit} (\text{fu-elem}_u)

Ref.: \text{upd-fu-elem 10-6(13), call-\text{io-\text{cond 10-30(86)}}}

(85) \text{raise-endfile-cond}(u, opt, ids, type) =
\[ \text{is-FILEMARK \text{elem}(fu-pos}_u, ids) \rightarrow \text{null} \]
\[ \text{T} \rightarrow \]
\[ \text{call-\text{io-\text{cond} (u, opt, ENDFILE, Q, type) ;}} \]
\[ \text{upd-fu-elem(u, s-endfile, T) } \]

Ref.: \text{upd-fu-elem 10-6(13), call-\text{io-\text{cond 10-30(86)}}}

(86) \text{call-\text{io-\text{cond} (u, opt, io-\text{cond, cbif}, type) =}}
\[ \text{is-}(\text{opt}) \rightarrow \]
\[ \text{s-type:}(\mu(\text{TE} ; <s-\text{cond-list}:s-\text{cond-list}(\text{TE})>)<\mu_0(<s-\text{cond:cond}_1>,<s-\text{cbif:cbif}_1>,<s-type:type>))>)) \]
\[ \text{T} \rightarrow \]
\[ \text{io-\text{cond-return}(io-\text{cond});} \]
\[ \text{call-\text{cond-1}(cond}_1,\emptyset, \text{cbif}_1, \text{type)} \]

cont'd

10.4.4
where: \( \text{cond}_1 = \mu \langle <\text{s-io-cond}, \text{io-cond}>, <\text{s-f-den}, \text{fu-f}_u> \rangle \)

\( \text{cbif}_1 = \mu (\text{cbif}; <\text{s-onkey}, <\text{s-key}(\text{opt}), <\text{s-onfile}, \text{mk-id}, \text{s-at-title}(\text{fu-elem}_u)> >) \)

\( \text{fu-f}_u = \text{s-f}(\text{fu-elem}_u) \)

for: \( \text{is-io-cond}(\text{io-cond}) \) \& \( \neg \text{is-UNDF}(\text{io-cond}) \) \& \( \text{is-cbif}(\text{cbif}) \)

Ref.: \text{call-cond-1} 9-12(19), \text{mk-id} 10-76(214)

Note: For record input and output on-conditions the first alternative is taken if an input and output event has been attached by the statement causing this instruction to be executed. The second alternative applies either for record or stream input and output on-conditions, i.e. also for endpage and name on-condition.

(87) \( \text{io-cond-return}(\text{io-cond}) = \)

\( \text{is-RECORD}(\text{io-cond}) \rightarrow \text{error} \)

\( \text{is-KEY}(\text{io-cond}) \) \& \( \text{is-ENDFILE}(\text{io-cond}) \rightarrow \text{int-next-st} \)

\( \text{T} \rightarrow \text{null} \)

for: \( \text{is-io-cond}(\text{io-cond}) \) \& \( \neg \text{is-UNDF}(\text{io-cond}) \)

Ref.: \text{int-next-st} 5-16(24)

10.4.5 Input and output events

For all record input and output statements and for each specific case where an event option may occur, the operations to be performed as a parallel action are gathered in the single instruction \text{io-transmission}. The distinction between attaching an input and output event and the execution under the control of the current task is made by \text{int-io-event}. The parallel action is established by \text{attach-io-event} in a way which guarantees subsequent notification of all tasks which might wait for the completion of proper transmission.

Metavariable

\text{type} \quad \text{is-WR-DIR} \text{\& is-WR-UNBUF} \text{\& is-RWR-DIR} \text{\& is-RWR-UNBUF} \text{\& is-RD-DIR} \text{\& is-RD-UNBUF-UNKEYLD} \text{\& is-IGN} \text{\& is-DEL-KLYLD} \text{\& is-DEL-UNBUF} \quad \text{a type which characterizes the different kinds of input and output transmission}
(88) \text{int-io-event}(\text{type},u,\text{opt},\text{ref-list}) =

\text{is-EVENT} \circ \text{s-da} \circ \text{s-event}(\text{opt})

\begin{align*}
\text{attach-io-event}(\text{type},u,\text{opt},\text{te},\text{ten}) ; \\
\text{s-ten}(\text{te}) : \text{pass}(\text{ten}(\text{TE})) , \\
\text{s-ev}(\text{te}) : \text{pass}(\text{s-event}(\text{opt})) , \\
\text{ten} : \text{un-name} , \\
\text{s-cond-list}(\text{te}) : \text{pass}(\langle \rangle ) , \\
\text{s-check-list}(\text{te}) : \text{pass}(\langle \langle \text{ref-list} : \text{ref-list} \rangle , \langle \langle \text{e} \rangle \rangle \rangle )
\end{align*}

\text{is-Q} \circ \text{s-event}(\text{opt})

\begin{align*}
\text{call-check-cond}(\text{permute(\text{ref-list})},E,809) ; \\
\text{io-transmission}(\text{type},u,\text{opt})
\end{align*}

\text{error}

Ref.: \text{un-name} 2-17(36), \text{call-check-cond} 9-15(27), \text{permute} 4-16(38)

Note: Any references for which the check on-condition is to be raised are known when the input and output event is attached. As a consequence the component s-check-list(\text{te}) remains unchanged throughout the input and output transmission whereas the component s-cond-list(\text{te}) will be updated by call-io-cond.

(89) \text{attach-io-event}(\text{type},u,\text{opt},\text{te},\text{ten}) =

\begin{align*}
\neg \text{is-active}(\text{s-ev}(\text{te}),\text{PA}) & \rightarrow \\
\text{s-s} : \text{el-ass}(\text{pp} \circ \text{s-ev}(\text{te}),\text{ev-vr}(\text{O-BIT},\text{O}),\text{e}) \\
\text{s-te} : \mu(\text{TE};\text{s-io-ev}(\text{s-io-ev}(\text{TE}) \cup \{\text{ten}\})) \\
\text{s-pa} : \mu(\text{PA};\text{ten} ; \mu(<\text{c} \circ \text{event-transmission}(\text{type},u,\text{opt})>) ) \\
\text{s-fu} : \mu(\text{FU};\text{s-io-ev}(\text{s-io-ev} \cup \text{FU}) \cup \{\text{ten}\})
\end{align*}

\text{error}

for: \text{is-te(\text{te})} \ & \text{is-n(\text{ten})}

Ref.: \text{is-active} 4-7(11), \text{el-ass} 2-22(70), \text{ev-vr} 4-7(12)

Note: The entry in s-io-ev(\text{TE}) enables deletion of the event at task termination (see \text{term-events}). The entry in s-io-ev \cup \text{u(FU)} serves for the same purpose but during closing of the file (see \text{delete-buffer-event}).
10.4.6 Exclusive files and the unlock statement

The entry in FU associated with an exclusive file has components selected by unique task names which are sets of keys. Any key which is a member of such a set is said to be locked by that task, the unique name of which selects the set.

These keys are entered only by the instruction wait-for-unlock which is executed before any input and output transmission over an exclusive file might occur. The deletion of a locked key occurs indirectly by execution of the instruction unlock. This instruction normally ends transmission over an exclusive file. A locked key may also be deleted directly by the instruction int-unlock-1.
Since it provides a notational convenience for sub-chapter 10.4.7, all instructions defined in the sequel except `int-unlock-st` are also defined for non-exclusive files (in this case they result in `null` instructions).

**Metavariable**

\[ \text{ten} = \text{is-s-main} \lor \text{is-n} \]

the unique name of a task, i.e. the name of the current task or the name of the task which attached the current input and output event or the name of any task

\[
(93) \quad \text{int-unlock-st}(t) = \]
\[
\begin{align*}
\text{int-unlock-1}(u, \text{opt}) ; \\
\text{u: int-impl-open}(t) ; \\
\text{s-key(opt):conv-char-string}(s\text{-key}(t))
\end{align*}
\]

Ref.: `conv-char-string 10-12(30), int-impl-open 10-13(32)`

\[
(94) \quad \text{int-unlock-1}(u, \text{opt}) = \]
\[
\begin{align*}
\text{is-locked-down}(\text{fu-elem}, s\text{-ten(TE)}, s\text{-key(opt)}) \rightarrow \\
\text{s-fu:u(FU);\langle s\text{-ten(TE)}\rangle u: (s\text{-ten(TE)}u(FU) - \{s\text{-key(opt)}\} \rangle) \\
\text{s-pa:activate-tasks(PA)} \\
T \rightarrow \text{null}
\end{align*}
\]

Ref.: `activate-tasks 4-18(44)`

**Note:** Regardless of what alternative is taken, the key (key) is considered unlocked for the scope of the file after execution of this instruction. Since an instruction wait-for-unlock might wait in any other task for this unlocking, provision is made by `activate-tasks(PA)` to reactivate any waiting task.

\[
(95) \quad \text{is-locked-down}(\text{fu-elem}, \text{ten}, \text{key}) = \]
\[
\text{EXCL} \in \text{s-csa}(\text{fu-elem}) \land \text{\neg is-Q}\text{ten(fu-elem)} \land \text{key \epsilon ten(fu-elem)}
\]

10.4.6
(96) \text{wait-for-unlock}(u,\text{opt}) =
\text{is-locked-foreign}(\text{fu-elem}_u, s\text{-ten}(\text{TE}), s\text{-key}(\text{opt})) \rightarrow
\text{wait-for-unlock}(u,\text{opt});
\text{set-wait-state-3}(u,\text{opt})
\text{is-\neg} \circ s\text{-nolock}(\text{opt}) \land \text{EXCL} \in \text{fu-csa}_u \rightarrow
s\text{-fu}:\mu(\text{FU}; (s\text{-ten}(\text{TE})) \cup u :\text{key-set}_{\text{ten}} \cup \{s\text{-key}(\text{opt})\})
T \rightarrow \text{null}
where: \text{key-set}_{\text{ten}} = (\text{is-\neg} \circ (s\text{-ten}(\text{TE})) \circ u(\text{FU}) \rightarrow \{\}, T \rightarrow (s\text{-ten}(\text{TE})) \circ u(\text{FU})

(97) \text{is-locked-foreign}(\text{fu-elem}, \text{ten}, \text{key}) =
\text{EXCL} \in s\text{-csa}(\text{fu-elem}) \land (\exists \text{ten-1})(\text{ten-1} \neq \text{ten} \land \text{key} \in \text{ten-1}(\text{fu-elem}))
Note: The predicates \text{is-locked-own} and \text{is-locked-foreign} depend on an entry of \text{FU} but not on the whole \text{FU}.

(98) \text{set-wait-state-3}(u,\text{opt}) =
\text{is-locked-foreign}(\text{fu-elem}_u, s\text{-ten}(\text{TE}), s\text{-key}(\text{opt})) \rightarrow
\text{null}
T \rightarrow
s\text{-te}:\mu(\text{TE}; <s\text{-wait}:T>)
Note: If the condition for which \text{set-wait-state-3} has been put into the control is no more satisfied when the instruction is selected for execution, the task is not set into the wait state. See also the note with \text{set-wait-state}.

(99) \text{unlock}(u,\text{opt}) =
\text{EXCL} \notin \text{fu-csa}_u \rightarrow \text{null}
\text{is-\neg} \circ s\text{-event}(\text{opt}) \rightarrow \text{int-unlock-1}(u,\text{opt})
T \rightarrow
s\text{-te}:\mu(\text{TE}; <s\text{-unlock}:\mu_0(<s\text{-u}:u>, <s\text{-rec-opt}:\text{opt}>)>)
Note: The last two alternatives make a similar distinction between the event and non-event case as in \text{call-io-cond}.
10.4.7 Record input and output statements

In this chapter all record input and output statements except the unlock statement (see 10.4.6) are described. The read statement is divided into three syntactically separable operations: read into, read set, read ignore, corresponding to the mutually exclusive options.

For all record input and output statements the sequence of operations is essentially the same and is described in the following.

1) Definition of the range of the statement, i.e. there is a check for admissible arguments in the options.

2) Evaluation of the options.

3) Implicit opening of the file and a test for compatibility of file and statement.

4) For sequential files and with the read statement only, the endfile condition can occur immediately, if this status has been reached already by a previous statement.

5A) For statements with the event option the event is attached, i.e. the parallel action is established.

5B) (Mutually exclusive with 5A)
   For buffered files, and if a buffer has been allocated previously, the contents is transmitted to the data set including a key in the case of keyed buffered files. (For details of the transmission see 7.)

6A) For exclusive files the test for locked or unlocked is made and locking is performed if necessary.

6B) (Mutually exclusive with 6A)
   For buffered files the old buffer is omitted, a new one is allocated and the pointer assigned.

7) Elementary transmission

7.1) Since for all data transmissions the data set is needed, the first instruction executed is tk-dataset(u,key). The second argument is a key for keyed files, otherwise it is the null object.
7.2) Test for multiple occurrences of keys (at writing) or for non-existent keys (at reading).

7.3) For sequential files the position is updated. It may be altered by +1 (or -1 for files with the backwards attribute) or by any other positive integer (read ignore) or by an access with a key.

7.4A) For input the record and transmit conditions may occur.

7.4B) (Mutually exclusive with 7.4A)
For output the new data set is formed and also the record and transmit conditions may occur.

7.5) The actual data transmission is performed, i.e. on input the record data is assigned to storage or buffer, on output the whole dataset is handled by `upd-dataset(u,key,ids)`.

7.6) For buffered input the second step, from buffer to storage is performed.

7.7) The check on-condition may be raised.

8) For exclusive files unlocking may be performed.

(100) \(\text{int-read-st}(t)\) =
\[\begin{align*}
\text{is-into-read}(t) & \rightarrow \text{int-into-read}(t) \\
\text{is-set-read}(t) & \rightarrow \text{int-set-read}(t) \\
\text{is-ignore}(t) & \rightarrow \text{int-ignore}(t)
\end{align*}\]

Ref.: `int-into-read 10-43(114), int-set-read 10-46(119), int-ignore 10-47(122)`

10.4.7.1 The write statement

The write statement adds to a data set new record data. If for a buffered file a buffer has been already allocated, the contents of the buffer will be written out first.
(101) int-write-st(t) =
    is-var->s-attr(s-id-list->s-from(t)(E))(AT) →
    int-write-1(u,opt);
    u:int-impl-open(t);
    s-from(opt):eval-ref-gen(s-from(t),E),
    s-key(opt):conv-char-string(s-keyfrom(t)),
    s-event(opt):eval-opt-gen(s-event(t))
    T → error
    Ref.: eval-opt-gen 10-12(29), conv-char-string 10-12(30),
          eval-ref-gen 7-25(59), int-impl-open 10-13(32)

(102) int-write-1(u,opt) =
    ~is-connected->s-from(opt) → error
    DIR ∈ fu-csa u →
    int-io-event(WR-DIR,u,opt,<>)
    UNBUF ∈ fu-csa u →
    int-io-event(WR-UNBUF,u,opt,<>)
    BUF ∈ fu-csa u →
    write-sequ(u,opt,ids);
    ids:tk-dataset(u,Ω);
    execute-locate(u)
    Ref.: is-connected 7-33(87), int-io-event 10-32(88),
          execute-locate 10-42(112), tk-dataset 10-7(16)

(103) write-direct(u,opt) =
    unlock(u,opt);
    write-direct-1(u,opt,ids);
    ids:tk-dataset(u,s-key(opt));
    wait-for-unlock(u,opt)
    Ref.: wait-for-unlock 10-35(96), tk-dataset 10-7(16),
          unlock 10-35(99)
(104) \( \text{write-direct-1}(u, opt, ids) = \)

\[ \neg \text{is-key-match}(s-key(opt), ids) \rightarrow \]

\( \text{raise-transmit-cond}(u, opt, 810); \)

\( \text{upd-dataset}(u, s-key(opt), idsnew); \)

\( \text{ids-new: from-direct}(opt, ids) \)

\[ \text{T} \rightarrow \]

\( \text{call-io-cond}(u, opt, KEY, \emptyset, 811) \)

Ref.: \text{from-direct} 10-25(64), \text{upd-dataset} 10-6(11), \text{raise-transmit-cond} 10-30(84), \text{call-io-cond} 10-30(86), \text{is-key-match} 10-25(63)

(105) \( \text{write-unbuf}(u, opt) = \)

\( \text{write-seq}(u, opt, ids); \)

\( \text{ids: tk-dataset}(u, \emptyset) \)

Ref.: \text{tk-dataset} 10-7(16)

(106) \( \text{write-seq}(u, opt, ids) = \)

\[ \text{is-} \emptyset \circ s-key(opt) \lor \neg \text{is-key-match}(s-key(opt), ids) \rightarrow \]

\( \text{raise-transmit-cond}(u, opt, 812); \)

\( \text{upd-dataset}(u, \emptyset, idsnew); \)

\( \text{ids-new: from-seq}(u, opt, ids); \)

\( \text{new-position}(u) \)

\[ \text{T} \rightarrow \]

\( \text{call-io-cond}(u, opt, KEY, \emptyset, 813) \)

Ref.: \text{new-position} 10-24(59), \text{from-seq} 10-25(65), \text{and see} 10-39(104)

10.4.7.2 The rewrite statement

The rewrite statement replaces record data in the data set by data defined by the from option or, if for a buffered file no from option exists, by the contents of the buffer.
(107) int-rewrite-st(t) =
    is-∅s-from(t) ∨ is-var-s-attr(sel-s-id-lists-from(t)(E))(AT) →
    int-rewrite-1(u,opt);
    u: int-impl-open(t);
    s-from(opt): eval-opt-gen(s-from(t),E),
    s-key(opt): conv-char-string(s-key(t)),
    s-event(opt): eval-opt-gen(s-event(t))
    T → error

Ref.: see 10-38(101)

(108) int-rewrite-1(u,opt) =
    ¬is-∅s-from(opt) & ¬is-connected-s-from(opt) →
    error
    DIR ∈ fu-csa_u →
    int-io-event(RWR-DIR,u,opt,<>)
    UNBUF ∈ fu-csa_u →
    int-io-event(RWR-UNBUF,u,opt,<>)
    BUF ∈ fu-csa_u & ¬is-∅s-from(opt) →
    raise-record-cond(u,opt,ids,814);
    write-seq(u,opt,ids);
    new-position-delete(u);
    ids: tk-dataset(u,∅)
    BUF ∈ fu-csa_u & is-∅s-from(opt) →
    raise-record-cond(u,opt,ids,815);
    upd-dataset(u,∅,idsnew);
    idsnew: buffer-seq(u,ids);
    ids: tk-dataset(u,∅)

Ref.: is-connected 7-33(87), int-io-event 10-32(88),
      tk-dataset 10-7(16), new-position-delete 10-24(60),
      write-seq 10-39(106), raise-record-cond 10-29(82),
      buffer-seq 10-28(78), upd-dataset 10-6(11)
(109) rewrite-direct(u,opt) =
    unlock(u,opt);
    raise-record-cond(u,opt,ids,816);
    write-direct-1(u,opt,ids);
    delete-keyed-1(u,opt,ids);
    ids:tk-dataset(u,s-key(opt));
    wait-for-unlock(u,opt)

Ref.: wait-for-unlock 10-35(96), tk-dataset 10-7(16),
      delete-keyed-1 10-49(129), raise-record-cond 10-29(82),
      unlock 10-35(99), write-direct-1 10-39(104)

(110) rewrite-unbuf(u,opt) =
    raise-record-cond(u,opt,ids,817);
    write-unbuf(u,opt);
    new-position-delete(u)

Ref.: write-unbuf 10-39(105), and see 10-40(108)

10.4.7.3 The locate statement

The locate statement operates on buffered output files only. If there
exists already a buffer, at first the contents will be transmitted to the
data set. Then the old buffer is omitted, i.e. the buffer is freed and its
pointer and the key (if any) are deleted from the entry in FU. A new buffer
is allocated which includes the assignment to the explicit or implicit
pointer variable. Initialization of the new buffer is followed by making
new entries for the new buffer pointer and new key in FU. The check on-
condition is called for the explicit or implicit pointer variable.
(111) \texttt{int-locate-st}(t) =
\begin{align*}
is\text{-based}(\textit{attr}_1) & \quad \rightarrow \\
call\text{-set\text{-}check}(t); & \\
note\text{-key}(u, \textit{key}); & \nonumber \\
note\text{-buffer}(u, \textit{p}); & \\
p: \text{pass\text{-}s-pp}(\text{gen\text{-}1}); & \\
\text{initialize}(s\text{-id}(t), s\text{-init\text{-}set}(\textit{attr}_1), s\text{-da}(\textit{attr}_1), \text{gen\text{-}1}, \textit{env}_1); & \\
\text{gen\text{-}1}: \text{based\text{-}allocate}(n_1(\text{DN}), s\text{-dens}(\textit{attr}_1), \text{gen\text{-}2}); & \\
\text{execute\text{-}locate}(u); & \\
u: \text{int\text{-}impl\text{-}open}(t); & \\
\text{gen\text{-}2}: \text{eval\text{-}set\text{-}gen}(t), & \\
\text{key} : \texttt{conv\text{-}char\text{-}string}(s\text{-keyfrom}(t)) & \\
\end{align*}

\text{T} \rightarrow \text{error}

where:
\begin{align*}
n_1 & = \text{sel}(s\text{-id}(t))(E) \\
\textit{attr}_1 & = s\text{-attr}\cdot n_1(\text{AT}) \\
\textit{env}_1 & = s\text{-e}\cdot n_1(\text{AT})
\end{align*}

Ref. : \texttt{conv\text{-}char\text{-}string} 10-12(30), \texttt{eval\text{-}set\text{-}gen} 6-8(14), \texttt{int\text{-}impl\text{-}open} 10-13(32), \texttt{based\text{-}allocate} 6-8(13), \texttt{initialize} 6-16(34), \texttt{note\text{-}buffer} 10-28(74), \texttt{note\text{-}key} 10-28(75), \texttt{call\text{-}set\text{-}check} 6-4(3)

(112) \texttt{execute\text{-}locate}(u) =
\begin{align*}
is\text{-Q}(fu\text{-bup}_u) & \rightarrow \text{null} \\
\text{T} & \rightarrow \\
\text{erase\text{-}buffer}(u); & \\
\text{buffer\text{-}output}(u, \textit{ids}); & \\
is\text{ids}: \texttt{tk\text{-}dataset}(u, \varnothing); & \\
\text{new\text{-}position}(u)
\end{align*}

Ref.: \texttt{new\text{-}position} 10-24(59), \texttt{tk\text{-}dataset} 10-7(16), \texttt{erase\text{-}buffer} 10-28(77)
buffer-output(u, ids) =

~is-Q(fu-key) & is-key-match(fu-key, ids) →

call-io-cond(u, opt, KEY, O, 818)

T →

raise-transmit-cond(u, O, 819);
upd-dataset(u, O, idsnew);
idsnew: buffer-seq(u, ids)

where: opt = μ_o(<s-key: fu-key>)

Ref.: is-key-match 10-25(63), call-io-cond 10-30(86), buffer-seq 10-28(78),
upd-dataset 10-6(11), raise-transmit-cond 10-30(84)

10.4.7.4 Read (into) statement

The read statement with into option assigns record data to the referenced variable, and if the keyto option is specified also the key is transmitted. For buffered files the old buffer is omitted, a new buffer is allocated, and the transmission is done in two steps.

int-into-read(t) =

is-vars-attr(sel·s-id-list·s-into(t)(E))(AT) →

int-into-1(u, opt, <s-into(t) opt-list·s-keyto(t)));

T → error

Ref.: eval-opt-gen 10-12(29), conv-char-string 10-12, (30),
eval-ref-gen 7-25(59), int-impl-open 10-13(32),
opt-list 10-108(295)
(115) \text{int-into-1}(u, \text{opt}, \text{ref-list}) =
\begin{align*}
\text{is-T}(\text{fu-endf}_u) & \rightarrow \\
\text{call-ic-ond}(u, \varnothing, \text{ENDFILE}, \varnothing, 800) & \rightarrow \\
\neg \text{is-CHAR-gen-base-s-keyto(opt)} \lor \\
\neg \text{is-\varnothing-s-into(opt)} \land \\
\neg \text{is-connected-s-into(opt)} & \rightarrow \\
\text{error} & \\
\text{DIR} \in \text{fu-csa}_u \land \text{UNBUF} \in \text{fu-csa}_u & \land \\
\neg \text{is-\varnothing-s-key(opt)} & \rightarrow \\
\text{int-io-event}(\text{RD-DIR}, u, \text{opt}, \text{ref-list}) & \\
\text{UNBUF} \in \text{fu-csa}_u & \land \\
\neg \text{is-\varnothing-s-key(opt)} & \rightarrow \\
\text{int-io-event}(\text{RD-UNBUF-UNKEYED}, u, \text{opt}, \text{ref-list}) & \\
\text{BUF} \in \text{fu-csa}_u & \land \\
\neg \text{is-\varnothing-s-key(opt)} & \rightarrow \\
\text{call-check-cond}(\text{permute}(\text{ref-list}), E, 820) & ; \\
\text{read-buffered-into}(u, \text{opt}, \text{ids}) & ; \\
\text{upd-fu-elem}(u, s, \text{read}, T) & ; \\
\text{new-position-direct}(u, \text{opt}, \text{ids}) & ; \\
\text{ids:tk-dataset}(u, s, \text{key(opt)}) & \\
\text{BUF} \in \text{fu-csa}_u & \land \\
\neg \text{is-\varnothing-s-key(opt)} & \rightarrow \\
\text{call-check-cond}(\text{permute}(\text{ref-list}), E, 822) & ; \\
\text{read-buffered-into}(u, \text{opt}, \text{ids}) & ; \\
\text{raise-endfile-cond}(u, \varnothing, \text{ids}, 821) & ; \\
\text{upd-fu-elem}(u, s, \text{read}, T) & ; \\
\text{new-position-seq}(u) & ; \\
\text{ids:tk-dataset}(u, \varnothing) & \\
\end{align*}

Ref.: \text{call-ic-ond} 10-30(86), \text{gen-base} 10-21(51), \text{is-connected} 7-33(87), 
\text{int-io-event} 10-32(88), \text{tk-dataset} 10-7(16), 
\text{new-position-direct} 10-24(61), \text{upd-fu-elem} 10-6(13), 
\text{call-check-cond} 9-15(27), \text{permute} 4-16(38), 
\text{new-position-seq} 10-24(58), \text{raise-endfile-cond} 10-30(85)

(116) \text{into-direct}(u, \text{opt}) =
\begin{align*}
\text{direct-into}(u, \text{opt}, \text{ids}) & ; \\
\text{raise-record-cond}(u, \text{opt}, \text{ids}, 824) & ; \\
\text{raise-transmit-cond}(u, \text{opt}, 823) & ; \\
\text{upd-fu-elem}(u, s, \text{read}, \text{read}_1) & ; \\
\text{new-position-direct}(u, \text{opt}, \text{ids}) & ; \\
\text{ids:tk-dataset}(u, s, \text{key(opt)}) & ; \\
\text{wait-for-unlock}(u, \text{opt}) & ; \\
\end{align*}

cont'd
10.4.7.5 Read (set) statement

The read statement with set option assigns record data to a buffer and sets a pointer to the buffer. If the keyto option is specified also the key is transmitted.
(119) \( \text{int-set-read}(t) = \)
\[
\begin{align*}
&\text{int-set-1}(u, \text{opt}, <s-set-ref>(t) > \text{opt-list}\text{-}\text{s-keyto}(t)) ; \\
&u: \text{int-impl-open}(t) ; \\
&s\text{-set}(\text{opt}): \text{eval-set-gen}(t) , \\
&s\text{-key}(\text{opt}): \text{conv-char-string}(s\text{-key}(t)) , \\
&s\text{-keyto}(\text{opt}): \text{eval-opt-gen}(s\text{-keyto}(t)) \\
\end{align*}
\]
Ref.: \text{eval-set-gen} 6-8(14), and see 10-43(114)

(120) \( \text{int-set-1}(u, \text{opt}, \text{ref-list}) = \)
\[
\begin{align*}
&\text{is-T}(\text{fu-endf}_u) \\
&\text{call-io-cond}(u, \mathcal{Q}, \text{ENDFILE}, \mathcal{Q}, 630) \\
&\text{is-CHAR-gen-base}\text{-}\text{s-keyto}(\text{opt}) \text{ v } \text{is-PTR}\text{-}\text{s-set}(\text{opt}) \text{ e r r o r } \\
&\text{is-\mathcal{Q}\text{-}\text{s-key}(\text{opt})} \\
&\text{call-check-cond}(\text{permute}(\text{ref-list}), \mathcal{E}, 831) \\
&\text{buffered-set}(u, \text{opt}, \text{ids}) ; \\
&\text{upd-fu-elm}(u, s\text{-read}, T) ; \\
&\text{new-position-direct}(u, \text{opt}, \text{ids}) ; \\
&\text{ids}: \text{tk-dataset}(u, s\text{-key}(\text{opt})) \\
&\text{is-\mathcal{Q}\text{-}\text{s-key}(\text{opt})} \\
&\text{call-check-cond}(\text{permute}(\text{ref-list}), \mathcal{E}, 833) ; \\
&\text{sequ-keyto}(u, \text{opt}, \text{ids}) ; \\
&\text{buffered-set}(u, \text{opt}, \text{ids}) ; \\
&\text{raise-endfile-cond}(u, \mathcal{Q}, \text{ids}, 832) ; \\
&\text{upd-fu-elm}(u, s\text{-read}, T) ; \\
&\text{new-position-sequ}(u) ; \\
&\text{ids}: \text{tk-dataset}(u, \mathcal{Q}) \\
\end{align*}
\]
Ref.: \text{sequ-keyto} 10-26(67), and see 10-44(115)
(121) \texttt{buffered-set}(u, opt, ids) =
\begin{align*}
\texttt{note-key}(u, s\text{-}key\cdot elem(fu-pos_u, ids)); \\
\texttt{sequ-buffer}(u, ids); \\
\texttt{raise-transmit-cond}(u, opt, 834); \\
\texttt{note-buffer}(u, p); \\
\texttt{assign}(s\text{-}set(opt), op); \\
\texttt{op}:\texttt{pass-val-op}(\texttt{PTR}, p); \\
\texttt{p}:\texttt{buffer-allocate}(s\text{-}size \cdot elem(fu-pos_u, ids)); \\
\texttt{erase-buffer}(u)
\end{align*}

Ref.: val-op 8-11(41), \texttt{assign} 7-9(20), and see 10-45(118)

10.4.7.6 Read (ignore) statement

The read statement with ignore option readjusts the position of a sequential file.

(122) \texttt{int-ignore}(t) =
\begin{align*}
\texttt{int-ignore-1}(u, opt); \\
\texttt{u}:\texttt{int-impl-open}(t); \\
\texttt{s\text{-}ignore}(opt):\texttt{eval-intg-expr}(s\text{-}ignore(t), E), \\
\texttt{s\text{-}event}(opt):\texttt{eval-opt-gen}(s\text{-}event(t))
\end{align*}

Ref.: \texttt{eval-opt-gen} 10-12(29), \texttt{eval-intg-expr} 7-21(52), \\
\texttt{int-impl-open} 10-13(32)

(123) \texttt{int-ignore-1}(u, opt) =
\begin{align*}
\texttt{s\text{-}ignore}(opt) \leq 0 \rightarrow \texttt{null} \\
\texttt{is-T}(fu-endf_u) \rightarrow \texttt{call-io-cond}(u, \varnothing, \texttt{ENDFILE}, \varnothing, 835) \\
T \rightarrow \\
\texttt{int-io-event}(\texttt{IGN}, u, opt, <>)
\end{align*}

Ref.: \texttt{call-io-cond} 10-30(86), \texttt{int-io-event} 10-32(88)
(124) \textbf{ignore}(u,\texttt{opt}) =
\begin{align*}
\text{s-ignore}(\texttt{opt}) & \neq 1 \rightarrow
\text{ignore}(u,\texttt{opt})_1;
\text{raise-transmit-cond}(u,\texttt{opt},837);
\text{raise-endfile-cond}(u,\texttt{opt},\texttt{ids},836);
\text{upd-fu-elem}(u,\texttt{s-read},T);
\text{new-position-segu}(u);
\text{ids} : \text{tk-dataset}(u,\varnothing)
\end{align*}
\begin{align*}
T & \rightarrow \text{null}
\end{align*}

where: \texttt{opt}_1 = \mu(\texttt{opt}; <\text{s-ignore} : \text{s-ignore}(\texttt{opt}) \rightarrow 1>)

Ref.: \text{tk-dataset} 10-7(16), \text{new-position-segu} 10-24(58),
\text{upd-fu-elem} 10-6(13), \text{raise-endfile-cond} 10-30(85),
\text{raise-transmit-cond} 10-30(84)

\textbf{10.4.7.7 The delete statement}

The delete statement deletes from a data set accessed by a direct update file the record data designated by the key option, or in the case of a sequential update file, the last record data read. In the latter case the record data is only deleted if a read statement has been executed previously.

(125) \textbf{int-delete-st}(t) =
\begin{align*}
\text{int-delete-1}(u,\texttt{opt});
\text{u} : \text{int-impl-open}(t);
\text{s-key}(\texttt{opt}) : \text{conv-char-string}(\texttt{s-key}(t)),
\text{s-event}(\texttt{opt}) : \text{eval-opt-gen}(\texttt{s-event}(t))
\end{align*}

Ref.: \text{eval-opt-gen} 10-12(29), \text{conv-char-string} 10-12(30),
\text{int-impl-open} 10-13(32)

(126) \textbf{int-delete-1}(u,\texttt{opt}) =
\begin{align*}
\text{DIR } \epsilon \text{ fu-csa}_u \rightarrow
\text{int-io-event}(\text{DEL-KEYED},u,\texttt{opt},<>)
\end{align*}
\begin{align*}
\text{is-\varnothing}(\text{fu-read}_u) & \rightarrow \text{null}
\text{UNBUF } \epsilon \text{ fu-csa}_u \rightarrow
\text{int-io-event}(\text{DEL-UNBUF},u,\texttt{opt},<>)
\end{align*}
\begin{align*}
\text{cont'd}
\end{align*}

\textbf{10.4.7.7 cont'd...}
BUF ∈ fu-csa_u

\[
erase-buffer(u); \\
upd-dataset(u, Q, idsnew);
\]
\[
upd-fu-elem(u, s-read, Q);
\]
\[
idsnew: pass-delete(ids, fu-pos_u);
\]
\[
ids: tk-dataset(u, Q)
\]

where: fu-read_u = s-read(fu-elem_u)

Ref. : int-io-event 10-32(88), tk-dataset 10-7(16),
\[
upd-fu-elem 10-6(13), \quad \text{upd-dataset} 10-6(11),
\]
\[
\text{erase-buffer} 10-28(77),
\]

127) delete(L,i) =

\[
is-list(L) \rightarrow \text{LIST} \text{elem}(k,L) \leftarrow \text{LIST} \text{elem}(k,L)
\]

for : is-list(L)

Note: This function deletes the i-th element from the list L.

(128) delete-keyed(u,opt) =

\[
\text{unlock}(u,\text{opt});
\]
\[
delete-keyed-l(u,\text{opt},\text{ids});
\]
\[
\text{ids: tk-dataset}(u, s\text{-key}(\text{opt}));
\]
\[
\text{wait-for-unlock}(u,\text{opt})
\]

Ref.: wait-for-unlock 10-35(96), tk-dataset 10-7(16), unlock 10-35(99)

129) delete-keyed-l(u,\text{opt},\text{ids}) =

\[
is-key-match(s\text{-key}(\text{opt}),\text{ids}) \rightarrow
\]
\[
\text{upd-dataset}(u, Q, \text{idsnew});
\]
\[
\text{idsnew: pass}(\text{ids} - (\text{lrec})(\text{is-key-to-key-match}(s\text{-key}(\text{rec}), s\text{-key}(\text{opt}))))
\]
\[
T \rightarrow
\]
\[
\text{call-\text{io}-\text{cond}}(u,\text{opt},\text{KEY}, Q, 838)
\]

Ref.: is-key-match 10-25(63), is-key-to-key-match 10-24(62),
\[
\text{upd-dataset} 10-6(11), \text{call-\text{io}-\text{cond}} 10-30(86)
(130) \texttt{delete-unbuf(u)} =
\begin{verbatim}
  \texttt{upd-dataset(u,\emptyset,idsnew);}
  \texttt{upd-fu-elem(u,s-read,\emptyset);}  
  \texttt{new-position-delete(u);}           
  \texttt{idsnew:pass-delete(ids,fu-pos_{u});}
  \texttt{ids:tk-dataset(u,\emptyset)}
\end{verbatim}

Ref.: \texttt{new-position-delete 10-24(60), and see 10-48(126)}
10.5 Stream Transmission

Data transmission by the get and put statement is defined in this section. As opposed to record transmission, the statements either name a file or a string which supply or receive converted data.

The first two subsections contain the actions common for stream transmission. Section 10.5.1 defines the expansion of data specifications, and section 10.5.2 defines the abstract syntax of data fields and functions and predicates dealing with data fields. Since conversion from character string to arithmetic data type is similar with data field handling, the definition of the instruction char-num-conv has been included as section 10.5.2.4.

The following two subsections have a similar arrangement and define the get and put statement, respectively. The definition of the control oriented part of the statement which contains initialization and the interpretation of the statement options is followed by the data field oriented part. This part in turn contains the definition of the elementary transmission as far as it is independent from the modes of transmission. The following subsections define list-, data- and edit-directed transmission.

10.5.1 Data specifications

Data specifications may consist of data lists and format lists. Iterated data lists contain items which specify iteration rather like do groups, format lists may specify iteration or substitution by means of repetition factors or remote formats.

The expansion of iterated data and format lists is defined in this section.

10.5.1.1 Iterated data lists

Iterated data lists which satisfy the predicate is-contr-item are handled by the instruction int-contr-item. This instruction is executed whenever an iterated data list is processed by a get or put statement; the instruction int-item-list links control back into the get or put statement which contained the iterated data list.
Metavariables

- **cit** is-contr-item an iterated data list
- **itl** is-item-list a list of items of a data list
- **io-link** is-io-link a link used for interpretation of iterated data lists
- **st** is-GET v is-PUT a statement type
- **u** is-n a file union name
- **tr** is-tr an object which characterizes the mode of transmission

(131) **int-contr-item**(st,u,tr,cit)

\[
\text{ref: int-contr-do 5-23(36)}
\]

(132) **int-item-list**(itl,io-link) =

\[
\text{ref: get-list-edit 10-67(189), put 1o-91(253)}
\]

Note: The parameter io-link actually will be replaced by the second argument of the instruction int-contr-do.

(133) **is-io-link** = (<s-st:is-GET v is-PUT>, <s-u:is-n>, <s-tr:is-tr>)

(134) **is-tr** = is-data-tr v is-list-tr v is-edit-tr

(135) **is-data-tr** = (<s-type:is->,<s-copy:is-opt>)

(136) **is-list-tr** = (<s-type:is-,*>,<s-copy:is-opt>)
(137) \textbf{is-edit-tr} = (\langle \texttt{s-type:is-format-list}, \texttt{s-fo-i:is-intg-val}, \texttt{s-copy:is-opt} \rangle)

\textbf{Note}: The format list is the (unevaluated) format list as taken from the data specification.

\textbf{10.5.1.2 Expansion of format lists}

The instruction \texttt{eval-format} yields the next evaluated format from a format list. This format is either supplied by the format list \texttt{spec-fol} from the data specification or by the format list entry in the element of \texttt{FU} selected by \texttt{u} and \texttt{fo-i} or, in the case of a remote format, is the denotation of a format label.

\textbf{Metavariables}

- \texttt{fo-val}: is-format-den
- \texttt{fo-op}: is-op
- \texttt{fol}, \texttt{spec-fol}: is-format-list
- \texttt{fo-i}: is-intg-val
- \texttt{i}: is-intg-val
- \texttt{u}: is-n
- \texttt{efo}

\textbf{10.5.1.2 cont'd}

(138) \texttt{eval-format} (\texttt{u,spec-fol,fo-i}) =

\begin{align*}
\texttt{is-\llcorner (fu-fol_u) \lor is-\llcorner (fu-fol_u) \longrightarrow}
\texttt{eval-format} \ (\texttt{u,spec-fol,fo-i});
\texttt{upd-fu-elem} \ (\texttt{u,fo-sele_u,spec-fol})
\texttt{is-format-iter} \ (\texttt{hf_u})
\texttt{eval-format} \ (\texttt{u,spec-fol,fo-i});
\texttt{upd-fu-elem} \ (\texttt{u,fo-sele_u,fol});
\texttt{fol:pass-rep-conc} \ (i, is-format-list(hf_u), t_{f_u});
\texttt{i:eval-intg-exp} \ (s-rep-factor(hf_u), E)
\end{align*}

cont'd
is-remote-format(hf_u) →
  eval-format(u,spec-fol,fo-i);
  upd-fu-elem(u,fo-sel_u,fol-1);
  fol-1: pass-rep-conc(1,fol-2,tf_u);
  fol-2: verify-remote-format(fo-val);
  fo-val: pass-op-val(fo-op);
  fo-op: eval-ref(s-ref(hf_u),E)

is-data-format(hf_u) ∨ is-control-format(hf_u) →
  pass(efo);
  upd-fu-elem(u,fo-sel_u,tf_u);
  \{ χ: efo: pass(χ(hf_u)) \} ∨ (\exists χ_1) (\forall χ: s\text{-}format-type: χ \& ~is-Q: χ(hf_u)) \cup
  \{ χ: efo: eval-intg-expr(χ(hf_u),E) \} ∨ (\exists χ_1) (\forall χ: s\text{-}format-type, χ \& ~is-Q: χ(hf_u)\} ∨
  (\exists χ_1) (\forall χ: s\text{-}format-type, χ \& ~is-Q: χ(hf_u)\}

where: fu-elem_u = (is-fu-elem\ u(U) → u(U)),
  T → error
  fo-sel_u = (elem(fo-i)) \& s\text{-}format-list
  fu-fol_u = fo-sel_u(fu-elem_u)
  hf_u = head(fu-fol_u)
  tf_u = tail(fu-fol_u)

Ref.: upd-fu-elem 10-6(13), rep-conc 6-17(3A), eval-intg-expr 7-21(52),
  op-val 8-11(40), eval-ref 7-19(46), is-fu-elem 2-35(113)

(139) verify-remote-format(fo-val) =
  is-format-den(fo-val) \&
  s-spp(CS) = s-spp(fo-val) \&
  s-e(fo-val) = E →
  PASS:s\text{-}format-list(fo-val)
  T → error

Ref.: is-format-den 2-20(50)
10.5.2 Data fields

Stream transmission handles correct or incorrect data fields. Incorrectness is detected if there is no continuation possible on the right of the partial data field which would make the whole data field correct. This section does not define the actions after detecting e.g., incorrectness but defines the necessary prerequisites for initiating these actions.

An abstract syntax of data fields is given in section 10.5.2.1 and 10.5.2.2, function and predicate schemes used in connection with the predicates of this abstract syntax are defined in section 10.5.2.3. Section 10.5.2.4 defines the values of arithmetic constants and complex expressions as they are needed in conversion from character string to arithmetic type.

10.5.2.1 Abstract syntax of constants

The abstract syntax of constants which are correct in list- or data-directed transmission is defined by the predicate is-io-data. This definition can be regarded as the transcription of the rules of the concrete syntax. Any object x which satisfies is-io-data(x) has at any node only selectors of the type elem(i) to existing subnodes with i>1; if i>1 then also the subnode selected by elem(i-1) must exist, and so on until elem(1). Such an object x is a nested list which is composed from characters of the data character set (satisfying is-char-val) if x satisfies is-io-data.

(140) is-io-data = is-io-arithm-data v is-io-char-string v is-io-bit-string

(141) is-io-arithm-data = is-io-arithm-const v is-io-cplx-expr

(142) is-io-arithm-const = (elem(1):is-< v is-io-sign>,
<elem(2):is-io-real-const>,
<elem(3):is-< v is-I-CHAR>)

(143) is-io-cplx-expr = (elem(1):is-< v is-io-sign>,
<elem(2):is-io-real-const>,
<elem(3):<elem(1):is-io-sign>,
<elem(2):is-io-real-const>,
<elem(3):is-I-CHAR>)

10.5.2.1
(144) is-io-sign = is-PLUS v is-PLUS

(145) is-io-char-string = (\<elem(1)\>:is-APOSTR>,
\<elem(2)\>:is-io-string-char-list>,
\<elem(3)\>:is-APOSTR>)

(146) is-io-bit-string = (\<elem(1)\>:is-APOSTR>,
\<elem(2)\>:is-io-bit-char-list>,
\<elem(3)\>:is-APOSTR>,
\<elem(4)\>:is-B-CHAR>)

(147) is-io-integer = (\<elem(1)\>:is-<> v is-io-integer>,
\<elem(2)\>:is-digit>)
Ref.: is-digit 0-5(12)

(148) is-io-bit-char = is-O-CHAR v is-L-CHAR

Note: Any bit string appears in a character representation.

(149) is-io-real-const = is-io-decimal-fixed v is-io-decimal-float v
is-io-binary-fixed v is-io-binary float

(150) is-io-decimal-fixed = (\<elem(1)\>:is-io-integer>,
\<elem(2)\>:is-<> v is-POINT>) v
(\<elem(1)\>:is-<> v is-io-integer>,
\<elem(2)\>:is-POINT>,\<elem(3)\>:is-io-integer>)

(151) is-io-decimal-float = (\<elem(1)\>:is-io-decimal-fixed>,
\<elem(2)\>:is-io-exponent)

(152) is-io-exponent = is-prop-exponent v is-editdir-exponent

(153) is-prop-exponent = (\<elem(1)\>:is-E-CHAR>,
\<elem(2)\>:is-<> v is-io-sign>,
\<elem(3)\>:is-io-integer>)

(154) is-editdir-exponent = (\<elem(1)\>:is-<>,
\<elem(2)\>:is-io-sign>,
\<elem(3)\>:is-io-integer>)

Note: This type of exponent is legal with E-format only.
(155) is-io-binary-fixed = (elem(1):is-io-binary-fixed-part;
                elem(2):is-B-CHAR))

(156) is-io-binary-fixed-part = (elem(1):is-io-binary-integer,
               elem(2):is-<> v is-POINT) v
               (elem(1):is-<> v is-io-binary-integer,
               elem(2):is-POINT,
               elem(3):is-io-binary-integer))

(157) is-io-binary-integer = (elem(1):is-<> v is-io-binary-integer,
               elem(2):is-io-bit-char)

Note: Any binary digit appears in a character representation.

(158) is-ios-binary-float = (elem(1):is-io-binary-fixed-part,
               elem(2):is-io-exponent,
               elem(3):is-B-CHAR))

(159) is-io-string-char(x) =
       (is-char-val(x) v is-io-string-apost(x)) & ~is-APOSTR(x)

Ref.: is-char-val 8-4(9)

(160) is-io-string-apost = (elem(1):is-APOSTR,
               elem(2):is-APOSTR)

10.5.2.2 Abstract syntax of Names

The abstract syntax of names which are allowed in data-directed transmission is defined by the predicate is-io-basic-ref. Similar as in section 10.5.2.1 for constants the definition can be regarded as the transcription of the rules of the concrete syntax.

(161) is-io-basic-ref = (elem(1):is-io-identifier,
               elem(2):is-io-qual-list,
               elem(3):is-<> v is-io-subspart))

(162) is-io-identifier = (elem(1):is-letter, elem(2):is-alpham-char-list)

Ref.: is-letter 8-4(11), is-alpham-char 3-4(19)

(163) is-io-qual = (elem(1):is-POINT-blanks, elem(2):is-io-identifier-blanks)
(164) is-io-sub-part = (elem(1):is-LEFT-PAR-blanks, 
  elem(2):is-io-sub-blanks, 
  elem(3):is-io-del-sub-list, 
  elem(4):is-RIGHT-PAR-blanks)

(165) is-io-del-sub = (elem(1):is-COMMA-blanks, elem(2):is-io-sub-blanks)

(166) is-io-sub(x) =
  is-io-arith-const(x) & is-io-decimal-fixed(elem(2,x)) &
  is<-&elem(3,x) & is-Q(elem(3,elem(2,x)))

10.5.2.3 Function and predicate schemes

The abstract syntax of data fields (as given in section 10.5.2.1
for constants and in section 10.5.2.2 for names) which is closely related
with a concrete syntax has to be brought into a relation with lists of
character values. This is necessary because stream transmission simply means
handling of lists of character values (this could be called the pure trans­
mission aspect), and on the other hand means discrimination between correct
and incorrect data fields, separation of the name part and the constant
part for data-directed transmission, etc. (this could be called the pure
syntax aspect). The functions and predicates defined in this section (with
two exceptions) have lists of character values as its arguments and yield
structured objects (in the case of functions) or classify the argument
according to syntax (in the case of predicates).

Since the full definition of all the functions and predicates would
prove to be lengthy and would be accompanied by much duplication the defini­
tions will be given in the form of schemes each of which standing for a
set of definitions.

Metavariables

cl is-char-val-list a list of character values
ncl is-ncl a nested list of character values
x, y, z -- an object

(167) is-ncl = is-char-val | is-ncl-list

10.5.2.3
10.5.2.3.1 Functions ending with "-parsing"

The definition of all functions ending with "-parsing" can be taken from a definition scheme given in this section. Substitution of a special name for all occurrences of "[name]" in the definition scheme yields the definition of one special function.

\[(168) \text{unnest-cl}(ncl) =\]

\[\text{is-<>}(ncl) \rightarrow <>\]
\[\text{is-char-val}(ncl) \rightarrow <ncl>\]
\[T \rightarrow \text{unnest-cl}\cdot\text{head}(ncl)\text{;; unnest-cl}\cdot\text{tail}(ncl)\]

Note: This function transforms a multiply nested list of character values into a simple list of character values. In other words, the function removes all structuring except the left-to-right ordering of the character values.

\[(169) [\text{name}]-\text{parsing}(ncl) =\]

\[(ncl-1)(\text{is-}[\text{name}](ncl-1) \& cl = \text{unnest-cl}(ncl-1))\]

Note: Let cl be a non-empty list of digits in character representation (is-digit-list(cl)). Then (by the definition of the predicates is-digit and is-io-integer) io-integer-parsing(cl) is the object satisfying is-io-integer, and (after unnesting) yields the list of digits cl.

This scheme defines the following functions basing on the following predicates:

- char-num-data-parsing
- listdir-proper-item-parsing
- datadir-ref-parsing
- datadir-rp-parsing
- f-data-parsing
- e-data-parsing

10.5.2.3.2 Predicates ending with "-blanks", "-corr", and "-leftcorr"

The definition of all predicates ending with "-blanks", "-corr" and "-leftcorr" can be taken from the corresponding definition schemes given in this section. The substitution of a special name for all occurrences of "[name]" in one of the definition schemes yields the definition of one special predicate.

10.5.2.3.2
(170) is-\{name\}-blanks(x) =

\( (\exists y,z) (is-BLANK-list(y) \& is-[name](z) \& x = \mu_{\alpha}(<\alpha_{\text{elem}(y)};y>,<\alpha_{\text{elem}(z)};z>)) \)

Note: This scheme defines the predicates following in the left column basing on the predicates of the right column:

is-COMMA-blanks	is-COMMA
is-io-prop-data-blanks	is-io-prop-data
is-io-basic-ref-blanks	is-io-basic-ref
is-EQ-blanks	is-EQ
is-SEMIC-blanks	is-SEMIC
is-POINT-blanks	is-POINT
is-io-identifier-blanks	is-io-identifier
is-LEFT-PAR-blanks	is-LEFT-PAR
is-io-subs-blanks	is-io-subs
is-RIGHT-PAR-blanks	is-RIGHT-PAR

(171) is-[name]-corr(cl) =

\( (\exists \text{ncl-1}) (is-[name]\text{ncl-1}) \& cl = \text{unnest-cl(ncl-1)} \)

Note: This scheme defines the predicates following in the left column basing on the predicates of the right column:

is-char-num-data-corr	is-char-num-data
is-COMMA-blanks-corr	is-COMMA-blanks
is-listdir-proper-item-corr	is-listdir-proper-item
is-SEMIC-blanks-corr	is-SEMIC-blanks
is-datadir-rp-corr	is-datadir-rp
is-datadir-ref-corr	is-datadir-ref
is-f-data-corr	is-f-data
is-e-data-corr	is-e-data

(172) is-[name]-leftcorr(cl) =

\( (\exists \text{cl-1}) (is-[name]-corr(cl\text{cl-1})) \)

Note: This scheme defines the predicates following in the left column basing on the predicates of the right column:

is-char-num-data-leftcorr	is-char-num-data
is-listdir-item-leftcorr	is-listdir-item
is-datadir-rp-leftcorr	is-datadir-rp
is-f-data-leftcorr	is-f-data
is-e-data-leftcorr	is-e-data

10.5.2.3
10.5.2.4 Arithmetic data fields

Arithmetic data fields can be composed from a single arithmetic constant or a so-called complex expression which in turn is the concatenation of a real and an imaginary arithmetic constant. Since the value of arithmetic data fields is needed in list- and data-directed transmission and in conversion from character string to arithmetic type it has been decided to incorporate in this section all the definitions which are relevant for transmission and conversion or for conversion only.

(173) \text{char-num-conv}(v) =

\text{is-char-num-data-corr}(v) \rightarrow \text{PASS: num-val} \cdot \text{elem}(2, \text{char-num-data-parsing}(v))

T \rightarrow \text{char-num-conv}(\text{corr-v});
\text{corr-v} \text{call-conv-cond}(v, i_0, 839)

where: \[ i_0 = \{(i_1 \mid i_1 \leq 1 \& (i_1 > \text{length}(v) \& \text{is-char-num-data-leftcorr}(v)) \}

\text{corr-v} = \text{call-conv-cond}(v, i_0, 839)

\text{for: is-char-val-list}(v)

Ref.: \text{call-conv-cond} 9-17(34), \text{is-char-num-data-corr} 10-60(171),
\text{char-num-data-parsing} 10-59(169), \text{is-char-num-data-leftcorr} 10-60(172)

Note: \( i_0 \) is "the position of the first wrong character"; in particular,
if \( v \) is incorrect only because it is too short, then \( i_0 = \text{length}(v)+1 \).

(174) \text{is-char-num-data} = \left(\text{<elem(1)\text{-is-BLANK-list>},}
\text{<elem(2)\text{-is-io-prop-arithm-data>},}
\text{<elem(3)\text{-is-BLANK-list>}}\right)

(175) \text{is-io-prop-arithm-data}(dt) =

\text{is-io-arithm-data}(dt) \& \text{is-e-exponent}(dt)

\text{for: is-io-arithm-data}(dt)

Ref.: \text{is-io-arithm-data} 10-55(141), \text{is-e-exponent} 10-72(203)

Note: This is to exclude the type of exponent which is legal with E-format only.
\[(176)\] \[\text{num-val}(dt) =\]

\[
\begin{align*}
&\text{is-io-arithm-const}(dt) \land \text{is-<>}(dt_2) \rightarrow \text{real-val}(dt) \\
&\text{is-io-arithm-const}(dt) \land \text{i-o-CHAR}(dt_2) \rightarrow \text{cplx}(0,\text{real-val}(dt_1)) \\
&\text{is-io-cplx-expr}(dt) \rightarrow \text{cplx}(\text{real-val}(dt_1),\text{real-val}(dt_2))
\end{align*}
\]

\[\text{where: } dt_1 = f(dt_1, \text{<elem(3):<>>})
\]
\[dt_2 = \text{elem}(3, dt)
\]
\[\text{for: } \text{is-io-arithm-data}(dt)
\]

\[(177)\] \[\text{real-val}(dt) =\]

\[\text{real-val-1}(dt, *)\]

\[\text{for: } \text{is-io-arithm-const}(dt)\]

\[(178)\] \[\text{real-val-1}(dt, \text{base}) =\]

\[
\begin{align*}
&\text{is-<>}(dt) \lor \text{is-O}(dt) \lor \text{i-o-CHAR}(dt) \rightarrow 0 \\
&\text{is-digit}(dt) \rightarrow \text{single-char-num}(dt) \\
&\text{is-io-arithm-const}(dt) \lor \text{is-io-exponent}(dt) \rightarrow \\
&\text{sg}_{dt} \cdot \text{real-val-1}(\text{pr}_{dt}, \text{bs}_{dt}) \\
&\text{is-io-binary-integer}(dt) \lor \text{is-i-o-binary-integer}(dt) \rightarrow \\
&\text{base} \cdot \text{real-val-1}(\text{head}(dt), \text{base}) + \text{real-val-1}(\text{elem}(2, dt), \text{base}) \\
&\text{is-i-o-decimal-fixed}(dt) \lor \text{is-io-binary-fixed-part}(dt) \rightarrow \\
&\text{real-val-1}(\text{head}(dt), \text{base}) + \text{real-val-1}(\text{elem}(3, dt), \text{base}). \text{base}^{\text{li}_{dt}} \\
&\text{is-io-decimal-float}(dt) \lor \text{is-io-binary-float}(dt) \rightarrow \\
&\text{real-val-1}(\text{head}(dt), \text{base}) \cdot \text{base}^{\text{li}_{dt}} \text{real-val-1}(\text{elem}(2, dt), \text{base})
\end{align*}
\]

\[\text{where: } \text{sg}_{dt} = (\text{is-MINUS}(\text{head}(dt)) \lor \text{is-MINUS}(\text{elem}(2, dt)) \rightarrow -1, T \rightarrow 1)
\]
\[\text{pr}_{dt} = (\text{is-io-exponent} \rightarrow \text{elem}(3, dt),
\text{is-io-binary-fixed} \rightarrow \text{elem}(2, dt) \rightarrow \text{head} \cdot \text{elem}(2, dt),
T \rightarrow \text{elem}(2, dt))
\]
\[\text{bs}_{dt} = (\text{is-B-CHAR} \rightarrow \text{elem}(2, \text{elem}(2, dt)) \lor
\text{is-B-CHAR} \rightarrow \text{elem}(3, \text{elem}(2, dt)) \rightarrow 2,
T \rightarrow 10)
\]
\[\text{li}_{dt} = \text{length} \cdot \text{unnest-cl}(dt)
\]
\[\text{for: } \text{is-io-arithm-const}(dt) \lor (\text{is-}(\text{base}) \lor \text{base} = 10 \lor \text{base} = 2)
\]

Ref. : all predicates beginning with "is-io" see section 10.5.2.1;
unnest-cl 10-59(168), single-char-num 8-34(133), is-digit 8-5(12)

10.5.2.4
10.5.3 Get statement

This section contains the definition of the get statement which names a file (is-file-get) or a string (is-string-get).

Abbreviations

\[ \text{fu-elem}_u = (\text{is-fu-elem}_u(FU) \rightarrow u(FU), T \rightarrow \text{error}) \]

Ref.: is-fu-elem 2-35(113)

\[ \text{is-string}_u = \text{is-fu-string}_u(FU) \]

Ref.: is-fu-string 2-35(114)

\[ \text{is-file}_u = \text{is-fu-file}_u(FU) \]

Ref.: is-fu-file 2-36(117)

Metavariables

\[ t \quad \text{is-get-st} \quad \text{the text of a get statement} \]
\[ u \quad \text{is-n} \quad \text{a file union name} \]
\[ w \quad \text{is-intg-val} \quad \text{an is-intg-val} \]
\[ \text{op} \quad \text{is-op} \quad \text{an is-op} \]
\[ \text{cy} \quad \text{is-opt} \quad \text{an is-opt} \]
\[ \text{fo-i} \quad \text{is-intg-val} \quad \text{an is-intg-val} \]
\[ \text{tr} \quad \text{is-tr} \quad \text{an is-tr} \]
\[ \text{itl} \quad \text{is-item-list} \quad \text{an is-item-list} \]
\[ g \quad \text{is-gen} \quad \text{an is-gen} \]
\[ gl \quad \text{is-gen-list} \quad \text{an is-gen-list} \]
\[ lg \quad \text{is-intg-val} \quad \text{an is-intg-val} \]
\[ \text{efo} \quad \text{---} \quad \text{an is-char-val} \]
\[ \text{edfo} \quad \text{---} \quad \text{an is-char-val} \]
\[ \text{c} \quad \text{is-char-val} \quad \text{a character value} \]
\[ \text{cl} \quad \text{is-char-val-list} \quad \text{a list of character values} \]
10.5.3.1 Control oriented part

(179) \[ \text{int-get-st}(t) = \]
\[ \text{is-file-get}(t) \mapsto \]
\[ \text{get-data-spec}(t,u,
\psi) ; \]
\[ \text{init-file-2}(u,s\cdot\text{copy}(t),w) ; \]
\[ w: \text{eval-intg-expr}(s\cdot\text{skip}(t),E) ; \]
\[ \text{init-file-1}(u) ; \]
\[ \text{get-endfile}(u) ; \]
\[ u: \text{int-impl-open}(t) \]
\[ \text{is-string-get}(t) \mapsto \]
\[ \text{get-data-spec}(t,u,
\psi) ; \]
\[ \text{init-string-1}(s\cdot\text{spec}(t),u,op) ; \]
\[ u: \text{un-name}, \]
\[ op: \text{eval-ref}(s\cdot\text{string}(t),E) \]

Ref.: \[ \text{int-impl-open} \ 10-13(32), \text{eval-intg-expr} \ 7-21(52), \]
\[ \text{un-name} \ 2-17(36), \text{eval-ref} \ 7-19(46), \]
\[ \text{is-file-get} \ 12-14(113), \text{is-string-get} \ 12-14(114) \]

(180) \[ \text{get-endfile}(u) = \]
\[ \text{is-}\varnothing(\text{fu-endf}_u) \mapsto \text{null} \]
\[ T \mapsto \text{call-io-cond}(u,\varnothing,\text{ENDFILE},\varnothing,840) ; \]

where: \[ \text{fu-endf}_u = s\cdot\text{endfile}(\text{fu-elem}_u) \]

Ref.: \[ \text{call-io-cond} \ 10-30(86) \]
(181) \text{init-file-1}(u) =

\text{null;}
\text{upd-fu-elem}(u, s\text{-count}, \text{bintg-op}(0)),
\text{upd-fu-elem}(u, s\text{-onsource}, \emptyset)

Ref.: bintg-op 8-11(43), upd-fu-elem 10-6(13)

(182) \text{init-file-2}(u, cy, w) =

is\text{-intg-val}(w) \rightarrow
\text{get-skip}(u, cy, w, *) ;
\text{get-copy}(u, cy, F);
\text{get-copy}(u, cy, T)

T \rightarrow \text{get-copy}(u, cy, T)

Ref.: get-copy 10-86(243), get-skip 10-80(226)

Note: A consistency check for the skip option is made in verify-impl-open(t).

(183) \text{init-string-1}(\text{spec}, u, op) =

(is\text{-CHAR}s\text{-base}s\text{-da}(op) \vee is\text{-char-pic}s\text{-da}(op)) \& \neg is\text{-<>}(\text{spec}) \rightarrow
s\text{-fu}: \mu(FU; <u\mu_0(<s\text{-data}s\text{-vr}(op)>, <s\text{-comma}T>))

T \rightarrow \text{error}

for: is\text{-data-spec-list}(\text{spec})

Note: Value representations of LINE-DELMITER and FILEMARK are not excluded. However, they are treated in the same way as value representations of any other characters.
(184) \[ \text{get-data-spec}(t, u, \text{fo-i}) = \]
\[
\begin{align*}
\text{is-<}(\text{s-spec}(t)) & \rightarrow \text{null} \\
\text{is-edit-directed}(\text{spec-1}_t) & \land \ \\
(\text{is-}< (\text{spec-2}_t) \lor \text{is-edit-directed}(\text{spec-2}_t)) & \land \\
\neg \text{is-<}(\text{itl}_t) & \land \neg \text{is-<}(\text{fol}_t) \\
& \rightarrow \\
\text{get-data-spec}(t\text{-tail}_t, u, \text{fo-i}_\text{tr}); \\
\text{get-list-edit}(u, e_{\text{tr}}/\text{itl}_t); \\
\text{upd-fu-elem}(u, s\text{-comma}, T)
\end{align*}
\]
\[
\text{is-data-directed}(\text{spec-1}_t) \\
\rightarrow \\
\text{get-data-lp}(u, s\text{-copy}(t), <, \text{idl}_{\text{d}}); \\
\text{get-copy}(u, s\text{-copy}(t), F)
\]
\[
\text{is-list-directed}(\text{spec-1}_t) \land \neg \text{is-<}(\text{itl}_t) \\
\rightarrow \\
\text{get-list-edit}(u, l_{\text{tr}}, \text{itl}_t)
\]
\[ T \rightarrow \text{error} \]

where: \[
\begin{align*}
\text{spec-1}_t & = \text{head}\cdot \text{s-spec}(t) \\
\text{spec-2}_t & = \text{elem}(1, \text{tail}\cdot \text{s-spec}(t)) \\
\text{t-tail}_t & = \mu(t;<\text{s-spec};\text{tail}\cdot \text{s-spec}(t)> ) \\
\text{itl}_t & = \text{s-data-list}(\text{spec-1}_t) \\
\text{fol}_t & = \text{s-format-list}(\text{spec-1}_t) \\
\text{fo-i}_\text{tr} & = (\text{is-int-}\text{val}(\text{fo-i}) \rightarrow \text{fo-i}, \neg \text{is-}<\text{spec-2}_t) \\
\text{fo-i}_\text{tr} & \rightarrow \text{s-format-list}(\text{fu-elem}_u) \rightarrow \\
\text{length}(\text{s-data-list}(\text{spec-1}_t)) + 1, \\
T & \rightarrow 1
\end{align*}
\]
\[
\text{e}_{\text{tr}} = \mu_o(<\text{s-type};\text{fol}_t>, <\text{s-fo-i};\text{fo-i}_\text{tr}>, <\text{s-copy};\text{spec-1}_t>) \\
\text{l}_{\text{tr}} = \mu_o(\text{\textquotedblleft}s-type\text{"};>, <\text{s-copy};\text{spec-1}_t>) \\
\text{idl}_{\text{d}} = \text{assoc-idl-set}(\text{s-data-list}(\text{spec-1}_t), E, AT)
\]

Ref.: \[ \text{upd-fu-elem} \text{ 10-6(13), get-copy 10-86(243)} \]

(185) \[ \text{assoc-idl-set}(\text{itl}, \text{env}, \text{at}) = \]
\[
\begin{align*}
\text{is-<}(\text{itl}) & \rightarrow \text{datadir-idl-set}(\text{env}, \text{at}) \\
\text{is-basic-ref-list}(\text{itl}) & \rightarrow \ \\
\text{length}(\text{itl}) \\
( \bigcup_{i=1} \text{assoc-idl(s-id-list}\cdot\text{elem}(i, \text{itl}), \text{env})) \cap \\
\text{datadir-idl-set}(\text{env}, \text{at})
\end{align*}
\]
\[ T \rightarrow \text{error} \]
for: is-e(env) & is-at(at)

Note: The second alternative yields the intended interpretation only if the basic reference itl is a fully qualified reference. This is not a requirement for the concrete program text.

(186) $\text{datadir-idl-set}(env, at) =$

$$\{\text{idl} \mid \text{sel}(\text{idl})(env) \in \text{datadir-n-set}(env, at)\}$$

for: is-e(env) & is-at(at)

(187) $\text{datadir-n-set}(env, at) =$

$$\{n \mid \text{is-n}(n) \& \text{is-prop-vars-attrn}(at) \& \neg \text{is-PARAM-scopes-attrn}(at)\}$$

for: is-e(env) & is-at(at)
Ref.: is-n 2-17(35)

(188) $\text{assoc-idl}(idl, env) =$

$$\{\text{idl}_1 \mid \text{is-n}(n_{\text{idl}_1}) \& n_1 = n_{\text{idl}} \& \exists \text{idl}_2 (\text{idl}_1 \land \text{idl}_2 = \text{idl}_1 - 1)\}$$

where: $n_{\text{idl}} = \text{sel}(\text{idl})(env)$
$n_1 = \text{sel}(\text{idl}-1)(env)$
for: is-e(env)
Ref.: is-n 2-17(35)

(189) $\text{get-list-edit}(u, tr, itl) =$

$$\begin{align*}
\text{is-}\llangle(\text{itl}) \rightarrow \text{null} \\
\text{is-ref}(\text{itl}_h) \rightarrow \\
\text{get-list-edit}(u, tr, \text{itl}_t); \\
\text{call-check-cond}(\llangle(\text{itl}_h), E, 841); \\
\text{get-scalar}(u, tr, \text{itl}_h, \text{eda}); \\
\text{eda:eda-expr}(\text{itl}_h, E) \\
\text{is-contr-item}(\text{itl}_h) \rightarrow \\
\text{get-list-edit}(u, tr, \text{itl}_t); \\
\text{int-contr-item}(\text{GET}, u, tr, \text{itl}_h) \\
T \rightarrow \text{error}
\end{align*}$$

where: $\text{itl}_h = \text{head}(\text{itl})$
$\text{itl}_t = \text{tail}(\text{itl})$

Ref.: $\text{call-check-cond}$ 9-15(27), $\text{int-contr-item}$ 10-52(131), $\text{eda-expr}$ 7-22(54)
(190) \texttt{get-scalar(u, tr, tm, eda)} = \\
\texttt{is-scalar(eda) \rightarrow} \\
\texttt{count-convert-assign(u, tr, g, op);} \\
\texttt{op: get-data-field(u, tr);} \\
\texttt{g: eval-lp(tm).} \\
T \rightarrow \\
\texttt{iterate-get(u, tr, tm, eda, lbd(eda))} \\
for: \texttt{is-expr(tm) \& is-eda(eda)} \\
\texttt{Ref.: eval-lp 7-4(5)} \\

(191) \texttt{iterate-get(u, tr, tm, eda, i)} = \\
i > \texttt{ubd(eda) \rightarrow null} \\
T \rightarrow \\
\texttt{iterate-get(u, tr, tm, eda, i + 1);} \\
\texttt{get-scalar(u, tr, tm-1, da-part(eda, i));} \\
\texttt{tm-1: mod(tm, eda, i, \Sigma)} \\
for: \texttt{is-expr(tm) \& is-eda(eda) \& is-intg-val(i)} \\
\texttt{Ref.: mod 7-7(13)} \\

(192) \texttt{count-convert-assign(u, tr, g, op)} = \\
\texttt{is-<>(op) \& is-edit-tr(tr) \& \sim is-\Omega: gen-base(g)} \rightarrow \\
\texttt{count-convert-assign(u, tr, g, str-op);} \\
\texttt{str-op: pass-val-op(da_{o, \sim})} \\
\texttt{is-\Omega(op) \lor is-<>(op) \rightarrow null} \\
\texttt{is-file_u \rightarrow} \\
\texttt{convert-assign(g, op);} \\
\texttt{verify-transmit(u, 842)} \\
\texttt{upd-fu-elem(u, s-count, bintg-op);} \\
\texttt{bintg-op: add-op.Expr(fu-count_{u}, bintg-op(1))} \\
\texttt{is-string_u \rightarrow} \\
\texttt{convert-assign(g, op)} \\
\texttt{where: fu-count_{u} = s-count(fu-elem_{u})} \\
\texttt{da_{o} = \mu_{o}(<s-base: gen-base(g)>, <s-length:0>)} \\
\texttt{cont'd}
(193) get-data-field\(u, tr\) =

\[
is-list-tr(tr) \rightarrow
\]

\[
\text{get-listdir}(u, s-copy(tr), \phi);
\]

\[
\text{get-copy}(u, s-copy(tr), F)
\]

\[
is-edit-tr(tr) \rightarrow
\]

\[
\text{mk-editdir-op}(cl, edfo);
\]

\[
\text{cl} : \text{get-char-list}(u, s-copy(tr), cl);
\]

\[
\text{lg} : \text{pass-get-fold}(edfo);
\]

\[
edfo \text{: get-data-format}(u, tr)
\]

Ref.: get-copy 10-86(243), get-listdir 10-71(199), get-fold 10-81(227),
get-char-list 10-82(228), mk-editdir-op 10-82(229),
is-list-tr 10-52(136), is-edit-tr 10-53(137)

(194) get-data-format\(u, tr\) =

\[
\text{get-contr-format}(u, tr, efo);
\]

\[
efo \text{: eval-format}(u, s-type(tr), s-fo-i(tr));
\]

\[
\text{get-copy}(u, s-copy(tr), F)
\]

Ref.: get-copy 10-86(243), eval-format 10-53(138)

(195) get-contr-format\(u, tr, efo\) =

\[
is-SPACE(ft_e) \& is-int-val(w_e) \rightarrow
\]

\[
\text{get-data-format}(u, tr);
\]

\[
\text{get-space}(u, s-copy(tr), w_e)
\]

\[
is-COLUMN(ft_e) \& is-int-val(w_e) \& is-file_u \rightarrow
\]

\[
\text{get-data-format}(u, tr);
\]

\[
\text{get-column}(u, s-copy(tr), w_e)
\]

cont'd
10.5.3.2 Elementary transmission

Elementary transmission from a stream source, i.e., input by a file or transmission from a string, is the transmission of a single character value. This section is related with section 10.5.3.5.1 (Control Formats).

Abbreviations

fu-ons_u = s-onsource(fu-elem_u)

fu-pos_u = s-position(fu-elem_u)

(196) \( \text{get-next-char}(u, cy) = \)

\( \text{get-space}(u, cy, 1) \)

Ref.: \text{get-space} 10-78(221)

Note: The spacing control format applied to any stream source is defined to be value returning. The definition of the spacing control format for files generally leads to the execution of the instruction \( \text{tk-data} \).

(197) \( \text{tk-data}(u, cy) = \)

\( \neg \text{is-}<\left(fu-ons_u\right) \rightarrow \text{tk-onsource}(u) \)

\( T \rightarrow \)

\( \text{pass}(in-q); \)

\( \text{get-copy}(u, cy, in-q); \)

\( \text{in-q}: \text{pass-elem}(fu-pos_u, ids); \)

\( ids:\text{tk-dataset}(u, Q); \)

\( \text{upd-fu-elem}(u, s-position, fu-pos_u + 1) \)

Ref.: \text{upd-fu-elem} 10-6(13), \text{tk-dataset} 10-7(16), \text{get-copy} 10-86(243)
10.5.3.3 List-directed transmission

This section defines transmission from a stream source in list-directed mode.

Predicates with names starting with "is-io" which are not cited in the references following a formula, should be looked up in sections 10.5.2.1 and 10.5.2.2.

(198) \text{tk-onsource}(u) =
\begin{align*}
\text{pass-head}(\text{fu-ons}_u); \\
\text{upd-fu-elem}(u, s\text{-onsource, tail}(\text{fu-ons}_u))
\end{align*}

Ref.: \text{upd-fu-elem} 10-6(13)

(199) \text{get-listdir}(u, cy, cl) =
\begin{align*}
\text{is-COMMA-blanks-corr}(cl) \land is-\varnothing(\text{fu-comma}_u) & \rightarrow \text{get-listdir}(u, cy, <>); \\
\text{upd-fu-elem}(u, s\text{-comma}, T) \\
\text{is-COMMA-blanks-corr}(cl) & \rightarrow \text{null} \\
is\text{-listdir-proper-item-corr}(cl) & \rightarrow \text{mk-io-data-op}(\text{dt}_c); \\
\text{upd-fu-elem}(u, s\text{-comma, CO}_c) \\
is\text{-listdir-item-leftcorr}(cl) & \rightarrow \text{get-listdir}(u, cy, cl-1); \\
\text{cl-1} : \text{mk-list-1}(cl, c); \\
\text{c} : \text{get-next-char}(u, cy) \\
T & \rightarrow \text{get-listdir}(u, cy, <>); \\
\text{upd-fu-elem}(u, s\text{-onsource, ons}); \\
on\text{s\text{-call-conv-cond}(cl, length(cl), 843)}
\end{align*}

where: \text{fu-comma}_u = s\text{-comma}(\text{fu-elem}_u) \\
\text{dt}_c = \text{elem}(2, \text{elem}(1, \text{listdir-proper-item-parsing}(cl))) \\
\text{CO}_c = (\text{is-COMMA}\text{-elem}(\text{length}(cl), cl) \rightarrow T, T \rightarrow \varnothing)

Ref.: \text{upd-fu-elem} 10-6(13), \text{get-next-char} 10-70(196), \text{call-conv-cond} 9-17(34), \text{is-COMMA-blanks-corr} 10-60(171), \text{is-listdir-proper-item-corr} 10-60(171), \text{is-listdir-item-leftcorr} 10-60(172), \text{listdir-proper-item-parsing} 10-59(169)
(200) \textit{is-listdir-item} = \textit{is-listdir-proper-item} \lor \textit{is-COMMA-blanks}

Ref.: \textit{is-COMMA-blanks} 10-60(170)

(201) \textit{is-listdir-proper-item} = (\langle \text{elem(1)} : \text{is-prop-data-blanks} \rangle,
\langle \text{elem(2)} : \text{BLANK} \lor \text{is-COMMA} \rangle)

Ref.: \textit{is-io-prop-data-blanks} 10-60(170)

(202) \textit{is-io-prop-data(dt)} =
\textit{is-io-data(dt)} \land \neg \textit{is-e-exponent(dt)}

for : \textit{is-io-data(dt)}

Note: This is to exclude the type of exponent which is legal with E-format only.

(203) \textit{is-e-exponent(dt)} =

\((\forall x) (\textit{is-io-decimal-float} \times x(dt) \lor
\textit{is-io-binary-float} \times x(dt))
\textit{is-prop-exponent-elem(2,x(dt))})

for : \textit{is-io-data(dt)}

Ref.: \textit{is-prop-exponent} 10-56(153)

(204) \textit{mk-io-data-op(dt)} =

\textit{is-io-arithmetic-const(dt)} \rightarrow

\text{mk-op}(da_{\text{dt}},vr);
vr:rep-real(real-da(dt),real-val(dt))
\textit{is-io-cplx-expr(dt)} \rightarrow

\text{mk-op}(da,vr);
da:pass-new(CPLX,pref(real-da(re_{\text{dt}}),real-da(im_{\text{dt}}))),
vr:pass-cplx-vr(vr-re,vr-im);
vrr-re:rep-real(real-da(re_{\text{dt}}),real-val(re_{\text{dt}})),
vrr-im:rep-real(real-da(im_{\text{dt}}),real-val(im_{\text{dt}}))
\textit{is-io-char-string(dt)} \rightarrow pass-val-op(char-da \times length(v_c),v_c)
\textit{is-io-bit-string(dt)} \rightarrow pass-val-op(bit-da \times length(v_b),v_b)

where: \text{da}_{\text{dt}} = (\text{is-I-CHAR \times elem(3,dt)} \rightarrow \text{new(CPLX,real-da(dt))},T \rightarrow \text{real-da(dt)})
\text{re}_{\text{dt}} = (dt;\langle \text{elem(3)}:<<>>\rangle)

cont'd
\[ \begin{align*}
im_{dt} &= \text{elem}(3, dt) \\
v_1 &= \text{elem}(i, \text{elem}(2, dt)) \\
v_{1dt} &= \text{length} \cdot \text{elem}(2, dt) \\
v_c &= \mathbf{L I S T} \begin{array}{c} \text{vc}_i \\
i \end{array} \\
v_{c1} &= (\text{is-io-string-apost}(v_1) \rightarrow \text{APOSTR}, T \rightarrow v_1) \\
v_{dt} &= \mathbf{L I S T} \begin{array}{c} \text{single-char-bit}(v_i) \\
i \end{array} \\
v_b &= \mathbf{L I S T} \begin{array}{c} \text{single-char-bit}(v_i) \\
i \end{array} \\
\text{for: } &\text{is-io-data}(dt) \\
\text{Ref.: } &\text{rep-real 8-9(31), mk-op 8-11(39), cplx-vr 8-9(30),}
\text{new 11-30(41), pref 11-32(45), val-op 8-11(41), char-da 8-13(60),}
\text{bit-da 8-13(61), single-char-bit 8-35(135)} \\
\end{align*} \]

\[ \begin{align*}
(205) &\text{real-da}(dt) = \\
&\mu(d_c; s\text{-}\text{prec}: \min(\text{max-prec}(d_c), s\text{-}\text{prec}(d_c))) \\
&\text{where: } d_c = \text{real-app-da} \cdot \text{elem}(2, dt) \\
&\text{for: } \text{is-io-arithm-const}(dt) \\
&\text{Ref.: max-prec 8-13(62)} \\
\end{align*} \]

\[ \begin{align*}
(206) &\text{real-app-da}(dt) = \\
&\text{is-io-decimal-fixed}(dt) \lor \text{is-io-binary-fixed}(dt) \\
&\mu_0(\langle s\text{-}\text{mode}: \text{REAL}, s\text{-}\text{base}: \text{BS}_{dt}, s\text{-}\text{scale}: \text{FIX},
\text{is-prec}: \text{head}(f_p) + \lg_3, \langle s\text{-}\text{scale}: f; \lg_3 \rangle) \\
&\text{is-io-decimal-float}(dt) \lor \text{is-io-binary-float}(dt) \\
&\mu(\delta(\text{real-da} \cdot \text{head}(dt); s\text{-}\text{scale}-f), \langle s\text{-}\text{scale}: \text{FLT} \rangle) \\
&\text{where: } \text{BS}_{dt} = (\text{is-io-decimal-fixed}(dt) \rightarrow \text{DEC}, T \rightarrow \text{BIN}) \\
&\lg_3 = (\text{is- } \text{elem}(3, f_p) \rightarrow 0, T \rightarrow \text{length} \cdot \text{elem}(3, f_p)) \\
f_p = (\text{is-io-decimal-fixed}(dt) \rightarrow dt, T \rightarrow \text{head}(dt)) \\
&\text{for: } \text{is-io-real-const}(dt) \\
\end{align*} \]
10.5.3.4 Data directed transmission

This section defines transmission from a stream source in data-directed mode.

Predicates with names starting with "is-io" which are not cited in the references following a formula, should be looked up in sections 10.5.2.1 and 10.5.2.2.

Metavariables

- id: is-id an identifier
- idl: is-id-list a list of identifiers
- idl-set: is-id-list-set a set of lists of identifiers which is associated with a certain get statement
- b-ref: is-basic-ref v a basic reference which is optional
- v: is-\{\} a basic reference which is optional
- opl: is-op-list a list of operands (constants)
- lp-cl: is-char-val-list a left part of a data-directed data field
- rp-cl: is-char-val-list a right part of a data-directed data field

(207) get-data-lp(u,cy,cl,idl-set) =

is-COMMA-blanks-corr(cl) → get-data-lp(u,cy,<>idl-set)
is-SEMIC-blanks-corr(cl) → int-next-st
is-eq cl →

test-group-end(u,cy,idl-set,last rp);
get-data(u,b-ref,cl,rp-cl,idl-set);
b-ref:expand-basic-ref(cl),
rp-cl:get-data-rp(u,cy,<>)

T →

get-data-lp(u,cy,cl-l,idl-set);
cl-l:mk-list-1(cl,c);
c:get-next-char(u,cy)

where: is-eq cl = (¬is-<>(cl) & is-END-elem(length(cl),cl) → T, T → F)
last rp = elem(length(rp-cl),rp-cl)

Ref.: int-next-st 5-16(24), is-COMMA-blanks-corr 10-60(171),
get-next-char 10-70(196), is-SEMIC-blanks-corr 10-60(171)
(208) \( \text{get-data-rp}(u, cy, cl) = \)

\[
is\text{-datadir-rp-corr}(cl) \rightarrow \text{PASS: cl}
\]

\[
is\text{-datadir-rp-leftcorr}(cl) \rightarrow \text{get-data-rp}(u, cy, cl-1);
\]

\[
\text{cl-1} = \text{mk-list-l}(cl, c);
\]

\[
c : \text{get-next-char}(u, cy)
\]

\[
\rightarrow \text{get-data-rp}(u, cy, <>);
\]

\[
\text{upd-fu-elem}(u, s-onsource, ons);
\]

\[
\text{ons} \text{: call-conv-cond}(cl, \text{length}(cl), 844)
\]

Ref.: \text{get-next-char 10-70(196)}, \text{upd-fu-elem 10-6(13)}, \text{call-conv-cond 9-17(34)},

\( \text{is\text{-datadir-rp-corr 10-60(171), is\text{-datadir-rp-leftcorr 10-60(171)}} \)

(209) \( \text{is\text{-datadir-rp} = } \left< \text{elem}(1) : \text{is\text{-io-prop-data-blanks}} \right>, \left< \text{elem}(2) : \text{is\text{-BLANK}} \lor \text{is\text{-COMMA}} \lor \text{is\text{-SEMIC}} \right> \)

Ref.: \text{is\text{-io-prop-data-blanks 10-60(170)}}

(210) \( \text{expand-basic-ref}(cl) = \)

\[
\text{is\text{-datadir-ref-corr}(cl) } \rightarrow \text{mk-basic-ref(idl, opl)};
\]

\[
\text{idl: pass-ref-idl(io-b-ref cl)}, \text{opl: mk-ref-opl(ref-arg1 elem(3, io-b-ref cl))}
\]

\[
\rightarrow \text{null}
\]

where: \( \text{io-b-ref cl} = \text{elem}(2, \text{elem}(1, \text{datadir-ref-parsing}(cl))) \)

Ref.: \text{is\text{-datadir-ref-corr 10-60(171)}, datadir-ref-parsing 10-59(169)}

(211) \( \text{is\text{-datadir-ref} } = \left< \text{elem}(1) : \text{is\text{-io-basic-ref-blanks}} \right>, \left< \text{elem}(2) : \text{is\text{-EQ-blanks}} \right> \)

Ref.: \text{is\text{-io-basic-ref-blanks 10-60(170)}, is\text{-EQ-blanks 10-60(170)}}

(212) \( \text{ref-idl(io-b-ref) =} \)

\[
\text{length(idl)}
\]

\[
\text{LIST}
\]

\[
i = 1
\]

\[
\text{mk-id\text{-unnest-cl}\text{-elcm}(i, idl)}
\]

where: \( \text{idl} = \text{head(io-b-ref) } \cap \text{expand-part\text{-elem}(2, io-b-ref)} \)

for: \text{is\text{-io-basic-ref(io-b-ref)}

10.5.3.4
(213) \( \text{expand-part(list)} = \)
\[
\begin{align*}
\text{is-<}(\text{list}) & \rightarrow < \\
\text{is-io-qual-list(list)} \lor \text{is-io-del-subs-list(list)} & \rightarrow \\
<\text{elem}(2,\text{elem}(2,\text{head(list)}))> & \sim \text{expand-part-tail(list)}
\end{align*}
\]
\[\text{for: is-io-qual-list(list)} \lor \text{is-io-del-subs-list(list)}\]

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

Note: This function maps lists of character values \( \text{(is-char-val-list(cl))} \) into identifiers \( \text{(is-id)} \). The following two equations hold:

\[\begin{align*}
a) \quad & (\forall \text{cl}) (\text{is-char-val-list(cl)} \supset (\exists \text{id}) (\text{is-id(id)} \land \text{id} = \text{mk-id(cl)})) \\
b) \quad & (\forall \text{cl-1, cl-2}) (\text{is-char-val-list(cl-1)} \land \text{is-char-val-list(cl-2)} \supset \\
& \quad \text{cl-1} = \text{cl-2} \equiv \text{mk-id(cl-1)} = \text{mk-id(cl-2)}
\end{align*}\]

In some applications of the function \( \text{mk-id} \) the following shorthand notation is used for the argument, e.g.

\( \text{mk-id(ABS)} \) stands for \( \text{mk-id(\langle A-CHAR, B-CHAR, S-CHAR\rangle)} \)

(215) \( \text{chl(id)} = \)
\[
(\forall \text{cl}) (\text{id} = \text{mk-id(cl)})
\]

Note: \( \text{chl\cdot mk-id(ABS)} = \langle A-CHAR, B-CHAR, S-CHAR\rangle \)

(216) \( \text{ref-argl(io-subs-part)} = \)
\[
\begin{align*}
\text{is-<}(\text{io-subs-part}) & \rightarrow < \\
\text{T} & \rightarrow <\text{elem}(2,\text{elem}(2,\text{io-subs-part}))> \sim \text{expand-part-\text{elem}(3,\text{io-subs-part)}}
\end{align*}
\]
\[\text{for: is-io-subs-part(io-subs-part)}\]

(217) \( \text{mk-ref-opl(argl)} = \)
\[
\begin{align*}
\text{is-<}(\text{argl}) & \rightarrow \text{PASS: <} \\
\text{T} & \rightarrow \\
\text{mk-list(op,opl)}; \\
\text{opl} & : \text{mk-ref-opl(tail(argl))}; \\
\text{op} & : \text{mk-io-data-op(head(argl))}
\end{align*}
\]
\[\text{for: is-io-subs-list(argl)}\]

Ref.: \( \text{mk-io-data-op \ 10-72(204)} \)
(218) \( \text{mk-basic-ref}(\text{idl}, \text{opl}) = \)

\[
\text{PASS: } \mu_0(<\text{idl-list}: \text{idl}>,<\text{arg-list}: \text{opl}>)
\]

(219) \( \text{get-data}((u,b-ref,lp-cl,rp-cl,\text{idl-set})) = \)

\[
\text{is-basic-ref}(b-ref) \land \text{s-id-list}(b-ref) \in \text{idl-set} \\
\text{call-check-cond}(<b-ref>,845) \\
\text{count-convert-assign}(u, \varnothing, \text{op}) ; \\
\text{op: mk-io-data-op}(dt_{rp}), \\
g: \text{eval-lp}(b-ref)
\]

\[
is-string_u \\
\text{error:} \\
\text{call-cond}(\text{ERROR},846)
\]

\[
\text{T} \rightarrow \\
\text{call-io-cond}(u, \varnothing, \text{NAME}, \mu_0(<\text{datafield}: \text{lp-cl} \in \text{rp-cl}>),847)
\]

where: \( dt_{rp} = \text{elem}(2,\text{elem}(1,\text{datadir-rp-parsing}(\text{rp-cl}))) \)

Ref.: \text{call-io-cond} 10-30(86), \text{call-check-cond} 9-15(27), \\
\text{count-convert-assign} 10-68(192), \text{mk-io-data-op} 10-72(204), \\
\text{eval-lp} 7-4(5), \text{datadir-rp-parsing} 10-59(169), \text{call-cond} 9-11(18)

(220) \text{test-group-end}(u,cy,\text{idl-set},c) = 

\[
is-COMMA(c) \lor \text{is-BLANK}(c) \\
\text{get-data-lp}(u,cy,<>,\text{idl-set}); \\
\text{get-copy}(u,cy,F) \\
is-SEMIC(c) \\
\text{int-next-st}
\]

Ref.: \text{int-next-st} 5-16(24), \text{get-copy} 10-86(243)
10.5.3.5 Edit-directed transmission

This section defines transmission from a stream source in edit-directed mode. Transmission is governed by an evaluated control or data format whose width determines the number of source characters to be transmitted. The elementary actions of control formats are related with elementary transmission (see section 10.5.3.2).

Predicates with names starting with "is-io" which are not cited in the references following a formula, should be looked up in sections 10.5.2.1 and 10.5.2.2.

10.5.3.5.1 Control Formats

Abbreviations

fu-data_u = s-data(fu-elem_u)  
fu-ons_u = s-onsource(fu-elem_u)  
fu-col_u = s-column(fu-elem_u)  

metavariable

q  is-stream-data  v  is-*  a character value taken from a stream source or the indication to take one character value

(221)  get-space(u, cy, w) =

is-file_u  -- get-file-space(u, cy, w, *)

is-string_u  -- get-string-space(u, w, *)

(222)  get-string-space(u, w, q) =

w <= 0  -- PASS:q

\neg is-<>(fu-ons_u)  --

get-string-space(u, w - 1, in-q);

in-q: tk-onsource(u)

\neg is-<>(fu-data_u)  --

get-string-space(u, w - 1, q_{in});

upd-fu-elem(u, s-data, tail(fu-data_u))

cont'd
\[ T \rightarrow \]

\[
\text{error;}
\]

\[
\text{call-cond} (\text{ERROR, 848})
\]

where: \( q_{in} = \text{val-list} (\text{CHAR, head(fu-data}_u)) \)

Ref.: \text{tk-onsource 10-71(198), upd-fu-elem 10-6(13),
call-cond 9-11(18), val-list 8-8(27)}

\begin{equation}
(223) \text{get-file-space}(u, cy, w, q) =
\end{equation}

\[
= \text{is-FILEMARK}(q) \rightarrow
\]

\[
\text{call-io-cond}(u, \varnothing, \text{ENDFILE, } \varnothing, 849);
\]

\[
\text{upd-fu-elem}(u, s\text{-endfile}, T)
\]

\[
\text{is-LINE-DELIMITER}(q) \rightarrow
\]

\[
\text{get-file-space}(u, cy, w, in-q);
\]

\[
in-q; \text{tk-data-col}(u, cy, q)
\]

\[
w \leq 0 \rightarrow \text{PASS}\; q
\]

\[
is\text{-char-val}(q) \lor is-\ast(q) \rightarrow
\]

\[
\text{get-file-space}(u, cy, w - 1, in-q);
\]

\[
in-q; \text{tk-data-col}(u, cy, q)
\]

Ref.: \text{upd-fu-elem 10-6(13), call-io-cond 10-30(86), is-char-val 8-4(9)}

\begin{equation}
(224) \text{tk-data-col}(u, cy, q) =
\end{equation}

\[
= \text{is-LINE-DELIMITER}(q) \rightarrow
\]

\[
\text{tk-data}(u, cy);
\]

\[
\text{upd-fu-elem}(u, s\text{-column}, 1)
\]

\[
is\text{-char-val}(q) \lor is-\ast(q) \rightarrow
\]

\[
\text{tk-data}(u, cy);
\]

\[
\text{upd-fu-elem}(u, s\text{-column}, fu\text{-col}_u + 1)
\]

Ref.: \text{upd-fu-elem 10-6(13), tk-data 10-70(197), is-char-val 8-4(9)}
(225) \texttt{get-column}(u, cy, w, q) = \\
\hspace{1cm} w < 1 \rightarrow \texttt{get-column}(u, cy, 1, *) \\
\hspace{1cm} \texttt{is-}(q) \& \ w < \texttt{fu-col}_u \rightarrow \\
\hspace{1cm} \texttt{get-column}(u, cy, w, *) ; \\
\hspace{1.5cm} \texttt{get-skip}(u, cy, l, q) \\
\hspace{1.5cm} w = \texttt{fu-col}_u \rightarrow \texttt{null} \\
\hspace{1cm} \texttt{is-FILEMARK}(q) \lor \texttt{is-LINE-DELIMITER}(w) \rightarrow \\
\hspace{1.5cm} \texttt{get-skip}(u, cy, l, q) \\
\hspace{1.5cm} \texttt{is-char-val}(q) \lor \texttt{is-*}(q) \rightarrow \\
\hspace{2cm} \texttt{get-column}(u, cy, w, \texttt{in-q}) ; \\
\hspace{2cm} \texttt{in-q:tk-data-col}(u, cy, q) \\
\texttt{Ref.:} \texttt{is-char-val} 8-4(9) \\
\texttt{Note:} The linesize of a stream input file is varying and is deduced from the line-delimiters.

(226) \texttt{get-skip}(u, cy, w, q) = \\
\texttt{is-FILEMARK}(q) \rightarrow \\
\hspace{1cm} \texttt{call-io-cond}(u, \emptyset , \texttt{ENDFILE}, \emptyset , 850) ; \\
\hspace{1.5cm} \texttt{upd-fu-elem}(u, \texttt{s-endfile}, T) \\
\texttt{is-LINE-DELIMITER}(q) \& \ w > 1 \rightarrow \\
\hspace{1.5cm} \texttt{act-skip}(u, cy, w - 1, \texttt{in-q}) ; \\
\hspace{1.5cm} \texttt{in-q:tk-data-col}(u, cy, q) \\
\texttt{is-LINE-DELIMITER}(q) \rightarrow \texttt{upd-fu-elem}(u, s-column, 1) \\
\hspace{1cm} w \leq 0 \rightarrow \texttt{error} \\
\texttt{is-char-val}(q) \lor \texttt{is-*}(q) \rightarrow \\
\hspace{1.5cm} \texttt{get-skip}(u, cy, w, \texttt{in-q}) ; \\
\hspace{1.5cm} \texttt{in-q:tk-data-col}(u, cy, q) \\
\texttt{Ref.:} \texttt{upd-fu-elem} 10-6(13), \texttt{call-io-cond} 10-30(86), \texttt{is-char-val} \ 8-4(9)
10.5.3.5.2 Data formats

**Abbreviations**

- \( ft_e = s\text{-format-type}(edfo) \)
- \( pic_e = s\text{-pic}(edfo) \)
- \( efo_{re} = s\text{-real-part}(edfo) \)
- \( efo_{im} = s\text{-imag-part}(edfo) \)
- \( lg_{re} = \text{get-fo-lg}(efo_{re}) \)
- \( lg_{im} = \text{get-fo-lg}(efo_{im}) \)
- \( dt_{ac} = \text{elem}(2,dt) \)
- \( dt_c = \text{elem}(2,dt_{ac}) \)
- \( is\text{-point}_c = is\text{-POINT}\text{-elem}(2,dt_c) \)

**Cl**

- \( cl_1 = \text{LIST elem}(k,cl) \)
- \( cl_j = \text{LIST elem}(k,cl) \)
- \( bl_{cl} = \text{LIST single-char-bit\text{-elem}(i,cl)} \)

**Metavariables**

- \( pic \): \( is\text{-prop-pic} \)
- \( ft \): \( is\text{-FIX} \lor is\text{-FLT} \)
- \( d,p \): \( is\text{-intg-val} \)

Ref.: single-char-bit \( 8\text{-}35(135) \)

(227) \( \text{get-fo-lg}(edfo) = \)

\[
\begin{align*}
& is\text{-FIX}(ft_e) \lor is\text{-FLT}(ft_e) \lor \\
& (is\text{-BIT}(ft_e) \lor is\text{-CHAR}(ft_e)) \land is\text{-intg-val}(\nu_e) \\
& is\text{-PIC}(ft_e) \land is\text{-prop-pic}(pic_e) \land is\text{-CPLX\text{-s-mode}(pic_e)} \\
& \quad \text{pic-length}(pic_e) \\
& is\text{-CPLX}(ft_e) \land (is\text{-bin\text{-dec-pic}c\text{-s-pic}(efo_{re})} \lor is\text{-f-e\text{-s-format-type}(efo_{re})}) \land \\
& (is\text{-bin\text{-dec-pic}c\text{-s-pic}(efo_{im})} \lor is\text{-f-e\text{-s-format-type}(efo_{im})}) \\
& \quad lg\text{-cplx}_e \\
& T \quad \text{error}
\end{align*}
\]
where: \( lg\text{-cplx}_e = (lg_{re}=0 \quad \text{&} \quad lg_{im}=0 \rightarrow 0, \)
\( lg_{re}>0 \quad \text{&} \quad lg_{im}>0 \rightarrow lg_{re} + lg_{im}, \) \( T \rightarrow \Omega \)

\( w_e = s\text{-w(edfo)} \)
\( is\text{-bin-dec-pic}_c = is\text{-bin-pic} \quad \text{v} \quad is\text{-dec-pic} \)
\( is\text{-f-e}_c = is\text{-FIX} \quad \text{v} \quad is\text{-FLT} \)

Ref.: pic-length 8-64(241), is-prop-pic 8-47(187)

(228) \text{get-char-list}(u, cy, lg) =

\( is\text{-Q}(lg) \quad \text{v} \quad lg<0 \rightarrow \text{null} \)
\( lg=0 \rightarrow \text{PASS;}<> \)
\( T \rightarrow \)
\( \text{mk-list}(c, cl); \)
\( cl; \text{get-char-list}(u, cy, lg - 1); \)
\( c; \text{get-next-char}(u, cy) \)

Ref.: \text{get-next-char} 10-70(196)

Note: The character \( c \) will never be taken from s-onsource(fu-elem_u)

(229) \text{mk-editdir-op}(cl, edfo) =

\( is\text{-Q}(cl) \quad \text{v} \quad is\text{-<>}(cl) \rightarrow \text{PASS;} cl \)
\( is\text{-FIX}(ft_e) \quad \text{v} \quad is\text{-FLT}(ft_e) \rightarrow \)
\( \text{mk-op}(da, vr); \)
\( vr; \text{rep-real}(da, vr); \)
\( da; \text{pass-f-e-da}(ft_e, dt, d_e, P_e); \)
\( vr; \text{pass-f-e-val}(ft_e, dt, d_e, P_e); \)
\( dt; \text{verify-f-e}(ft_e, cl) \)
\( is\text{-BIT}(ft_e) \rightarrow \)
\( \text{pass-bit-op}(v); \)
\( v; \text{char-bit-conv}(da_b, cl_b) \)
\( is\text{-CHAR}(ft_e) \rightarrow \)
\( \text{pass-val-op}(char\text{-da-length}(cl), cl) \)
\( is\text{-PIC}(ft_e) \rightarrow \)
\( \text{mk-pic-op}(pic_e, cl-1); \)
\( cl-1; \text{verify-pic}(pic_e, cl) \)
\( is\text{-CPLX}(ft_e) \rightarrow \)
\( \text{mk-cplx-op}(op\text{-re}, op\text{-im}); \)
\( op\text{-im}; \text{mk-editdir-op}(cl_{im}, efo_{im}); \)
\( op\text{-re}; \text{mk-editdir-op}(cl_{re}, efo_{re}); \)

cont'd
where: $d_e = s-d(edfo)$

$$p_e = (is-FLT(ft) \lor is-Q\circ s-p(edfo) \rightarrow o,T \rightarrow s-p(edfo))$$

$$da_b = \mu (\langle s-baser\ BIT\rangle, \langle s-length;length(cl)\rangle)$$

$$cl_b = (is-BLANK-list(cl) \longrightarrow cl,T \longrightarrow \text{CONC } cl_1)$$

$$cl_l = (is-BLANK\cdot elem(i,cl) \longrightarrow \langle \rangle, T \longrightarrow \langle \text{elem}(i,cl) \rangle)$$

$$cl_{re} = \text{LIST} \cdot \text{elem}(i,cl)$$

$$cl_{lm} = \text{LIST} \cdot \text{elem}(i+l_{re},cl)$$

for: $is-Q(cl) \lor is-char-val-list(cl)$

Ref.: rep-real 8-9(31), mk-op 8-11(39), char-bit-conv 8-35(134), val-op 8-11(41), char-da 8-13(60)

Note: Expressions allowed by the abstract syntax of formats have been evaluated by eval-format. There is no test whether their values will actually be used.

(230) $\text{bit-op}(v) =$

$$\text{val-op}(\text{bit-d\ length}(v),v)$$

for: $is-bit-val-list(v)$

Ref.: val-op 8-11(41), bit-da 8-13(61)

(231) $\text{verify-f-e}(ft,cl) =$

$$\text{is-match}_{ft}(cl) \longrightarrow \text{PASS}; \text{parsing}_{ft}(cl)$$

$$T \longrightarrow$$

$$\text{verify-f-e}(ft,corr-cl);$$

$$\text{corr-cl:call-conv-cond}(cl,i_0,851)$$

where: $\text{is-match}_{ft} = (is-FIX(ft) \rightarrow is-f-data-corr,$

$$is-FLT(ft) \rightarrow is-e-data-corr)$$

$parsing_{ft} = (is-FIX(ft) \rightarrow f-data-parsing,$

$$is-FLT(ft) \rightarrow e-data-parsing)$$

$$i_0 = (i)(i > l \& \neg \text{is-leftmatch}_{ft}(cl_l) \&$$

$$(\forall j)(1 < j < i \Rightarrow \text{is-leftmatch}_{ft}(cl_j)))$$

$\text{is-leftmatch}_{ft} = (is-FIX(ft) \rightarrow is-f-data-leftcorr,$

$$is-FLT(ft) \rightarrow is-e-data-leftcorr)$$

Ref.: call-conv-cond 9-17(34), is-f-data-corr 10-60(171), is-e-data-leftcorr 10-60(172), is-e-data-corr 10-60(171)/is-f-data-leftcorr 10-60(172),

f-data-parsing 10-59(169), e-data-parsing 10-59(169),

10.5.3.5.2.
(232) is-f-data = is-BLANK-list \lor is-f-const-data

(233) is-f-const-data = (<elem(1):is-BLANK-list>,
                <elem(2):is-f-const>,
                <elem(3):is-BLANK-list>)

(234) is-f-const(dt) = is-io-arithm-const(dt) \& is-io-decimal-fixed elem(2,dt) \&
                      is-<>*elem(3,dt)

for: is-io-arithm-const(dt)

(235) is-e-data = (<elem(1):is-BLANK-list>,
                <elem(2):is-e-const>,
                <elem(3):is-BLANK-list>)

(236) is-e-const(dt) = is-io-arithm-const(dt) \& is-<>*elem(3,dt) \&
                      (is-io-decimal-fixed elem(2,dt) \lor is-io-decimal-float elem(2,dt))

for: is-io-arithm-const(dt)

(237) f-e-da(ft,dt,d,p) =
        is-FLT(ft) \longrightarrow re-da_c
        is-point_c \longrightarrow \mu(re-da_c, \langle s-scale-f, s-scale=f(re-da_c) - p \rangle)
        T \longrightarrow \mu(re-da_c, \langle s-scale-f, -d \rangle)

where: re-da_c = real-da(dt_{ac})
Ref.: real-da 10-73(205)

(238) f-e-val(ft,dt,d,p) =
        is-point_c \longrightarrow v_c.10 \uparrow p
        T \longrightarrow v_c.10 \uparrow (p - d)

where: v_c \in real-val(dt_{ac})

(239) verify-pic(pic,cl) =
        is-pic-match-l(pic,cl) \longrightarrow \text{PASS}:cl
        T \longrightarrow \text{verify-pic}(pic,\text{corr-cl});
        corr-cl: \text{call-conv-cond}(cl,i_o,852) \text{cont'd}

10.5.3.5.2
where: \( i_0 = (\forall l)(i \geq 1 \& \text{is-pic-match}_{\text{left}}(c_l)) \& \)
\((\forall j)(1 \leq j \leq i, \text{is-pic-match}_{\text{left}}(c_l))\)

\[\text{is-pic-match}_{\text{left}} = (\exists c_{l-1})(\text{is-pic-match}_{l-1}(\text{pic,c}l \cap c_{l-1})\]

Ref.: call-conv-cond 9-17(34)

(240) \text{is-pic-match}_{l}(\text{pic,c}l) =

\[\text{is-bin-p}ic(\text{pic}) \& \text{is-io-bit-char-list}(\text{cl}) \rightarrow \]
\[\text{is-pic-match}(\text{pic,b}l_{\text{cl}})\]

Ref.: is-pic-match 8-63(237)

(241) \text{mk-pic-op}(\text{pic,c}l) =

\[\text{mk-op}(\text{pic,rep-list}(\text{str-base}(\text{pic}), v_1))\]

where: \( v_1 = (\text{is-bin-p}ic(\text{pic}) \rightarrow \text{bl}_{\text{cl}}, \text{T} \rightarrow \text{cl})\)

Ref.: rep-list 8-10(34), mk-op 8-11(39), str-base 8-39(156)

(242) \text{mk-cpl}x-op(\text{op-re,op-im}) =

\[\text{mk-op}(\text{da,v}r, )\]
\[\text{da:pass-new(CPLX, pref(s-da(op-re), s-da(op-im))))},\]
\[\text{vr:pass-cpl}x-vr(s-vr(op-re), s-vr(op-im))\]

for: \(\text{is-op}(\text{op-re}) \& \text{is-op}(\text{op-im})\)

Ref.: new 11-30(41), pref 11-32(45), cplx-vr 8-9(30), mk-op 8-11(39)

10.5.3.6 Copy action

The instruction get-copy is executed whenever a single character is taken from a dataset by the instruction tk-data. If this character has no special meaning and if the copy option is specified in the get statement under execution, this character is copied on the standard system print file.
(243) \texttt{get-copy}(u, cy, q) =
\begin{align*}
is-\Omega(q) &\rightarrow \text{error} \\
is-\Omega(cy) \lor is-LINE-DELIMITER(q) \lor is-FILEMARK(q) &\rightarrow \text{null} \\
is-char-val(q) &\rightarrow \\
\text{put-next-charl}(u_p, q); \\
qu &\rightarrow \text{int-stand-print-open} \\
T &\rightarrow \text{def-copy}(u_p)
\end{align*}

where: \(u_p = s-fuos-stand-print(FD)\)

for: is-stream-data(q) \lor is-\Omega(q) \lor is-T(q) \lor is-F(q)

Ref.: \texttt{put-next-charl} 10-96(261), \texttt{int-stand-print-open} 10-16(38),
\texttt{is-char-val} 8-4(9)

(244) \texttt{def-copy}(u) =

Note: This instruction is implementation defined. It can cause additional
actions on the standard system print file.

\[10.5.3.6\]
10.5.4 Put Statement

This section contains the definition of the put statement which names a file (is-file-put) or a string (is-string-put) and the definition of the standard system action of the check on-condition.

Abbreviations

fu-elem\textsubscript{u} = (is-fu-elem\textsubscript{u}(FU) \rightarrow u(FU), T \rightarrow error)  
Ref.: is-fu-elem 2-35(113)

is-string\textsubscript{u} = is-fu-string\textsubscript{u}(FU)  
Ref.: is-fu-string 2-35(114)

is-file\textsubscript{u} = is-fu-file\textsubscript{u}(FU)  
Ref.: is-fu-file 2-36(117)

fu-data\textsubscript{u} = s-data(fu-elem\textsubscript{u})

is-print\textsubscript{u} = is-file\textsubscript{u} & PRINT & s-csa(fu-elem\textsubscript{u})

Metavariabes

t \quad is-put-st the text of a put statement
u \quad is-n a file union name
q \quad is-gen \quad is-ps-gen a generation or pseudo-generation of the target string variable

fo-i \quad is-intg-val \quad is-Q an index which, together with a file union name, selects the appropriate format list in FU

tr \quad is-tr a characterization of the transmission mode

itl \quad is-item-list a list of items of a data list

tm \quad is-expr a text of an expression (including a reference) which is modified in order to yield the scalar parts

eda \quad is-eda \quad is-Q the evaluated data attributes of an expression (including a reference) which govern modification of tm or Q

op \quad is-op \quad is-Q an operand which has special meaning if it is Q

efo \quad an evaluated data or control format or Q
10.5.4.1 Control oriented part

(245) \texttt{int-put-st(t) =}
\begin{itemize}
  \item \texttt{is-file-put(t) \rightarrow} \texttt{put-data-spec(t,u,Q)};
  \item \texttt{init-file-3(u,s-page(t),line,skip),}
  \item \texttt{line:eval-intg-expr(s-line(t),E),}
  \item \texttt{skip:eval-intg-expr(s-skip(t),E),}
  \item \texttt{upd-fu-elem(u,s-count,bintg-op(0));}
  \item \texttt{u:int-impl-open(t)}
\end{itemize}

\texttt{is-string-put(t) \rightarrow}
\begin{itemize}
  \item \texttt{call-check-cond(<s-string(t)>,E,853);}
  \item \texttt{convert-assign(g,op);}
  \item \texttt{optk-str-data(u);}\texttt{put-data-spec(t,u,Q);}\texttt{init-string-2(s-spec(t),u,g);}\texttt{u:un-name,}
  \item \texttt{g:eval-lp(s-string(t))}
\end{itemize}

Ref.: \texttt{int-impl-open 10-13(32), bintg-op 8-11(43),}
\texttt{upd-fu-elem 10-6(13), eval-intg-expr 7-21(52),}
\texttt{eval-lp 7-4(5), un-name 2-17(36), convert-assign 7-9(19),}
\texttt{call-check-cond 9-15(27)}

(246) \texttt{init-file-3 (u,page,line,skip) =}
\begin{itemize}
  \item \texttt{is-T(page) \rightarrow} \texttt{init-file-3(u,Q,line,Q)};
  \item \texttt{put-page(u)}
\end{itemize}
\begin{itemize}
  \item \texttt{is-intg-val(line) \rightarrow} \texttt{put-line(u,w,Q)}
\end{itemize}
\begin{itemize}
  \item \texttt{is-intg-val(skip) \& is-printu \& skipsO \rightarrow} \texttt{upd-data-col(u,1,CARR-RETURN)}
\end{itemize}

cont'd
is-intg-val(skip) & skip := 1 →
    put-skip(u,skip,\emptyset)

is-intg-val(skip) → error
T → null

where: \( w_1 = (\text{line}:0 \rightarrow 1, \ T \rightarrow \text{line}) \)

for: is-opt(page) & (is-intg-val(line) v is-Q(line)) &
    (is-intg-val(skip) v is-Q(skip))

Ref.: put-page 10-101(278), put-line 10-101(277),
    upd-data-col 10-97(263), put-skip 10-100(274)

Note: A consistency check for the options is made in verify-impl-open(t).

(247) init-string-2(spec,u,g) =
    is-CHAR*gen-base(g) & ~is-<>(spec) →
    s-fu:u:(FU;u;\emptyset\langle<s-data>>,\langle<s-gen>\rangle)\rangle
T → error

for: is-data-spec-list(spec)
Ref.: gen-base 10-21(51)

(248) tk-str-data(u) =
    mk-op(char-da*length(fu-data_u),fu-data_u)
Ref.: mk-op 8-11(39), char-da 8-13(60)

(249) put-data-spec(t,u,fo-i) =
    is-<>s-spec(t) → null
    is-edit-directed(spec-l_t) &
    (is-Q(spec-2_t) v is-edit-directed(spec-2_t)) &
    ~is-<>(itl_t) & ~is-<>(fcl_t) →
    put-data-spec(t-tail_t,u,fo-i_tr);
    put(u,e_tr,\langle itl_t \rangle)
    is-data-directed(spec-l_t) v is-ref(spec-l_t) →
    put-next-charl(u,\langle SEMIC \rangle);
    put(u,tl_o,\langle itl_o \rangle)

cont'd
is-list-directed(spec-1_t) & ~is-<>(itl_t) ➞ put(u,t,tr,\itl_t)

T ➞ error

where:

spec-1_t = head\ast s-spec(t)
spec-2_t = elem(1, tail\ast s-spec(t))
t-tail_t = \mu(t;\langle s-spec\ast tail\ast s-spec(t) \rangle)
itl_t = s-data-list(spec-1_t)
fol_t = s-format-list(spec-1_t)
fo-i_tr = (is-intg-val(fo-i) ➞ fo-i,
is-file_u & ~is-\varnothing\ast s-format-list(fu-elem_u) ➞ length\ast s-format-list(fu-elem_u) + 1,
T ➞ 1)
e_tr = \mu_o(\langle s-type:fol_t,\langle s-fo-i:fo-i_tr \rangle \rangle)
l_tr = \mu_o(\langle s-type:<> \rangle)
tr_o = (is-ref(spec-1_t) ➞ CHECK, T ➞ \varnothing)
itl_o = (is-ref(spec-1_t) ➞ <ref>,
is-<>\ast s-d-data-list(spec-1_t) ➞ mk-datadir-ref-list(order-set\ast datadir-n-set(E,AT),<>),
T ➞ s-d-data-list(spec-1_t))

for:
is-put-st(t) \lor is-ref\ast head\ast s-spec(t)

Ref.: put-next-char 10-96(261), datadir-n-set 10-67(187)

(250) order-set(n-set) =

for: is-n-set(n-set)

Note: This function arranges the names which are the members of its argument n-set into a list, in an unspecified way. The definition of order-set could be done in a similar way as described with the function permute.

(251) mk-datadir-ref-list(n-list,ref-list) =

is-<>(n-list) ➞ ref-list

T ➞ mk-datadir-ref-list(tail(n-list), ref-list \cap mk-elim-ref-list
(head(n-list), expand-idl(<id_n>,da_n)))

where:
id_n = (\exists idl)(\exists idl)(id=\text{head}(idl) & sel(idl)(\varnothing) = n_e)
da_n = s-d\ast s-attr\ast n_e(\varnothing)

for: is-n-list(n-list) & is-basic-ref-list(ref-list)

Ref.: expand-idl 9-16(32)
(252) \text{mk-elim-ref-list}(n, e-idl-list) =

\text{length}(e-idl-list) \to \text{CONC} \to \text{<ref1>}

where: \text{ref1} = (\text{sel}(\text{idl1}_1)(E) \neq n \to \text{<>},
\quad T \to \mu \text{O(}<\text{idl-list:idl1}_1>,<\text{arg-list:<>}>))
\quad \text{idl1}_1 = \text{elem}(i, e-idl-list)

for: \text{is-n}(n) \& \text{is-id-list-list}(e-idl-list)

(253) \text{put}(u, tr, itl) =

\text{is-<>}(itol) \to \text{null}

\text{is-ref}(itol_h) \& \text{is-data-tr}(tr) \& \text{idl}_h \in \text{datadir-idl-set}(E, AT) \to

\text{is-CHECK}(tr) \to

\text{put}(u, O, itl_t);
\quad \text{count-iterate}(u, O, itl_h, edar_h)

\text{is-expr}(itol_h) \to

\text{put}(u, tr, itl_t);
\quad \text{count-iterate}(u, tr, itl_h, eda)
\quad \text{eda:eda-expr}(itol_h, E)

\text{is-contr-item}(itol_h) \to

\text{put}(u, tr, itl_t);
\quad \text{int-contr-item}(\text{PUT}, u, tr, itl_h)

T \to \text{error}

where: itl_h = \text{head}(itol)
\quad itl_t = \text{tail}(itol)
\quad idl_h = \text{s-id-list}(itol_h)
\quad edar_h = (\text{is-LABEL}(attr_h) \vee \text{is-entry}(attr_h) \to \text{Q},
\quad \text{is-prop-var}(attr_h) \to \text{sub-da}(eda_h, s_l_h, eql_h))
\quad eda_h = \text{s-da-head}(n_h(DN))(AC)
\quad s_l_h = \text{s-arg-list}(itol_h)
\quad eql_h = \text{eval-ql}(\text{tail}\cdot \text{s-id-list}(itol_h), s\cdot \text{da}(attr_h))
\quad attr_h = \text{s-attr\cdot n_h(AT)}
\quad n_h = \text{sel}\cdot \text{s-id-list}(itol_h)(E)

for: \text{is-tr}(tr) \vee \text{is-CHECK}(tr)

\text{Ref.} \quad \text{datadir-idl-set} 10-67(186), \text{eda-expr} 7-22(54), \text{sub-da} 7-34(89),
\text{int-contr-item} 10-52(131), \text{eval-ql} 7-27(66), \text{is-data-tr} 10-52/135

10.5.4.1
(254) \text{count-iterate}(u, tr, tm, eda) = \\
\text{is-file}_u \& (\text{is-scalar}(eda) \lor \text{is-}Q(eda)) \quad \rightarrow \\
\text{verify-transmit}(u, 854); \\
\text{put-l}(u, tr, tm, eda); \\
\text{upd-fu-elem}(u, \text{s-count}, \text{bintg-op}); \\
bintg-op; \text{add-op-expr}(\text{fu-count}_u, \text{bintg-op}(1)) \\
is\text{-string}_u \& \text{is-scalar}(eda) \quad \rightarrow \quad \text{put-l}(u, tr, tm, eda) \\
T \quad \rightarrow \quad \text{iterate-put}(u, tr, tm, eda, \text{lb}(eda))

\text{where: } \text{fu-count}_u = \text{s-count}((\text{fu-elem}_u))

\text{Ref.: add-op-expr 10-101(276), upd-fu-elem 10-6(13),} \quad \text{verify-transmit 10-7(19), bintg-op 8-11(43)}

(255) \text{iterate-put}(u, tr, tm, eda, i) = \\
i > \text{ub}(eda) \quad \rightarrow \quad \text{null} \\
T \quad \rightarrow \\
\text{iterate-put}(u, tr, tm, eda, i + 1); \\
\text{count-iterate}(u, tr, tm - 1, \text{da-part}(eda, i)); \\
tm - 1; \text{mod}(tm, eda, i, E)

\text{for: is-intg-val}(i) \\
\text{Ref.: mod 7-7(13)}

(256) \text{put-l}(u, tr, tm, eda) = \\
\text{is-}Q(eda) \quad \rightarrow \quad \text{put-data-dir}(u, idl_t, \langle \rangle, \langle \rangle) \\
\text{is-data-tr}(tr) \quad \rightarrow \\
\text{put-data-dir}(u, idl_t, \langle \rangle, \langle \rangle, \langle \rangle); \\
isl; \text{pass-eval-sl}(da_t, rl); \\
op-l; \text{convert}(da, op); \\
da; \text{pass-s-da}(gen); \\
op; \text{pass-gen-op}(gen, \langle \rangle); \\
gen; \text{eval-sub-gen}(tm, rl, E); \\
rl; \text{mk-rl}(eda_t, sl_t, eql_t, E)

T \quad \rightarrow \\
\text{put-list-dir-edit-dir}(u, tr, op - 1); \\
op-l; \text{convert}(da, op); \\
da; \text{pass-s-da}(op); \\
op; \text{eval-expr}(tm, E)

\text{cont'd}
where: idl_\_t = s-id-list(tm)
sl_\_t = s-arg-list(tm)
eql_\_t = eval-ql(tail(idl_\_t),da_\_t)
da_\_t = s-da\_s-attr(sel(idl_\_t)(E)(AT))
ed_\_a = s-da\_head(sel(idl_\_t)(E)(DH)(AG))

Ref.: put-data-dir 10-98(267), mk-rl 7-26(62), eval-sub-gen 7-25(60),
gen-op 7-19(47), convert 8-30(118), eval-expr 7-18(43),
eval-ql 7-27(66), is-data-tr 10-52(135)

(257) eval-sl(da,rl) =

\[
\text{is-array(da)} \\ \text{\text{<head(rl)\textsuperscript{>}}} \\ \text{eval-sl(s-elem(da),tail(rl))}
\]
\[
\text{is-struct(da) } \lor \text{ is-cell(da)} \\ \text{eval-sl(s-da\_e\_elem(head(rl),da),tail(rl))}
\]
\[
\text{is-scalar(da)} \\ <>
\]

for: is-da(da) & is-intg-val-list(rl)

(258) put-listdir-editdir(u,tr,op) =

\[
\text{is-list-tr(tr)} \\ \text{\text{put-listdir(u,op)}}
\]
\[
\text{T} \\ \text{put-editdir(u, tr,op,edfo);} \\ \text{edfo:put-format(u,tr,\text{\O} )}
\]

Ref.: is-list-tr 10-52(136), put-listdir 10-97(264),
put-editdir 10-102(280)

(259) put-format (u,tr,efo) =

\[
\text{is-\O (efo)} \\ \text{put-format(u,tr,efo-1);} \\ \text{efo-1:eval-format(u,s-type(tr),s-fo-i(tr))}
\]
\[
\text{is-SPACE(\text{ft}_e) } \& \text{ is-intg-val(\text{w}_e)} \\ \text{put-format(u,tr,\text{\O} )}; \\ \text{put-space(u,\text{\text{\O} })}
\]

cont'd
is-COLUMN($ft_e$) & is-intg-VAL($w_e$) & is-file$_u$ 

    put-format($u$, $tr$, $\emptyset$);
    put-column($u$, $w_e$)

is-SKIP($ft_e$) & is-intg-VAL($w_e$) & is-print$_u$ & $w_e < 0$ 

    put-format($u$, $tr$, $\emptyset$);
    upd-data-col($u$, 1, CARR-RETURN)

is-SKIP($ft_e$) & is-intg-VAL($w_e$) & is-file$_u$ & $w_e > 0$ 

    put-format($u$, $tr$, $\emptyset$);
    put-skip($u$, $w_e$, $\emptyset$)

is-LINE($ft_e$) & is-intg-VAL($w_e$) & is-print$_u$ 

    put-format($u$, $tr$, $\emptyset$);
    put-line($u$, $w_1$, $\emptyset$)

is-PAGE($ft_e$) & is-print$_u$ 

    put-format($u$, $tr$, $\emptyset$);
    put-page($u$)

$ft_e \in \{\text{FIX, FLT, CPLX, PIC, BIT, CHAR}\}$ 

    PASS: efo

T $\rightarrow$ error

where: $ft_e$ = s-format-type(efo)

    $w_e$ = s-w(efo)

    $w_1 = (w_e < 0 \rightarrow 1, T \rightarrow w_e)$

Ref.: eval-format 10-53(138), put-space 10-100(272), put-column 10-100(273),

    upd-data-col 10-97(263), put-skip 10-100(274), put-line 10-101(277),

    put-page 10-101(278)
10.5.4.2 Elementary transmission

Elementary transmission to a stream target, i.e. output by a file or transmission to a string is the transmission of a list of character values. This section is related with section 10.5.4.5.1 (Control Formats).

Abbreviations

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>fu-lsz&lt;sub&gt;u&lt;/sub&gt;</td>
<td>s-linese(fu-elem&lt;sub&gt;u&lt;/sub&gt;)</td>
</tr>
<tr>
<td>fu-col&lt;sub&gt;u&lt;/sub&gt;</td>
<td>s-column(fu-elem&lt;sub&gt;u&lt;/sub&gt;)</td>
</tr>
<tr>
<td>fu-gen&lt;sub&gt;u&lt;/sub&gt;</td>
<td>s-gen(fu-elem&lt;sub&gt;u&lt;/sub&gt;)</td>
</tr>
<tr>
<td>fu-pos&lt;sub&gt;u&lt;/sub&gt;</td>
<td>s-position(fu-elem&lt;sub&gt;u&lt;/sub&gt;)</td>
</tr>
</tbody>
</table>

Metavariabes

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cl</td>
<td>is-char-val-list a list of character values</td>
</tr>
</tbody>
</table>

(260) \text{put-next-tab}(u,eda,cl) =

\text{is-print}_u \rightarrow \text{put-next-charl}(u,cl)

\text{is-num-type}(eda) \& \text{is-intg-val}(\text{next-tab}_u) \& 
\text{next-tab}_u + \text{length}(cl) \leq \text{fu-lsz}_u \\
\neg \text{is-num-type}(eda) \& \text{is-intg-val}(\text{next-tab}_u) \& \text{next-tab}_u \leq \text{fu-lsz}_u \\
\rightarrow \text{put-next-charl}(u,cl);

\text{upd-data-col}(u,\text{next-tab}_u,\text{TABULATOR})

T \rightarrow 

\text{put-next-tab}(u,eda,cl);

(\text{put-skip}(u,1,\Omega))

where: \text{next-tab}_u = (\neg (\exists i) (\text{is-ge-tab}_i) \rightarrow \Omega,

T \rightarrow (ti)(\text{is-ge-tab}_i \& \neg (\exists j)(\text{is-ge-tab}_j \& j<i)))

\text{is-ge-tab}_i = i \in \text{TAB-SET} \& \text{fu-col}_u^i \\
\text{is-ge-tab}_j = j \in \text{TAB-SET} \& \text{fu-col}_u^j \\

for: \text{is-scalar-eda}(eda) \lor \text{is-\Omega}(eda)

Ref.: \text{is-num-type} 8-23(92), \text{put-skip} 10-100(274)

Note: This instruction is executed in list-or data-directed transmission only. In the latter case also for data attributes which are neither of numeric nor of string type. The elementary object TAB-SET is an implementation defined set of integer values (is-intg-val-set(TAB-SET)).
(261) \( \text{put-next-charl}(u, cl) = \)

\[
\text{is-<>}(cl) \rightarrow \text{null} \\
\text{is-file}_u \& \text{fu-col}_u \leq \text{fu-lsz}_u \rightarrow \text{put-next-charl}(u, \text{tail}(cl)); \\
\text{upd-data-col}(u, \text{fu-col}_u + 1, \text{head}(cl)) \\
\text{is-file}_u \rightarrow \\
\text{put-next-charl}(u, cl); \\
\text{put-skip}(u, 1, \varnothing) \\
\text{is-string}_u \& \text{gen-length}(\text{fu-gen}_u) > \text{length}(\text{fu-data}_u) \rightarrow \\
\text{put-next-charl}(u, \text{tail}(cl)); \\
\text{upd-fu-elem}(u, s-data, \text{fu-data}_u \ominus \text{rep(CHAR, head}(cl))) \\
\text{T} \rightarrow \\
\text{error}; \\
\text{call-cond}(\text{ERROR}, 855)
\]

Ref.: \text{put-skip} 10-100(274), \text{upd-fu-elem} 10-6(13), \text{rep} 8-7(22), \text{call-cond} 9-11(18), \text{gen-length} 10-96(262)

Note: This instruction performs data transmission to a stream target with some analogy to the instruction \text{get-next-char} which transmits a single datum from a stream source, and with some analogy to the instruction \text{get-char-list}.

(262) \text{gen-length}(g) =

\[
\text{is-string-type}\&_s\text{-da}(g) \rightarrow \text{str-length}\&_s\text{-da}(g) \\
\text{s-id}(g) = \text{mk-id}(\text{SUBSTR}) \rightarrow \text{gen-length}\&_\text{elem}(1, \text{s-arg-list}(g)) \\
\text{s-id}(g) = \text{mk-id}(\text{ONSOURCE}) \rightarrow \text{MAX-LENGTH} \\
\text{s-id}(g) = \text{mk-id}(\text{ONCHAR}) \rightarrow 1 \\
\text{T} \rightarrow \varnothing
\]

Ref.: \text{is-string-type} 8-23(93), \text{str-length} 8-40(157), \text{mk-id} 10-76(214), \text{MAX-LENGTH} 9-17(36)

Note: This function is used to find the maximum length of a character string type generation or pseudo-generation.
(263) \texttt{upd-data-col}(u,col,q) = \\
\quad \texttt{upd-dataset}(u,Q,ids); \\
\quad \texttt{elem}(fu-pos_u)(ids);\texttt{pass}(q); \\
\quad ids: \texttt{tk-dataset}(u,Q); \\
\quad \texttt{upd-fu-elem}(u,s-position,fu-pos_u+1), \\
\quad \texttt{upd-fu-elem}(u,s-column,col) \\
\quad \text{for: is-intg-val(col) & is-print-data(q) } \\
\text{Ref.: upd-fu-elem 10-6(13), tk-dataset 10-7(16),} \\
\text{upd-dataset 10-6(11), is-print-data 2-40(129)} \\

10.5.4.3 List-directed transmission \\

This section defines the transformation of an operand into a list of 
character values.

(264) \texttt{put-listdir}(u,op) = \\
\quad \texttt{put-next-tab}(u,eda_0,cl); \\
\quad cl: \texttt{pass-mod-op-string}(\neg\texttt{is-print_u},eda_0,cl-l); \\
\quad cl-l: \texttt{mk-num-string}(*,op) \\
\quad \text{where: eda}_0 = s-da(op)  \\
\text{Ref.: put-next-tab 10-95(260)} \\

(265) \texttt{mk-num-string}(tr,op) = \\
\quad \texttt{is-dec-pic}(eda_0) \lor \texttt{is-sterling-pic}(eda_0) \lor \\
\quad \texttt{is-string-type}(eda_0) \land \neg\texttt{is-BIT's-base}(eda_0) \\
\quad \texttt{PASS:val-list}(\texttt{CHAR,vr}_o) \\
\quad \texttt{is-bin-pic}(eda_0) \lor \texttt{is-BIT's-base}(eda_0) \lor \texttt{is-arithm}(eda_0) \lor \texttt{is-\ast}(tr) \\
\quad \texttt{convert}(\texttt{CHAR-DA,op}) \\
\quad \texttt{is-\neg}(tr) \longrightarrow \texttt{null} \\
\quad \text{where: eda}_0 = s-da(op) \\
\quad \quad vr_o = s-vr(op) \\
\text{Ref.: is-string-type 8-23(93), convert 8-30(118), CHAR-DA 8-13(58)} \\
\text{val-list 8-8(27)}
10.5.4.4 Data-directed transmission

This section defines the transformation of a list of identifiers, of a list of integer subscripts and of an operand into a list of character values.

Metavariabes

\( \text{idl} \) \( \text{idl-list} \) a list of identifiers
\( \text{intgl} \) \( \text{intgl-val-list} \) a list of integer subscripts

(267) \( \text{put-datadir}(u,\text{idl},\text{intgl},\text{op}) = \)

\( \text{put-next-tab}(u, \text{eda}_0, \text{cl}); \)
\( \text{cl-pass-mod-string}(\text{idl}, \text{intgl}, \text{eda}_0, \text{cl}-1); \)
\( \text{cl-1.mk-num-string}(\text{op}) \)

where: \( \text{eda}_0 = s-da(\text{op}) \)
Ref.: \( \text{mk-num-string} \) 10-97(265), \( \text{put-next-tab} \) 10-95(260)

(268) \( \text{mod-string}(\text{idl}, \text{intgl}, \text{eda}, \text{cl}) = \)

\( \text{is-}Q(\text{cl}) \)

\( \text{idl-string}(\text{idl}) \text{^ intgl-string}(\text{intgl}) \)
\( \text{is-char-val-list}(\text{cl}) \)

\( \text{idl-string}(\text{idl}) \text{^ intgl-string}(\text{intgl}) \text{^ <B0> mod-op-string}(T, \text{eda}, \text{cl}) \)

Ref.: \( \text{mod-op-string} \) 10-98(266), \( \text{is-char-val} \) 8-4(9)
(269) idl-string(idl) =
\[ \text{length(idl)} \]
\[ \text{CONC} \left( \text{point}_i \cap \text{chl(idl)} \right) \]
where: \( idl_i = \text{elem}(i, idl) \)
\( \text{point}_i = (i=1 \rightarrow \text{< >}, i>1 \rightarrow \text{<POINT>}) \)
Ref.: chl 10-76(215)

(270) intgl-string(intgl) =
\[ \text{is-<}(\text{intgl}) \rightarrow \text{<>} \]
\[ \text{T} \rightarrow (\text{length(intgl)} \rightarrow \text{CONC} \left( \text{del}_i \cap \text{signed-intg}_i \right)) \rightarrow \text{<RIGHT-PAR>} \]
where: \( \text{del}_i = (i=1 \rightarrow \text{<LEFT-PAR>}, i>1 \rightarrow \text{<COMMA>}) \)
\( \text{signed-intg}_i = (v_i=0 \rightarrow \text{<O-CHAR>}, v_i<0 \rightarrow \text{<MINUS> \cap digit-list(abs(v_i))}, v_i>0 \rightarrow \text{digit-list(v_i)}) \)
\( v_i = \text{elem}(i, \text{intgl}) \)

(271) digit-list(i) =
\[ i=0 \rightarrow \text{<>} \]
\[ i>0 \rightarrow \text{digit-list \cdot trunc(i/10) \cap \text{<single-num-char \cdot modulo(i,10)>}} \]
for: \( \text{is-intg-val(i)} \& i\geq0 \)
Ref.: single-num-char 8-33(128)

10.5.4.5 Edit-directed Transmission

This section defines transmission to a stream target in edit-directed mode. The definition of control formats is related with section 10.5.4.2 (Elementary Transmission).

10.5.4.5.1 Control Formats

Abbreviations

\[ \text{fu-lsz}_u = s\text{-linesize}(\text{fu-elem}_u) \]
\[ \text{fu-col}_u = s\text{-column}(\text{fu-elem}_u) \]
\[ \text{fu-line}_u = s\text{-line}(\text{fu-elem}_u) \]
\[ \text{fu-psz}_u = s\text{-pagesize}(\text{fu-elem}_u) \]
Metavariables

\[ w \quad \text{is-intg-val} \quad \text{an integer used for spacing, column, skip and line formats} \]

\[ \text{opt} \quad \text{is-opt} \quad \text{an indication whether or not endpage on-condition has been raised as part of skip or line format} \]

\[ \text{op} \quad \text{is-bintg-op} \quad \text{a binary integer operand with default precision used for incrementing the values of count and line number} \]

(272) \[ \text{put-space}(u,w) = \]

\[ \text{put-next-charl}(u, \text{LST} \text{ BLANK}) \]

Ref.: \text{put-next-charl} 10-96(261)

Note: Spacing is a special case of data transmission to a stream target; this is in contrast with data transmission from a stream source which is a special case of spacing.

(273) \[ \text{put-column}(u,w) = \]

\[ \neg (1 \leq w \leq f-1sz_u) \rightarrow \text{put-column}(u,1) \]
\[ w \geq f-col_u \rightarrow \text{put-space}(u,w - f-col_u) \]
\[ w < f-col_u \rightarrow \]
\[ \text{put-column}(u,w); \]
\[ \text{put-skip}(u,1,\emptyset) \]

(274) \[ \text{put-skip}(u,w,\text{opt}) = \]

\[ \text{is-T(opt) \lor w=0} \rightarrow \text{null} \]

\[ T \rightarrow \]
\[ \text{put-skip}(u,w - 1,\text{opt}-1); \]
\[ \text{opt-1:start-line}(u) \]

10.5.4.5.1
(275) \text{start-line}(u) = \\
\quad \text{is-print}_u \& \text{fu-psz}_u = \text{op-val}(\text{fu-line}_u) \rightarrow \\
\quad \text{pass}(T); \\
\quad \text{call-io-cond}(u, \Omega, \text{ENDPAGE}, \Omega, 856); \\
\quad \text{upd-fu-elem}(u, s-column, 1), \\
\quad \text{upd-fu-elem}(u, s-line, \text{bintg-op}); \\
\quad \text{bintg-op}: \text{add-op-expr}(\text{fu-line}_u, \text{bintg-op}(l)) \\
\quad T \rightarrow \text{upd-data-col}(u, 1, \text{LINE-DELIMITER}); \\
\quad \text{upd-fu-elem}(u, s-line, \text{bintg-op}); \\
\quad \text{bintg-op}: \text{add-op-expr}(\text{fu-line}_u, \text{bintg-op}(l)) \\
\text{Ref.:} \text{op-val} 8-11(40), \text{call-io-cond} 10-30(86), \text{upd-fu-elem} 10-6(13), \\
\text{upd-data-col} 10-97(263), \text{bintg-op} 8-11(43) \\
\text{Note:} The instruction returns the value \Omega if the endpage condition has not 
been called.

(276) \text{add-op-expr}(op-1, op-2) = \\
\quad \text{convert}(s-da(op-1), op); \\
\quad \text{op: eval-infix-expr}(op-1, op-2, \text{ADD}) \\
\text{Ref.:} \text{eval-infix-expr} 8-14(65), \text{convert} 8-30(118)

(277) \text{put-line}(u, w, \text{opt}) = \\
\quad \text{is-T}(\text{opt}) \lor (w = \text{op-val}(\text{fu-line}_u) \& \text{fu-col}_u = 1) \rightarrow \text{null} \\
\quad \text{fu-psz}_u < \text{op-val}(\text{fu-line}_u) \& w < \text{op-val}(\text{fu-line}_u) \rightarrow \text{put-page}(u) \\
\quad T \rightarrow \\
\quad \text{put-line}(u, w, \text{opt}-1); \\
\quad \text{opt}-1: \text{start-line}(u) \\
\text{Ref.:} \text{op-val} 8-11(40)

(278) \text{put-page}(u) = \\
\quad \text{upd-data-col}(u, 1, \text{PAGE-DELIMITER}); \\
\quad \text{upd-fu-elem}(u, s-line, \text{bintg-op}(l)) \\
\text{Ref.:} \text{upd-fu-elem} 10-6(13), \text{upd-data-col} 10-97(263), \text{bintg-op} 8-11(43)
(279) \( \text{syst-endpage-exec}(f) = \)

\[
\text{put-page}(u_f)
\]

where: \( u_f = (\text{is-n-s-fu}\cdot f(FD) \rightarrow s-fu\cdot f(FD)) \),

\( \text{is-STAND-PRINT}\cdot s-fu\cdot f(FD) \rightarrow s-fu\cdot s-\text{stand-print}(FD) \)

10.5.4.5.2 Data formats

Abbreviations

\( \text{ft}_e = \text{s-format-type}(\text{edfo}) \)

\( \text{pic}_e = \text{s-pic}(\text{edfo}) \)

\( \text{we} = \text{s-w}(\text{edfo}) \)

\( \text{lg}_{dl} = \text{length}(\text{dl}) \)

\( \text{lg}_{cl} = \text{length}(\text{cl}) \)

Metavariables

\( \text{edfo} \) an evaluated data format

\( \text{sign} \) a sign of a data field

\( \text{dl} \) a non-edited list of digits including the empty list

\( d,s \) the width of a fractional part or the number of significant digits

(280) \( \text{put-editdir}(u,\text{tr},\text{op},\text{edfo}) = \)

\[
\text{is-prop-}\text{we} \& (\text{is-REAL}(\text{tr}) \vee \text{is-CPLX}(\text{tr}) \vee \\text{is-CPLX}(\text{ft}_e)) \rightarrow
\]

\[
\text{put-next-charl}(u,\text{cl});
\]

\[
\text{cl:verify-width}(\text{cl}-1,\text{edfo});
\]

\[
\text{cl}-1:\text{pass-edit-op}(\text{op}-1,\text{edfo});
\]

\[
\text{op}-1:\text{convert}(d_{ae},\text{op})
\]

\[
\text{is-CPLX}(\text{ft}_e) \rightarrow
\]

\[
\text{put-editdir}(u,\text{CPLX},\text{op},\text{efo}_{im});
\]

\[
\text{put-editdir}(u,\text{REAL},\text{op},\text{efo}_{re})
\]

\[
\text{T} \rightarrow
\]

\[
\text{put-editdir}(u,\text{tr},\text{op},\text{edfo}-1);
\]

\[
\text{edfo}-1:\text{put-format}(u,\text{tr},Q)
\]

cont'd
where: is-prop-\(w_e\) = (is-CHAR(\(f_t\)) \(\rightarrow\) is-\(Q(w_e) \cup w_e > 0\),
\[ T \rightarrow \neg is-Q(\lg_e) \& \lg_e > 0 \]

\[ \lg_e = \text{get-fo-lg(}edfo) \]
\[ da_e = (is-FIX(\(f_t\)) \cup is-FLT(\(f_t\)) \rightarrow \]
\[ \mu_{\{s-mode:md_e, s-base:DEC, s-scale:ft_e, s-prec:*\}} \],
is-CHAR(\(f_t\)) \& is-\(Q(\(w_e\)) \rightarrow \text{CHAR-DA},
\[ is-CHAR(\(f_t\)) \& is-BIT(\(f_t\)) \rightarrow \text{char-da}(\(w_e\)),
\[ is-PIC(\(f_t\)) \rightarrow \mu(\text{pic}_e; s-mode:md_e)) \]

\[ md_e = (is-CPLX(tr) \rightarrow \text{CPLX}, is-char-pic(\text{pic}_e) \rightarrow Q, T \rightarrow \text{REAL}) \]
\[ efo_re = s\text{-real-part}(edfo) \]
\[ efo_im = s\text{-imag-part}(edfo) \]

for: is-edit-tr(tr) \& is-REAL(tr) \& is-CPLX(tr)

Ref.: convert 8-30(118), put-next-char1 10-96(261), put-format 10-93(259),
get-fo-lg 10-81(227), CHAR-DA 8-13(58), char-da 8-13(60),
is-edit-tr 10-53(137)

Note: The real and imaginary parts of a complex format and the total field
width \(w_e\) are checked with the predicate is-prop-\(w_e\).

(281) edit-op(op, edfo) =
\[ \text{is-bin-pic(pic}_e) \& \text{is-BIT(f}_t) \rightarrow \]
\[ \text{bit-char-conv} \& \text{val-list(BIT, }vr_e) \]
\[ \text{is-CHAR(f}_t) \& \text{is-PIC(f}_t) \rightarrow \]
\[ \text{val-list(CHR, }vr_e) \]
\[ \text{is-FIX(f}_t) \& d_e > 0 \rightarrow \]
\[ \text{mk-f-field(sign}_e, \text{d}_e, \text{digit-list} \& \text{abs} \& \text{trunc}(v_f, 10 \uparrow (p_e + d_e)) \)
\[ \text{is-FLT(f}_t) \& s_e > 0 \& s_e \geq d_e > 0 \rightarrow \]
\[ \text{mk-m-field(sign}_e, \text{d}_e, s_e, \text{digit-list} \& \text{abs} \& \text{trunc}(v_e, 10 \uparrow (s_e - \lg v)) \}
\[ \text{mk-x-field(sign}_x, \text{digit-list} \& \text{abs}(\lg v, (s_e - d_e)) \)
\[ T \rightarrow \text{error} \]

cont'd
where: \( v_r = (\text{is-CPLX} \cdot \text{s-mode} \cdot \text{s-da}(\text{op}) \longrightarrow \)
\[
\begin{align*}
\log_2 \left( \frac{1}{\text{int}(i + \log_2(s-vr(\text{op})))} \right) \\
T \longrightarrow s-vr(\text{op})
\end{align*}
\]

\( \log_2 = \text{length} \cdot s-vr(\text{op}) / 2 \)

\( v_e = (\text{is-CPLX} \cdot \text{s-mode} \cdot \text{s-da}(\text{op}) \longrightarrow \text{imag} \cdot \text{op-val}(\text{op}), T \longrightarrow \text{op-val}(\text{op})) \)

\( v_d = (d_e = 0 \longrightarrow \text{trunc}(v_e), T \longrightarrow v_e) \)

\( d_e = s-d(\text{edfo}) \)

\( s_e = (\text{is-Q} \cdot \text{s-s}(\text{edfo}) \longrightarrow d_e + 1, T \longrightarrow s-s(\text{edfo})) \)

\( p_e = (\text{is-Q} \cdot \text{s-p}(\text{edfo}) \longrightarrow 0, T \longrightarrow s-p(\text{edfo})) \)

\( \text{sign}_e = (v_e \geq 0 \longrightarrow >, T \longrightarrow <\text{MINUS}> ) \)

\( \text{sign}_x = (\log_2 \cdot (s_e - d_e) \geq 0 \longrightarrow <\text{PLUS}>, T \longrightarrow <\text{MINUS}> ) \)

\( \log_2 = (v_e \geq 0 \longrightarrow s_e', T \longrightarrow (i \cdot (\text{is-intg-val}(i) \& 10 \uparrow (i - 1) \leq \text{abs}(v_e) < 10 \uparrow i)) \)

Ref.: bit-char-conv 8-36(138), val-list 8-8(27),
digit-list 10-99(271), op-val 8-11(40)

Note: The last alternative represents the erroneous case of having an improper number of fractional or significant digits. This alternative applies only for fixed-point and floating-point formats, and does not depend on the total field width \( w_e \).

(282) \( \text{mk-f-field}(\text{sign}, d, d_1) = \)

\[
\begin{align*}
d = 0 \& \log_2 d_1 = 0 \longrightarrow <\text{O-CHAR}> \\
d = 0 \longrightarrow \text{sign} \land d_1 \\
d > \log_2 d_1 \longrightarrow \text{sign} \land <\text{O-CHAR}> \land <\text{POINT}> \land \left( \prod_{i=1}^{d - \log_2 d_1} \text{elem}(i, d_1) \right) \land <\text{POINT}> \land \left( \prod_{i=1}^{d} \text{elem}(i + \log_2 d_1 - d, d_1) \right)
\end{align*}
\]

(283) \( \text{mk-m-field}(\text{sign}, d, s, d_1) = \)

\[
\begin{align*}
d = 0 \& \log_2 d_1 = 0 \longrightarrow \left( \prod_{i=1}^{s} \text{list}(i, d_1) \right) \land \text{O-CHAR} \\
T \longrightarrow \text{mk-f-field}(\text{sign}, d, d_1)
\end{align*}
\]
(284) \( mk\text{-}x\text{-field}(\text{sign}, d1) = \)
\[
\text{EXP-SIZE} < \lg d1 \rightarrow \text{error} \\
T \rightarrow \langle \text{E-CHAR} \rangle \wedge \text{sign} \wedge (\text{LIST} \wedge \text{O-CHAR}) \wedge d1
\]

Note: The elementary object EXP-SIZE is an implementation defined integer. 
See the note following out-flt-pic, 8- (229).

(285) \[ \text{verify-width}(cl, edfo) = \]
\[
(\text{is-FIX}(ft_e) \lor \text{is-FLT}(ft_e)) \land w_e < \lg cl \rightarrow \\
\text{pass}(cl_{sh}); \\
\text{call-cond}(\text{SIZE}, 857) \\
\text{is-FIX}(ft_e) \lor \text{is-FLT}(ft_e) \rightarrow \\
\text{PASS}: cl_{pa} \\
T \rightarrow \\
\text{PASS}: cl
\]
where: \( cl_{sh} = \langle \text{LIST} \rangle \text{elem}(\lg cl - w_e + i, cl) \)
\( w_e = \lg d1 \)
\( cl_{pa} = (\text{LIST} \wedge \text{BLANK}) \wedge cl \)

Ref.: \text{call-cond} 9-11(18)

10.5.4.6 Check standard system action

(286) \[ \text{stand-check}(\text{ref}) = \]
\[
\text{put-data-spec}(\mu_0 (<\text{s-spec}:<\text{ref}>), u, \infty); \\
\text{upd-fu-elem}(u, s\text{-count}, \text{bintg-op}(0)); \\
\text{u;} \text{tk-stand-print-u}; \\
\text{int-stand-print-open}
\]

for: \text{is-basic-ref}(\text{ref})

Ref.: \text{int-stand-print-open} 10-16(38), \text{upd-fu-elem} 10-6(13),
\text{put-data-spec} 10-89(238), \text{bintg-op} 8-11(43)

Note: The special form of the first argument in \text{put-data-spec} is used to 
distinguish the data transmission by the check standard system ac­tion from ordinary data-directed transmission.

10.5.4.6
The execution of any display statement appends a named display message to the list of messages $s$-display($M$). If a reference to receive the reply message is specified with the display statement, the appropriate message is taken from $s$-reply($M$).

Metavariabes

- $t$ is-display-st: a text of a display statement
- $ten$ is-n: a unique name which serves to identify the display- and reply messages and the event
- $cl$ is-char-val-list: a list of character values
- $ref$ is-ref: a reference of the reply variable
- $gen$ is-gen $\vee$ is-ps-gen: a generation or pseudo-generation
- $env$ is-e: an environment
- $m$ is-m: a message part
- $rp$ is-named-message: a named reply message

(288) \textit{int-display-st}(t) =

\begin{verbatim}
int-reply-event(s-reply(t),gen-1,gen-2,ten);
gen-1:eval-opt-gen(s-reply(t)),
gen-2:eval-opt-gen(s-event(t));
display-message(ten,cl);
ten:un-name,
cl:conv-char-string(s-display(t))
\end{verbatim}

Ref.: \textit{conv-char-string} 10-12(30), \textit{un-name} 2-17(36), \textit{eval-opt-gen} 10-12(29)

(289) \textit{display-message}(ten,cl) =

\begin{verbatim}
s-m: (M;<s-display:s-display(M) \cup \mu\{<s-name:ten>,
\item \textbf{cont'd}
\end{verbatim}
where: \( lg_m = \min(\text{length}(cl), \text{MESS-LENGTH}) \)

Note: MESS-LENGTH is the implementation-defined maximum length of a display message, is-intg-val(MESS-LENGTH).

(290) \text{int-reply-event}(\text{ref}, \text{gen-1}, \text{gen-2}, \text{ten}) =
\begin{align*}
is-\Omega(\text{ref}) & \rightarrow \text{null} \\
is-\text{CHAR}\text{-gen-base}(\text{gen-1}) & \& is-\Omega(\text{gen-2}) \rightarrow \\
is-\text{CHAR}\text{-gen-base}(\text{gen-1}) & \& is-\text{EVENT}\text{-s-da}(\text{gen-2}) & \\
\neg is-\text{active}(\text{gen-2}, \text{PA}) & \rightarrow \\
s-s : el-ass(s-pp(\text{gen-2}), \text{ev-vr}(0, 0, 0), s) \\
s-te : \mu(TE; \langle s-\text{io-ev}; s-\text{io-ev}(TE) \cup \{ \text{ten} \} \rangle) \\
s-pa : \mu(\text{PA}; \langle \text{ten} ; \mu(\langle s-\text{te}; t_{ed} \rangle, \\
\langle s-c : \text{event-reply}(\text{gen-1}, \text{ten}) \rangle) \rangle) \\
T & \rightarrow \text{error}
\end{align*}

where: \( t_{ed} = \mu(\langle s-\text{ten}; s-\text{ten}(TE) \rangle, \langle s-\text{ev}; \text{gen-2} \rangle, \langle s-\text{check-list}; \mu(\langle s-\text{ref-list}; \langle \text{ref} \rangle, \langle s-e; E \rangle) \rangle) \)

Ref. : el-ass 2-22(70), ev-vr 4-7(12), gen-base 10-21(51), is-active 4-7(11)

Note: See the note following attach-io-event, 10-32(89).

(291) \text{event-reply}(\text{gen}, \text{ten}) =
\begin{align*}
\text{activate-tasks-1} ; \\
\text{int-reply}(\Omega, \text{gen}, \text{ten})
\end{align*}

Ref.: \text{activate-tasks-1} 10-33(91)

(292) \text{int-reply}(\text{ref}, \text{gen}, \text{ten}) =
\begin{align*}
is-\text{rp}_{\text{ten}} & \& is-\Omega(\text{ref}) \rightarrow \\
\text{reply-assign}(\text{gen}, \text{rp}_{\text{ten}}) \\
is-\text{rp}_{\text{ten}} \rightarrow \\
\text{call-check-cond}(\langle \text{ref} \rangle, E, 858) ; \\
\text{reply-assign}(\text{gen}, \text{rp}_{\text{ten}}) \\
T & \rightarrow \text{int-reply}(\text{ref}, \text{gen}, \text{ten})
\end{align*}

where: \( \text{rp}_{\text{ten}} = \text{s-message}(\text{rp})(\text{is-reply}(\text{ten}, \text{rp}, M)) \)

\( \text{is-}\text{rp}_{\text{ten}} = (\exists \text{rp})(\text{is-reply}(\text{ten}, \text{rp}, M)) \)

cont'd
for: is-opt-ref(ref)
Ref.: call-check-cond 9-15(27)
Note: As long as there has no proper reply message been put in s-reply(M)
from outside the PL/I machine, the instruction replaces itself in
the control.

(293) reply-assign (gen, cl) =
    ~is-Q(cl1) \rightarrow
    convert-assign (gen, val-op (char-da-length (cl1), cl1))
    T \rightarrow error
where: cl1 = adjust-str (gen, cl)
Ref.: convert-assign 7-9(19), adjust-str 10-21(70), char-da 8-13(60),
val-op 8-11(41)

(294) is-reply (ten, rp, m) =
    (\exists i) (rp = elem (i, s-reply (m)) \& is-named-message (rp) \& ten = s-name (rp))
Ref.: is-named-message 2-44(144)

10.7. Auxiliary Definitions

(295) opt-list (ref) =
    is-Q (ref) \rightarrow <>
    T \rightarrow <ref>
for: is-opt-ref (ref)
11. BUILTIN FUNCTIONS

This chapter handles references to built-in functions, which occur in expressions to be evaluated. 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 11.1), the evaluation of a scalar reference (section 11.2), the determination of data attributes (section 11.3), and the handling of built-in functions which are passed to entry parameters (section 11.4). Section 11.5 contains a survey of all built-in functions by a table giving their characteristic properties. Finally, section 11.6 describes the evaluation of the individual built-in functions.

11.1 SYNTACTIC EXPANSION OF BUILTIN 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. 7.1), where each step in the expansion process is accomplished by application of the modifying instruction mod to the text of the expressions. On modifying references to built-in functions the distinction is to be made between expanding arguments and non-expanding arguments of the built-in function. 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-built in}(t,eda,i,env), where \( t \) is the text of the reference, \( \text{eda} \) are the attributes of the master variable for the expansion, \( i \) is the number of the sub-part which is to be referenced by the modified reference (or a sub-aggregate name in case of BY NAME expansion, cf. 7.1.2), and \( \text{env} \) is the environment. An argument place is expandable if \texttt{arg-class-1(id, i) = EXPND}, where \( \text{id} \) is the identifier of the built-in function and \( i \) the number of the argument (cf. 11.5.1.3, and \texttt{arg-class-1 11-9(19)}).

**Metavariabes**

\[
\begin{align*}
\text{t} & \quad \text{is-ref} \quad \text{the reference to a built-in function} \\
\text{env} & \quad \text{is-e} \\
\text{eda} & \quad \text{is-eda} \quad \text{the evaluated data attributes of the master variable} \\
i & \quad \text{is-intg-val \lor is-id}
\end{align*}
\]
(1) \[ \text{mod-builtin}(t, \text{eda}, i, \text{env}) = \]
\[ \text{is-BUILTIN}(\text{attr}_t) \rightarrow \text{mod-builtin-1}(t, \text{eda}, i, \text{env}) \]
\[ \text{is-param-builtin}(\text{attr}_t) \rightarrow \text{mod-param-builtin}(t, i) \]

where: \( \text{attr}_t = \text{s-attr}(\text{sel}\cdot\text{s-id-list}(t)(\text{env})(\text{AT})) \)

Ref.: \text{mod-param-builtin 11-12(26)}

Note: Builtin functions passed to parameters are treated separately in chapter 11.4.

(2) \[ \text{mod-builtin-1}(t, \text{eda}, i, \text{env}) = \]
\[ \text{mk-builtin-ref}(\text{s-id-list}(t), \text{list}); \]
\[ \{ \text{elem}(j)(\text{list}): \text{mod}(\text{elem}(j, \text{sl}_t), \text{eda}, i, \text{env}) \mid \]
\[ 1 \leq j \leq \text{length}(\text{sl}_t) \& \text{is-EXPND-arg-class-1}(\text{id}_t, j) \} \cup \]
\[ \{ \text{elem}(j)(\text{list}): \text{pass}(\text{elem}(j, \text{sl}_t)) \mid \]
\[ 1 \leq j \leq \text{length}(\text{sl}_t) \& \neg \text{is-EXPND-arg-class-1}(\text{id}_t, j) \} \]

where: \( \text{sl}_t = \text{s-arg-list}(t) \)
\( \text{id}_t = \text{head}\cdot\text{s-id-list}(t) \)

Ref.: \text{mod 7-7(13), arg-class-1 11-9(19)}

(3) \[ \text{mk-builtin-ref}(\text{idl}, \text{list}) = \]
\[ (\exists i)(\text{is-\&\&elem}(i, \text{list})) \rightarrow \text{PASS:*} \]
\[ \text{is-\&\&}(\text{list}) \rightarrow \text{mk-basic-ref}(\text{idl}, <>) \]
\[ T \rightarrow \text{mk-basic-ref}(\text{idl}, \text{list}) \]

for: \( \text{is-id-list}(\text{idl}), \text{is-arg-expr-list}(\text{list}) \lor \text{is-\&\&}(\text{list}) \)

Ref.: \text{mk-basic-ref 10-77(208)}
11.2 Evaluation of Builtin Function References

This section describes the evaluation of a builtin function reference occurring in a scalar expression to be evaluated (cf. 7.2.1). If it is no builtin function passed as argument to an entry parameter (cf. 11.4), the reference to a builtin function is evaluated in two steps: First, the evaluation of the argument list, and second, the activation of the individual builtin function. The latter is performed by the instruction `eval-builtin-2` which performs the case distinction according to the different builtin identifiers and is defined by the table given in section 11.5.

The argument evaluation constructs a list of "builtin arguments", for each argument expression occurring in the text of the reference one builtin argument. The evaluation of a single argument is governed by the function `arg-type(id, i)`, defined by the table in section 11.5, which specifies for each individual builtin function identifier `id` and argument position `i` the type of the builtin argument. The possible types are: an operand, an operand list, an integer value, evaluated data attributes, a generation, the text of the argument expression itself. During the argument evaluation, governed by the function `arg-test(id, i)`, which is defined by the table in section 11.5 and depends on the individual builtin identifier `id` and argument position `i`, the arguments are tested for correctness. For builtin arguments of type operand or operand list, the single operands are converted to (incomplete) data attributes specified by the function `arg-da(id, i)`, which also is defined by the table in section 11.5 for each individual builtin identifier `id` and argument position `i`. Finally, the length of the argument list is tested, governed by the function `lmin(id)`, which is defined for each builtin identifier `id` by the table in section 11.5 and denotes the minimum length of the argument list, and by the function `arg-type(id, i)`, which yields `0` if `i` exceeds the maximum length.

**Metavariables**

- `id` is-id the builtin function identifier
- `i` is-intg-val an integer denoting an argument position
- `t` is-arg-expr in the first two instructions the text of the builtin function reference; in all other cases the text of an argument expression;
- `env` is-e the environment in which the evaluation is performed
11.2

(4) \( \text{eval-builtin}(t, \text{env}) = \)

\[ \text{is-BUILTIN}(\text{attr}_t) \rightarrow \text{eval-builtin-l}(t, \text{env}) \]

\[ \text{is-param-builtin}(\text{attr}_t) \rightarrow \text{eval-param-builtin}(t, \text{env}) \]

where: \( \text{attr}_t = \text{s-attr}\left(\text{sel}\cdot\text{s-id-list}(t)\left(\text{env}\right)\left(\text{AT}\right)\right) \)

Ref.: \( \text{is-param-builtin} \ 2-19(41), \ \text{eval-param-builtin} \ 11-12(27) \)

(5) \( \text{eval-builtin-l}(t, \text{env}) = \)

\[ \text{eval-builtin-2}(\text{id}_t, \text{bal}, \text{env}); \]

\[ \text{bal}: \text{builtin-arg-list}(\text{s-arg-list}(t), \text{id}_t, \text{id}, \text{env}) \]

where: \( \text{id}_t = \text{head}\cdot\text{s-id-list}(t) \)

Ref.: \( \text{eval-builtin-2} \ 11-21(40) \)

(6) \( \text{builtin-arg-list}(\text{expr-list}, \text{id}, \text{i}, \text{env}) = \)

\[ \text{is-<>}(\text{expr-list}) \& \text{i} = \text{lmin}(\text{id}) \rightarrow \text{PASS};<> \]

\[ \text{is-<>}(\text{expr-list}) \rightarrow \text{error} \]

\[ \text{T} \rightarrow \]

\[ \text{mk-list}(\text{ba}, \text{bal}); \]

\[ \text{bal}: \text{builtin-arg-list}(\text{tail}(\text{expr-list}), \text{id}, \text{i}+1, \text{env}); \]

\[ \text{bal}: \text{builtin-arg}(\text{head}(\text{expr-list}), \text{id}, \text{i}\text{id}, \text{env}) \]

where: \( \text{id} = (\text{id} \in \{\text{mk-id}(\text{MAX}), \text{mk-id}(\text{MIN})\} \rightarrow 1, \]

\[ \text{T} \rightarrow \text{i} + 1 \]

for: \( \text{is-arg-expr-list}(\text{expr-list}) \)

Ref.: \( \text{lmin} \ 11-17(34), \ \text{mk-id} \ 10-76(214) \)

Note: Since the builtin functions \( \text{MAX} \) and \( \text{MIN} \) may have an arbitrary number of arguments to be handled in the same way the position counter \( i \) is reset to 1 for each single argument evaluation. Nevertheless, it is updated in the list in order to test the final condition \( \text{i} = \text{lmin}(\text{id}) \).
(7) \[ \text{builtin-arg}(t, id, i, \text{env}) = \]
\[
\begin{align*}
\text{is-} & \ast(t) \rightarrow \text{error} \\
\text{is-OP} & \text{-arg-type}(id, i) \rightarrow \text{builtin-arg-op}(t, id, i, \text{env}) \\
\text{is-OPL} & \text{-arg-type}(id, i) \rightarrow \text{builtin-arg-opl}(t, id, i, \text{env}) \\
\text{is-INTG} & \text{-arg-type}(id, i) \rightarrow \text{builtin-arg-intg}(t, id, i, \text{env}) \\
\text{is-EDA} & \text{-arg-type}(id, i) \rightarrow \text{eda-expr}(t, \text{env}) \\
\text{is-GEN} & \text{-arg-type}(id, i) \rightarrow \text{eval-ref-gen}(t, \text{env}) \\
\text{is-TEXT} & \text{-arg-type}(id, i) \rightarrow \text{builtin-arg-text}(t, id, i, \text{env}) \\
\text{is-} & \Omega \text{-arg-type}(id, i) \rightarrow \text{error}
\end{align*}
\]

Ref.: eda-expr 7-22(54), arg-type 11-18(36), eval-ref-gen 7-25(59)

(8) \[ \text{builtin-arg-op}(t, id, i, \text{env}) = \]
\[
\begin{align*}
\text{test-builtin-arg-op}(op, \text{arg-test}(id, i)) ; \\
op & \text{-convert}(\text{arg-da-1}(id, i), \text{op-1}) ; \\
op-1 & \text{-eval-expr}(t, \text{env})
\end{align*}
\]

Ref.: eval-expr 7-18(43), convert 8-30(118), arg-da-1 11-10(24), arg-test 11-19(38)

(9) \[ \text{test-builtin-arg-op}(op, \text{test}) = \]
\[
\begin{align*}
is- & \text{-REAL}(\text{test}) \cup is-\text{-REAL-s-mode-s-da}(op) \rightarrow \text{pass}(op) \\
T \rightarrow & \text{error}
\end{align*}
\]

for: is-op(op), test \( \in \{ T, \text{REAL}, \text{ARR}, \text{LIN-ARR}, \text{LIN} \} \)

(10) \[ \text{builtin-arg-opl}(t, id, i, \text{env}) = \]
\[
\begin{align*}
\text{builtin-arg-opl-1}(t, \text{eda}, id, i, \text{env}) ; \\
\text{test-builtin-arg-opl}(\text{eda}, \text{arg-test}(id, i)) ; \\
\text{eda} & \text{-eda-expr}(t, \text{env})
\end{align*}
\]

Ref.: eda-expr 7-22(54), arg-test 11-19(38)

(11) \[ \text{test-builtin-arg-opl}(\text{eda}, \text{test}) = \]
\[
\begin{align*}
is- & \text{-ARR}(\text{test}) \& is-\text{array}(\text{eda}) \vee \\
is- & \text{-LIN-ARR}(\text{test}) \& is-\text{scalar-s-elem}(\text{eda}) \vee \\
is- & \text{-LIN}(\text{test}) \& (is-\text{scalar-s-elem}(\text{eda}) \vee is-\text{scalar}(\text{eda})) \rightarrow \text{null} \\
T \rightarrow & \text{error}
\end{align*}
\]

for: is-eda(eda), test \( \in \{ \text{ARR}, \text{LIN-ARR}, \text{LIN} \} \)
(12) \texttt{builtin-arg-opl-1(t,eda,id,i,env)} =
\begin{align*}
\text{is-scalar}(eda) & \quad \rightarrow \quad \text{mk-list}(op,<>) ; \\
& \quad \quad \text{op;builtin-arg-op}(t,id,i,env) \\
\text{is-array}(eda) & \quad \rightarrow \quad \text{iterate-opl}(t,eda,id,i,env,s-lbd(eda)) \\
T & \quad \rightarrow \quad \text{error}
\end{align*}
for: \text{is-eda}(eda)

Note: This instruction and the iteration mechanism tests also that the base element of eda is scalar.

(13) \texttt{iterate-opl(t,eda,id,i,env,k)} =
\begin{align*}
k>s-\text{ubd}(eda) & \quad \rightarrow \quad \text{PASS}:<> \\
T & \quad \rightarrow \quad \text{conc}(opl-1,opl-2) ; \\
& \quad \quad \text{opl-2;iiterate-opl}(t,eda,id,i,env,k+1) ; \\
& \quad \quad \text{opl-1;builtin-arg-opl-1}(t-1,s-\text{elem}(eda),id,i,env) ; \\
& \quad \quad t-1:\text{mod}(t,eda,k,env)
\end{align*}
for: \text{is-eda}(eda), \text{is-intg-val}(k)
Ref.: \text{mod 7-7(13), conc 9-16(30)}

(14) \texttt{builtin-arg-intq(t,id,i,env)} =
\begin{align*}
\text{is-INTG}(\text{tst}_1) & \quad \& \quad \text{is-intg-on(t)} \quad \lor \\
\text{is-S-INTG}(\text{tst}_1) & \quad \& \quad \text{is-intg-const(t)} \quad \lor \\
\text{is-T}(\text{tst}_1) & \quad \rightarrow \quad \text{eval-intg-expr}(t,env) \\
T & \quad \rightarrow \quad \text{error}
\end{align*}
where: \text{tst}_1 = \text{arg-test}(id,i)
Ref.: \text{eval-intg-expr 7-21(52), arg-test 11-19(38), is-intg-op 8-11(44), is-intg-const 5-41(69)}
\((15) \) \(\text{builtin-arg-text}(t, id, i) =\)

\(\text{is-SIMP-REF}(\text{test}_i) \supset \text{is-simple-ref}(t) \rightarrow \text{PASS}: t\)

\(T \rightarrow \text{error}\)

where: \(\text{test}_i = \text{arg-test}(id, i)\)
Ref.: \(\text{arg-test} 11-19(38)\)

11.3 The Data-Attributes of Builtin Function References

The data-attributes of the aggregate returned by a builtin function reference are evaluated by the instruction \(\text{eda-builtin}(t, \text{env})\), where \(t\) is the text of the reference and \(\text{env}\) the environment. The instruction is needed in the definition of the instruction \(\text{eda-expr}\) (cf. 7.2.3).

If all expanding arguments of a builtin function reference are scalar, then the operand returned is also scalar. The data attributes of the returned scalar are given by the function \(\text{rt-builtin}(id, \text{list})\), where \(id\) is the identifier of the builtin function and \(\text{list}\) contains elements corresponding to the arguments of the reference. These elements are either data attributes (the data attributes of the corresponding argument for expanding arguments places, or the data attributes of the base element of the corresponding argument array, for aggregate arguments places), 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 data attributes of the returned operand.

The function \(\text{rt-builtin}\) is defined by the table in chapter 11.5.

Metavariables

- \(t\) is-ref \(\) the builtin function reference
- \(\text{env}\) is-e
- \(id\) is-id \(\) the builtin function identifier
- \(i\) is-intg-val \(\) an argument position of the builtin function
- \(\text{list}\) is-built-in-arg-descr-list, where is-built-in-arg-descr = is-eda \(\lor\) is-intg-val \(\lor\) is-*
(16) \( \text{eda-built in}(t, \text{env}) = \)

\[
\text{is-BUILTIN}(\text{attr}_t) \rightarrow \text{eda-built in-1}(t, \text{env}) \\
\text{is-param-built in}(\text{attr}_t) \rightarrow \text{eda-param-built in}(t, \text{env})
\]

where: \( \text{attr}_t = s\text{-attr}(s\text{-id-list}(t)(\text{env})(\text{AT})) \)

Ref.: \text{eda-param-built in} 11-14(33)

Note: Builtin functions passed to parameters are treated separately in chapter 11.4.

(17) \( \text{eda-built in-1}(t, \text{env}) = \)

\[
\text{is-\langle\rangle}(s_l_t) \rightarrow \text{pass-eda-built in-2}(id_t, \text{\langle\rangle}) \\
\neg(\exists i)(\text{is-\ast\ast}\text{elem}(i, s_l_t)) \rightarrow \\
\text{pass-eda-built in-2}(id_t, \text{list}); \\
\{\text{elem}(i)(\text{list}): \text{eda-expr}(\text{elem}(i, s_l_t), \text{env}) | \\
1 \leq i \leq \text{length}(s_l_t) \& (\text{is-EXPND} \vee \text{is-AGGR})(\text{arg-class-1}(id_t, i))\} \cup \\
\{\text{elem}(i)(\text{list}): \text{eval-intg-expr}(\text{elem}(i, s_l_t), \text{env}) | \\
1 \leq i \leq \text{length}(s_l_t) \& \text{is-CONST}\text{arg-class-1}(id_t, i)\} \cup \\
\{\text{elem}(i)(\text{list}): \text{pass}(\ast) | \\
1 \leq i \leq \text{length}(s_l_t) \& \text{is-IRREL}\text{arg-class-1}(id_t, i)\}
\]

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

where: \( id_t = \text{head}\text{-s-id-list}(t) \)
\( s_l_t = \text{s-arg-list}(t) \)

Ref.: \text{eda-expr} 7-22(54), \text{eval-intg-expr} 7-21(52)

Note: The function \text{arg-class-1}, defining the nature of the argument places of builtin functions, is defined by the function \text{arg-class}, which is given by the table in chapter 11.5.
(18) \( \text{eda-builtin-2}(id,\text{list}) = \)

\[
\begin{align*}
(\exists i) & \ (\text{is-EXPND} \cdot \text{arg-class-1}(id,i) \ \& \ \text{is-array\_elem}(i,\text{list})) \ \& \\
\sum_{i=1}^{\text{length(\text{list})}} & \ (\text{is-EXPND} \cdot \text{arg-class-1}(id,i) \ \& \ \text{is-array\_elem}(i,\text{list}) \ \supset \\
& \text{dim}(\text{elem}(i,\text{list})) = \text{dim}_{\text{list}}) \ \supset \\
& \mu_{o}(\langle s-lbd: \text{lbdl}_{\text{list}} \rangle, \langle s-ubd: \text{ubdl}_{\text{list}} \rangle, \\
& \langle s-\text{elem} : \text{eda-builtin-2}(id, [\text{LIST} \ \text{array\_elem}(id,i,\text{list})]) \rangle) \\
(\exists i) & \ (\text{is-EXPND} \cdot \text{arg-class-1}(id,i) \ \& \ \text{is-struct\_elem}(i,\text{list})) \ \& \\
(\neg (\exists i)) & \ (\text{is-EXPND} \cdot \text{arg-class-1}(id,i) \ \& \ \text{is-array\_elem}(i,\text{list})) \ \supset \\
\sum_{i=1}^{\text{length(\text{list})}} & \ [\{<s-\text{da}\_\text{elem}(i) : \text{eda-builtin-2}(id, [\text{LIST} \ \text{struct\_elem}(id,i,j,\text{list})])> | \\
& 1 \leq i \leq \text{order}_{\text{list}}\}]
\end{align*}
\]

where: \( \text{lbdl}_{\text{list}} = (\langle k \rangle (\sum_{i=1}^{\text{length(\text{list})}} (\text{is-EXPND} \cdot \text{arg-class-1}(id,i) \ \& \\
& \text{is-array\_elem}(i,\text{list}) \ \supset \ k=s-lbd\_\text{elem}(i,\text{list})))) \)

\( \text{ubdl}_{\text{list}} = (\langle k \rangle (\sum_{i=1}^{\text{length(\text{list})}} (\text{is-EXPND} \cdot \text{arg-class-1}(id,i) \ \& \\
& \text{is-array\_elem}(i,\text{list}) \ \supset \ k=s-ubd\_\text{elem}(i,\text{list})))) \)

\( \text{dim}_{\text{list}} = (\langle k \rangle (\sum_{i=1}^{\text{length(\text{list})}} (\text{is-EXPND} \cdot \text{arg-class-1}(id,i) \ \& \\
& \text{is-array\_elem}(i,\text{list}) \ \supset \ k=\text{dim}(\text{elem}(i,\text{list})))) \)

\( \text{order}_{\text{list}} = (\langle k \rangle (\sum_{i=1}^{\text{length(\text{list})}} (\text{is-EXPND} \cdot \text{arg-class-1}(id,i) \ \& \\
& \text{is-struct\_elem}(i,\text{list}) \ \supset \ k=\text{order}(\text{elem}(i,\text{list})))) \)

(19) \( \text{arg-class-1}(id,i) = \)

\[
\begin{align*}
\text{id} & \in \{\text{mk-id(MIN)}, \text{mk-id(MAX)}\} \ \supset \ \text{arg-class}(id,1) \\
T & \ \supset \ \text{arg-class}(id,1)
\end{align*}
\]

Ref.: arg-class 11-18(35), mk-id 10-76(214)
(20) array-elem(id,i,list) =
    is-EXPND•arg-class-1(id,i) & is-array-elem(i,list) "-->
    s-elem-elem(i,list)
    T "--" elem(i,list)

(21) struct-elem(id,i,j,list) =
    is-EXPND•arg-class-1(id,i) & is-struct-elem(i,list) "-->
    s-da•elem(j,elem(i,list))
    T "--" elem(i,list)

(22) rt-builtin-1(id,list) =
    length(list)
    rt-builtin(id, LIST ![ST convert-elem(id,i,list)])

Ref.: rt-builtin 11-20(39)
Note: The function rt-builtin is defined by the table in chapter 11.5.

(23) convert-elem(id,i,list) =
    is-**arg-da-l(id,i) "--" elem(i,list)
    is-EXPND•arg-class-1(id,i) "--" complete-tg(arg-da-l(id,i),elem(i,list))
    is-AGGR•arg-class-1(id,i) "--"
    complete-tg(arg-da-l(id,i),base-elem•elem(i,list))

Ref.: complete-tg 8-37(143), base-elem 7-28(71)

(24) arg-da-l(id,i) =
    id € {mk-id(MIN),mk-id(MAX)} "--" arg-da(id,1)
    T "--" arg-da(id,1)

Ref.: arg-da 11-19(37), mk-id 10-76(214)
Note: The function arg-da is defined by the table in chapter 11.5.
11.4 Builtin Functions Passed to Entry Parameters

It is possible to pass float arithmetic generic builtin functions to entry parameters of a procedure call. The identifiers of these builtin functions are characterized by the predicate is-float-generic-built-in.

Similar as on passing of a generic entry to an entry parameter, in this case the generic selection is made before the passing according to the nested parameter descriptors of the entry attribute. The generic selection is performed by passing as denotation of the selected member of the builtin function family an object consisting of the builtin identifier and the nested parameter descriptor list, if the latter fits to the special builtin function at all (cf. 5.6.2.1). The entry parameter to which this denotation is passed may only be used as the denoted member of the builtin function family.

An entry parameter, to which a builtin function is passed, is in the following shortly named as "builtin parameter". It is distinguished from an entry or an entry parameter, to which an entry is passed, by the s-builtin component T added to its attribute in the attribute directory AT, resulting in an attribute satisfying the predicate is-param-builtin (cf. 5.6.2.4). A builtin parameter is in all respects to be handled corresponding to an entry and not to a builtin function; that means the following:

1. On installation of the passed denotation (cf. 5.6.2.4) for the builtin parameter or on further passing of this builtin parameter to an entry parameter of a nested procedure call (cf. 5.6.2.1 and 5.6.2.4) no matching of nested parameter descriptors or return types is required.

2. On occurrence of a builtin parameter reference in an aggregate expression the syntactic modification (cf. 7.1.3) does not depend on the arguments, but is performed like that of a scalar. This is done by the instruction mod-param-builtin.

3. On evaluation of a builtin parameter reference for expression evaluation (cf. 7.2.1), first the arguments are evaluated and converted to the data attributes of the parameter descriptors given in the builtin parameter's declaration. Thereby only one or two scalar arithmetic arguments are allowed. Then the call of the member of the builtin family passed to the builtin parameter is performed; this call consists of: Testing of the converted arguments against the parameters of this member (contained in its denotation), testing of the declared return type against the actual return type of this member.
(received by the instruction eda-built-in-1) and evaluation of the denoted
builtin function. This is done by the instruction eval-param-built-in.

Note: The instructions used for this purpose are named similar to instruc-
tions in sections 5.6.2 and 5.6.3 (with the additional prefix "pb"
standing for "parameter-built-in"), in order to signify corresponding
actions for call of either a normal entry or a builtin parameter,
though these actions themselves are completely different.

4. For evaluation of the resulting data attributes of an expression
(cf. 7.2.3) the return type given in the builtin parameter's declaration
is taken and not the actual return type resulting from evaluation of the
builtin function. This is done by the instruction eda-param-built-in.

(25) is-float-generic-built-in(id) =

\[
\begin{align*}
\text{id} \in & \{ \text{mk-id(\text{EXP}), mk-id(\text{LOG}), mk-id(\text{LOG10}), mk-id(\text{LOG2}), mk-id(\text{ATAND}),} \\
& \text{mk-id(\text{ATAN}), mk-id(\text{TAND}), mk-id(\text{TAN}), mk-id(\text{SIND}), mk-id(SIN),} \\
& \text{mk-id(\text{COSD}), mk-id(COS), mk-id(\text{TANH}), mk-id(ERF), mk-id(SQRT),} \\
& \text{mk-id(ERFC), mk-id(COSH), mk-id(SINH), mk-id(\text{ATANH})} \}
\end{align*}
\]

for: is-id(id)
Ref.: mk-id 10-76(214)

(26) mod-param-built-in(t,i) =

\[
\begin{align*}
\text{is-intg-val(i) } & \longmapsto \text{PASST} \\
\text{is-id(i) } & \longmapsto \text{ERROR}
\end{align*}
\]

for: is-intg-val(i) \lor is-id(i), is-ref(t)

(27) eval-param-built-in(t,env) =

\[
\begin{align*}
evaluatibuiltin-1(t-1, Q) ; \\
\text{test-pb-ret-type(t-1,s-ret-type(atr_t)) ;} \\
t-1 \text{=} \text{mk-basic-ref(<s-id(den_t), opl) ;} \\
\text{install-pb-arg-list(opl, s-param-list(den_t)) ;} \\
opl \text{=} \text{eval-pb-arg-list(s-arg-list(t), s-param-list(atr_t), env)}
\end{align*}
\]

where:
\[
\begin{align*}
\text{nt} & = \text{sel\#s-id-list(t)(env)} \\
\text{attr_t} & = \text{s-attr\#n_t(AT)} \\
\text{den_t} & = \text{n_t(DM)}
\end{align*}
\]

for: is-ref(t), is-e(env)
Ref.: evaluatibuiltin-1 11-4(5), mk-basic-ref 10-77(218)
(28) \( \text{eval-pb-arg-list}(\text{expr-list}, \text{dl}, \text{env}) = \)

\( \text{is-\ast}(\text{dl}) \) \& \( \text{lslength}(\text{expr-list}) \leq 2 \) \rightarrow 

\( \text{pass}(\text{op1}); \)

\( \text{elem}(2)(\text{op1}): \text{eval-pb-arg}(\text{elem}(2, \text{expr-list}), \ast, \text{env}); \)

\( \text{elem}(1)(\text{op1}): \text{eval-pb-arg}(\text{elem}(1, \text{expr-list}), \ast, \text{env}) \)

\( \text{is-\ast}(\text{dl}) \rightarrow \text{error} \)

\( \text{lslength}(\text{expr-list}) = \text{length}(\text{dl}) \leq 2 \) \rightarrow 

\( \text{pass}(\text{op1}); \)

\( \text{elem}(2)(\text{op1}): \text{eval-pb-arg}(\text{elem}(2, \text{expr-list}), \text{elem}(2, \text{dl}), \text{env}); \)

\( \text{elem}(1)(\text{op1}): \text{eval-pb-arg}(\text{elem}(1, \text{expr-list}), \text{elem}(1, \text{dl}), \text{env}) \)

\( T \rightarrow \text{error} \)

for: \( \text{is-arg-expr-list}(\text{expr-list}), \text{is-decr-list}(\text{dl}) \) \& \( \text{is-\Omega}(\text{dl}), \text{is-e}(\text{env}) \)

Note: For most of the relevant built-in functions only one argument is allowed. In these cases \( \text{elem}(2, \text{expr-list}) \) will be \( \Omega \) and also \( \text{eval-pb-arg} \) will yield \( \Omega \) as \( \text{elem}(2)(\text{op1}) \).

(29) \( \text{eval-pb-arg}(\text{expr, descr, env}) = \)

\( \text{is-expr(\text{expr}) \& is-arithmetic-\text{sd}(\text{descr})} \rightarrow 

\( \text{convert(\text{sd}(\text{descr}), op);} \)

\( \text{op: eval-expr(\text{expr, env})} \)

\( \text{is-\text{expr}(\text{expr}) \& is-\ast(\text{descr})} \rightarrow 

\( \text{eval-expr(\text{expr, env})} \)

\( \text{is-\Omega}(\text{expr}) \rightarrow \text{PASS: \Omega} \)

\( T \rightarrow \text{error} \)

for: \( \text{is-\text{expr(\text{expr}) \& is-\Omega(\text{expr}), is-descr(\text{descr}), is-e(\text{env})} \)

Ref.: eval-expr 7-18(43), convert 8-30(118)

Note: The test that the descriptor is arithmetic, is made only to guarantee that the first argument of convert satisfies is-scalar-ed. The complete test is made by install-pb-arg-list.
\[(30) \text{install-pb-arg-list}(opl, dl-1) = \]
\[
\text{length}(opl) = \text{length}(dl-1) \&
\]
\[
\text{length}(opl)
\]
\[
E_t \ (s\text{-da\_elem}(i, opl) = s\text{-da\_elem}(i, dl-1)) \rightarrow \text{null}
\]
\[
T \rightarrow \text{error}
\]
for: is-op-list(opl), is-descr-list(dl-1)

\[(31) \text{test-pb-ret-type}(t-1, rt) = \]
\[
\text{test-pb-ret-type-l}(rt-1, rt);
\]
\[
rt-1: \text{eda-builtin-l}(t-1, Q)
\]
for: is-basic-ref(t-1), is-ret-type(rt)
Ref.: \text{eda-builtin-l 11-8(17)}

\[(32) \text{test-pb-ret-type-l}(rt-1, rt) = \]
\[
rt-1 = rt \rightarrow \text{null}
\]
\[
T \rightarrow \text{error}
\]
for: is-arithm(rt-1), is-ret-type(rt)

\[(33) \text{eda-param-builtin}(t, env) = \]
\[
\text{is-arithm\_s\_ret-type}(attr_t) \rightarrow \text{PASS}; \text{s\_ret-type}(attr_t)
\]
\[
T \rightarrow \text{error}
\]
where: \(attr_t = s\text{-attr}(sel\_s\_id\_list(t)(env)\_\text{AT})\)

Note: The test is performed only to guarantee that the passed result satisfies is-eda.
11.5 Table of Builtin Functions

11.5.1 Explanation

The following table (cf. 11.5.2) lists all PL/I builtin functions and gives a survey of their characteristic properties: admissible numbers and kinds of arguments and data attributes of the result values ("return types"). Furthermore, it lists the instructions, defined in section 11.6, which after suitable preparation of arguments evaluate the individual builtin functions.

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 lmin(id), arg-class(id,i), arg-type(id,i), arg-da(id,i), arg-test(id,i), rt-builtin(id,list) and the instruction eval-builtin-2(id,list,env) by means of conditional expressions. These functions and instruction are used in sections 11.1, 11.2 and 11.3. 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 instruction, respectively. Thereby, of course, the argument column i has to be neglected for lmin(id), rt-builtin(id,list) and eval-builtin-2(id,list,env) 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:

\[
\text{arg-type}(id,i) = \\
\begin{align*}
& \text{id} = \text{mk-id}(\text{ABS}) \& i = 1 \rightarrow \text{OP} \\
& \text{id} = \text{mk-id}(\text{MAX}) \& i = 1 \rightarrow \text{OP} \\
& \ldots \\
& \text{id} = \text{mk-id}(\text{FIXED}) \& i = 1 \rightarrow \text{OP} \\
& \text{id} = \text{mk-id}(\text{FIXED}) \& i = 2 \rightarrow \text{INTG} \\
& \text{id} = \text{mk-id}(\text{FIXED}) \& i = 3 \rightarrow \text{INTG} \\
& \ldots \\
& \text{id} = \text{mk-id}(\text{PRIORITY}) \& i = 1 \rightarrow \text{OP} \\
& \text{T} \rightarrow \varnothing
\end{align*}
\]
Or, the definition of the instruction \texttt{eval-builtin-2} reads:

\begin{verbatim}
\texttt{eval-builtin-2(id, list, env) =}
  id = \texttt{mk-id(ABS)} \rightarrow \texttt{eval-abs(op₁)}
  id = \texttt{mk-id(MAX)} \rightarrow \texttt{eval-max(op₀)}
  \vdots
  id = \texttt{mk-id(FIXED)} \rightarrow \texttt{eval-fixed(op₁, k₂, k₃)}
  \vdots
  id = \texttt{mk-id(PRIORITY)} \rightarrow \texttt{eval-priority(op₁)}
\end{verbatim}

For more details see the following descriptions of the single columns of the table.

\textbf{11.5.1.1 The argument columns id and i}

The first column contains the builtin function identifiers in their concrete representation. To use these identifiers in the formal definition, one has to apply the function \texttt{mk-id(10-76(214))} to them in order to have objects satisfying the predicate \texttt{is-id(12-5(7))}.

The second column contains integer values denoting for each single builtin 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-da} and \texttt{arg-test}, which describe properties of the single arguments of the different builtin functions. Since the builtin functions MAX and MIN have an indefinite number of arguments of the same kind, for these two functions only one entry with \texttt{i = 1} is contained in the table.

\textbf{11.5.1.2 Number of arguments of builtin functions}

Several builtin 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 builtin function is determined by the number of entries in the column \texttt{i}, except for the two builtin 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

(34) \text{\texttt{lmin(id)}}

for: \texttt{is-id(id)}

Note: The range of this function is the set of integer values \{0,1,2,3\}. The default value of this function for all non-builtin identifiers is 1. This value guarantees that erroneous programs are rejected by the instructions \texttt{builtin-arg-list} or \texttt{builtin-arg} in section 11.2.

11.5.1.3 The classes of arguments of builtin functions

The possible arguments of built-in functions may be categorized into four classes, which are characterized by the four elementary objects \texttt{EXPND}, \texttt{CONST}, \texttt{AGGR}, \texttt{IREEL}. This categorization considers the way, how the different arguments of built-in functions influence the syntactic expansion of aggregate expressions to scalar expressions (cf. 7.1.3 and especially 11.1) and the data attributes of the result of an expression (cf. 7.2.3 and especially 11.3):

\texttt{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. 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. 11.5.1.5) influence the data attributes of the result of the built-in function.

\texttt{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 \texttt{builtin-arg-intg} (cf. 11-5(7), 11-6(14)): as can be seen from the table in 11.5.2, arg-type is \texttt{INTG}, and arg-test is \texttt{INTG} or \texttt{S-INTG}, whenever arg-class is \texttt{CONST} (cf.11.5.1.4,11.5.1.6).

\texttt{AGGR}. The arguments belonging to this class may be aggregate expressions, but they do not influence the syntactic expansion of expressions (like aggregate arguments of programmed function procedures). Their data attributes after appropriate conversion of the scalar components, influence the data attributes of the result.

\texttt{IREEL}. The arguments belonging to this class are irrelevant both for syntactic expansion and for determination of data attributes of the result.
The classification into these four classes is given by the fourth column of the table. It defines the function

\[(35) \text{arg-class}(id, i)\]

for: is-id(id), is-intg-val(i)

Note: The range of this function is the set of the four elementary objects \{EXPND, CONST, AGGR, IRREL\} and the null object \(\mathcal{Q}\) as default value for erroneous cases.

11.5.1.4 The types of arguments of builtin functions

On evaluation of a builtin function reference, the arguments are evaluated by a general mechanism (cf. 11.2) and passed to the individual instruction for the builtin 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 data attributes of an aggregate expression, a generation, the unchanged 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

\[(36) \text{arg-type}(id, i)\]

for: is-id(id), is-intg-val(i)

Note: The range of this function is the set of the six elementary objects \{OP, OPL, INTG, EDA, GEN, TEXT\} and the null object \(\mathcal{Q}\) as default value for erroneous cases.

11.5.1.5 Conversion of arguments of builtin functions

In those cases where evaluated operands or operand lists of the arguments are passed to the instruction of a builtin function, before passing these operands are converted to specified (incomplete) data attributes, e.g. real, float, string, bit etc. The target data attributes for this conversion are given in the sixth column of the table. An asterisk denotes that no conversion takes place.
This column defines the function

(37) \text{arg-da}(id,i)

for: \text{is-id}(id), \text{is-intg-val}(i)

Ref.: AR-DA 8-13(56), STRING-DA 8-13(57), BIT-DA 8-13(59),
bit-da 8-13(61), CPLX-DA 11-33(51), FLT-DA 11-33(49),

Note: The range of this function is the set of objects \{x, PTR, OFFSET, TASK, EVENT, AR-DA, STRING-DA, BIT-DA, bit-da(4), CPLX-DA, FLT-DA\}.

11.5.1.6 Testing of arguments of builtin functions

During the evaluation of arguments of builtin functions by the general mechanism described in section 11.2, some general tests are performed. These tests supplement the tests performed automatically by the instructions \text{convert}, \text{eval-exp}, \text{eval-ref-gen}, \text{builtin-arg-opl-1}, etc. and the special tests performed by the individual instructions of the builtin functions.

These tests are specified by the following elementary objects contained in the seventh column of the table:

\begin{itemize}
  \item REAL The converted operand has to be of real mode.
  \item ARR The argument has to be an array expression.
  \item LIN-ARR The argument has to be a one dimensional array expression.
  \item LIN The argument has to be a one dimensional array expression or a scalar expression.
  \item INTG The argument has to be an unsigned integer constant.
  \item S-INTG The argument has to be a possibly signed integer constant.
  \item SIMP-REF The argument has to be a simple reference (i.e., unqualified and unsubscripted).
  \item T The argument may be any expression (no test is performed).
\end{itemize}

This column defines the function

(33) \text{arg-test}(id,i)

for: \text{is-id}(id), \text{is-intg-val}(i)

Note: The range of this function is the set of elementary objects \{REAL, ARR, LIN-ARR, LIN, INTG, S-INTG, SIMP-REF, T\}.
11.5.1.7 Return types of built-in functions

The last but two column of the table gives for each built-in function an expression which determines its return type, i.e. the data attributes of its result. This expression depends on the data attributes or values of the arguments of the built-in function (cf. 11.5.1.3).

This column defines the function

(39) \texttt{rt-builtin}(id, list)

for: \texttt{is-id}(id), \texttt{is-builtin-arg-descr-list}(list),
\texttt{is-builtin-arg-descr} = \texttt{is-eda} \lor \texttt{is-int-val} \lor \texttt{is-\_}

Ref.: char-da 8-13 (60), bit-da 8-13 (61), BINTG-DA 8-12 (53), new 11-30 (41),
\texttt{pref} 11-32 (45), \texttt{prec} 11-31 (42), new-1 11-31 (44), \texttt{pref-1} 11-33 (48),
\texttt{ID-CHAR-DA} 11-33 (52), \texttt{rt-abs} 11-34 (56), \texttt{rt-mod} 11-35 (63),
\texttt{rt-trunc} 11-37 (71), \texttt{rt-repeat} 11-44 (132), \texttt{rt-unspec} 11-44 (134),
\texttt{rt-round} 11-52 (174), \texttt{rt-string} 11-52 (176), ONSOURCE-DA 9-17 (35),
\texttt{ONKEY-DA} 11-48 (153), \texttt{rt-fixed} 11-36 (66).

Note: The range of this function consists of evaluated scalar data attributes, i.e. objects satisfying the predicate \texttt{is-scalar-eda}. Its argument list consists of the descriptors of the single arguments of the built-in function: for classes \texttt{EXPND} and \texttt{AGGR} generally the scalar data attributes, for class \texttt{CONST} the integer value and for class \texttt{IRREL} the object \* (cf. 11.5.1.3).

11.5.1.8 Evaluation of the individual built-in functions

The last but one column of the table lists the instructions, which after evaluation of the arguments evaluate the values of the individual built-in functions (cf. 11.2). These instructions are listed in section 11.6, the last column of the table gives the references to their definitions. Arguments of these instructions are the evaluated arguments of the built-in function reference (cf. 11.5.1.4) and in case of type \texttt{TEXT} the environment necessary to interpret the argument. For the built-in functions \texttt{MAX} and \texttt{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 the data attributes defined by the function \texttt{rt-builtin} (cf. 11.5.17).

The last but one column defines the instruction
(40) **eval-builtin-2(id, list, env)**

for:  is-id(id), is-builtin-arg-list(list), is-e(env),

is-builtin-arg = is-op v is-op-list v is-intg-val v is-eda v

is-gen v is-expr

Note: The instruction returns an operand, i.e., an object satisfying is-op.

### 11.5.2 The Table

**Abbreviations**

- **op**<sub>i</sub> = elem(i, list), for i = 1, 2, 3
  - used if the i-th element of list is an operand
- **opl**<sub>i</sub> = elem(i, list), for i = 1, 2
  - used if the i-th element of list is an operand list
- **k**<sub>i</sub> = elem(i, list), for i = 1, 2, 3, 4
  - used if the i-th element of list is an integer value
- **eda**<sub>1</sub> = elem(1, list)
  - used if the first element of list consists of the evaluated data attributes of an aggregate
- **gen**<sub>i</sub> = elem(i, list), for i = 1, 2
  - used if the i-th element of list is a generation
- **t**<sub>1</sub> = elem(1, list)
  - used if the first element of list is the text of an expression
- **opl**<sub>0</sub> = list
  - used for MAX and MIN if list is an operand list
- **da**<sub>i</sub> = elem(i, list), for i = 1, 2
  - used if the i-th element of list consists of evaluated scalar data attributes
- **da-list**<sub>0</sub> = list
  - used for MAX and MIN if the elements of list consist of scalar data attributes
<table>
<thead>
<tr>
<th>id</th>
<th>i</th>
<th>lmin(id)</th>
<th>arg-class(id,i)</th>
<th>arg-type(id,i)</th>
<th>arg-da(id,i)</th>
<th>arg-test(id,i)</th>
</tr>
</thead>
<tbody>
<tr>
<td>ABS</td>
<td>1</td>
<td>1</td>
<td>EXPND</td>
<td>OP</td>
<td>AR-DA</td>
<td>T</td>
</tr>
<tr>
<td>MAX</td>
<td>1</td>
<td>2</td>
<td>EXPND</td>
<td>OP</td>
<td>AR-DA</td>
<td>REAL</td>
</tr>
<tr>
<td>MIN</td>
<td>1</td>
<td>2</td>
<td>EXPND</td>
<td>OP</td>
<td>AR-DA</td>
<td>REAL</td>
</tr>
<tr>
<td>MOD</td>
<td>1</td>
<td>2</td>
<td>EXPND</td>
<td>OP</td>
<td>AR-DA</td>
<td>REAL</td>
</tr>
<tr>
<td>SIGN</td>
<td>1</td>
<td>1</td>
<td>EXPND</td>
<td>OP</td>
<td>AR-DA</td>
<td>REAL</td>
</tr>
<tr>
<td>FIXED</td>
<td>1</td>
<td>1</td>
<td>EXPND</td>
<td>OP</td>
<td>AR-DA</td>
<td>T</td>
</tr>
<tr>
<td>FLOAT</td>
<td>2</td>
<td>1</td>
<td>EXPND</td>
<td>OP</td>
<td>AR-DA</td>
<td>INTG</td>
</tr>
<tr>
<td>FLOOR</td>
<td>1</td>
<td>1</td>
<td>EXPND</td>
<td>OP</td>
<td>AR-DA</td>
<td>REAL</td>
</tr>
<tr>
<td>CEIL</td>
<td>1</td>
<td>1</td>
<td>EXPND</td>
<td>OP</td>
<td>AR-DA</td>
<td>REAL</td>
</tr>
<tr>
<td>TRUNC</td>
<td>1</td>
<td>1</td>
<td>EXPND</td>
<td>OP</td>
<td>AR-DA</td>
<td>REAL</td>
</tr>
<tr>
<td>BINARY</td>
<td>1</td>
<td>3</td>
<td>EXPND</td>
<td>OP</td>
<td>AR-DA</td>
<td>T</td>
</tr>
<tr>
<td>BINF</td>
<td>2</td>
<td>1</td>
<td>EXPND</td>
<td>OP</td>
<td>AR-DA</td>
<td>INTG</td>
</tr>
<tr>
<td>DEC</td>
<td>1</td>
<td>1</td>
<td>EXPND</td>
<td>OP</td>
<td>AR-DA</td>
<td>T</td>
</tr>
<tr>
<td>DEC</td>
<td>2</td>
<td>1</td>
<td>EXPND</td>
<td>OP</td>
<td>AR-DA</td>
<td>INTG</td>
</tr>
<tr>
<td>PRECISION</td>
<td>1</td>
<td>2</td>
<td>EXPND</td>
<td>OP</td>
<td>AR-DA</td>
<td>T</td>
</tr>
<tr>
<td>PREC</td>
<td>2</td>
<td>1</td>
<td>EXPND</td>
<td>OP</td>
<td>AR-DA</td>
<td>INTG</td>
</tr>
<tr>
<td>ADD</td>
<td>1</td>
<td>3</td>
<td>EXPND</td>
<td>OP</td>
<td>AR-DA</td>
<td>T</td>
</tr>
<tr>
<td>ADD</td>
<td>2</td>
<td>3</td>
<td>EXPND</td>
<td>OP</td>
<td>AR-DA</td>
<td>INTG</td>
</tr>
<tr>
<td>MULTIPY</td>
<td>1</td>
<td>3</td>
<td>EXPND</td>
<td>OP</td>
<td>AR-DA</td>
<td>T</td>
</tr>
<tr>
<td>DIVIDE</td>
<td>1</td>
<td>3</td>
<td>EXPND</td>
<td>OP</td>
<td>AR-DA</td>
<td>T</td>
</tr>
<tr>
<td>COMPLEX</td>
<td>1</td>
<td>2</td>
<td>EXPND</td>
<td>OP</td>
<td>AR-DA</td>
<td>REAL</td>
</tr>
<tr>
<td>REAL</td>
<td>1</td>
<td>1</td>
<td>EXPND</td>
<td>OP</td>
<td>CPLX-DA</td>
<td>T</td>
</tr>
<tr>
<td>IMAG</td>
<td>1</td>
<td>1</td>
<td>EXPND</td>
<td>OP</td>
<td>CPLX-DA</td>
<td>T</td>
</tr>
<tr>
<td>id</td>
<td>rt-builtin(id,list)</td>
<td>eval-builtin-2(id,list,env)</td>
<td>Ref</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>ABS</td>
<td>rt-abs(da₁)</td>
<td>eval-abs(op₁)</td>
<td>11-34(54)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MAX</td>
<td>pref-1(Ω,da-list₀)</td>
<td>eval-max(op₁₀)</td>
<td>11-34(57)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MIN</td>
<td>pref-1(Ω,da-list₀)</td>
<td>eval-min(op₁₀)</td>
<td>11-35(59)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MOD</td>
<td>rt-mod(da₁,da₂)</td>
<td>eval-mod(op₁,op₂)</td>
<td>11-35(61)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SIGN</td>
<td>BINTG-DA</td>
<td>eval-sign(op₁)</td>
<td>11-36(64)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FIXED</td>
<td>rt-fixed(da₁,k₂,k₃)</td>
<td>eval-fixed(op₁,k₂,k₃)</td>
<td>11-36(65)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FLOAT</td>
<td>prec(new(FLT,da₁),k₂,Ω)</td>
<td>eval-float(op₁,k₂)</td>
<td>11-36(67)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FLOOR</td>
<td>rt-trunc(da₁)</td>
<td>eval-floor(op₁)</td>
<td>11-36(68)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CEIL</td>
<td>rt-trunc(da₁)</td>
<td>eval-ceil(op₁)</td>
<td>11-36(69)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TRUNC</td>
<td>rt-trunc(da₁)</td>
<td>eval-trunc(op₁)</td>
<td>11-36(70)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BINARY</td>
<td>}</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BIN</td>
<td>new-1(BIN,da₁,k₂,k₃)</td>
<td>eval-bin(op₁,k₂,k₃)</td>
<td>11-37(72)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DECIMAL</td>
<td>}</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DEC</td>
<td>new-1(DEC,da₁,k₂,k₃)</td>
<td>eval-dec(op₁,k₂,k₃)</td>
<td>11-37(73)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PRECISION</td>
<td>}</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PREC</td>
<td>prec(da₁,k₂,k₃)</td>
<td>eval-prec(op₁,k₂,k₃)</td>
<td>11-37(74)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ADD</td>
<td>prec(pref(da₁,da₂),k₃,k₄)</td>
<td>eval-add(op₁,op₂,k₃,k₄)</td>
<td>11-37(75)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MULTIPLY</td>
<td>prec(pref(da₁,da₂),k₃,k₄)</td>
<td>eval-multiply(op₁,op₂,k₃,k₄)</td>
<td>11-37(76)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DIVIDE</td>
<td>prec(pref(da₁,da₂),k₃,k₄)</td>
<td>eval-divide(op₁,op₂,k₃,k₄)</td>
<td>11-37(77)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>COMPLEX</td>
<td>}</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CPLX</td>
<td>new(CPLX,pref(da₁,da₂))</td>
<td>eval-cplx(op₁,op₂)</td>
<td>11-38(79)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>REAL</td>
<td>new(REAL,da₁)</td>
<td>eval-real(op₁)</td>
<td>11-38(81)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IMAG</td>
<td>new(REAL,da₁)</td>
<td>eval-imag(op₁)</td>
<td>11-38(82)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>id</td>
<td>i</td>
<td>lmin(id)</td>
<td>arg-class(id,i)</td>
<td>arg-type(id,i)</td>
<td>arg-da(id,i)</td>
<td>arg-test(id,i)</td>
</tr>
<tr>
<td>-------</td>
<td>---</td>
<td>----------</td>
<td>-----------------</td>
<td>----------------</td>
<td>--------------</td>
<td>----------------</td>
</tr>
<tr>
<td>CONJG</td>
<td>1</td>
<td>1</td>
<td>EXPND</td>
<td>OP</td>
<td>CPLX-DA</td>
<td>T</td>
</tr>
<tr>
<td>EXP</td>
<td>1</td>
<td>1</td>
<td>EXPND</td>
<td>OP</td>
<td>FLT-DA</td>
<td>T</td>
</tr>
<tr>
<td>LOG</td>
<td>1</td>
<td>1</td>
<td>EXPND</td>
<td>OP</td>
<td>FLT-DA</td>
<td>T</td>
</tr>
<tr>
<td>LOG 10</td>
<td>1</td>
<td>1</td>
<td>EXPND</td>
<td>OP</td>
<td>FLT-DA</td>
<td>REAL</td>
</tr>
<tr>
<td>LOG 2</td>
<td>1</td>
<td>1</td>
<td>EXPND</td>
<td>OP</td>
<td>FLT-DA</td>
<td>REAL</td>
</tr>
<tr>
<td>TAND</td>
<td>1</td>
<td>1</td>
<td>EXPND</td>
<td>OP</td>
<td>FLT-DA</td>
<td>T</td>
</tr>
<tr>
<td>SIN</td>
<td>1</td>
<td>1</td>
<td>EXPND</td>
<td>OP</td>
<td>FLT-DA</td>
<td>T</td>
</tr>
<tr>
<td>COSD</td>
<td>1</td>
<td>1</td>
<td>EXPND</td>
<td>OP</td>
<td>FLT-DA</td>
<td>REAL</td>
</tr>
<tr>
<td>COS</td>
<td>1</td>
<td>1</td>
<td>EXPND</td>
<td>OP</td>
<td>FLT-DA</td>
<td>T</td>
</tr>
<tr>
<td>TANH</td>
<td>1</td>
<td>1</td>
<td>EXPND</td>
<td>OP</td>
<td>FLT-DA</td>
<td>T</td>
</tr>
<tr>
<td>ERF</td>
<td>1</td>
<td>1</td>
<td>EXPND</td>
<td>OP</td>
<td>FLT-DA</td>
<td>REAL</td>
</tr>
<tr>
<td>SQRT</td>
<td>1</td>
<td>1</td>
<td>EXPND</td>
<td>OP</td>
<td>FLT-DA</td>
<td>T</td>
</tr>
<tr>
<td>ERFC</td>
<td>1</td>
<td>1</td>
<td>EXPND</td>
<td>OP</td>
<td>FLT-DA</td>
<td>REAL</td>
</tr>
<tr>
<td>COSH</td>
<td>1</td>
<td>1</td>
<td>EXPND</td>
<td>OP</td>
<td>FLT-DA</td>
<td>T</td>
</tr>
<tr>
<td>SINH</td>
<td>1</td>
<td>1</td>
<td>EXPND</td>
<td>OP</td>
<td>FLT-DA</td>
<td>T</td>
</tr>
<tr>
<td>ATANH</td>
<td>1</td>
<td>1</td>
<td>EXPND</td>
<td>OP</td>
<td>FLT-DA</td>
<td>T</td>
</tr>
<tr>
<td>ATAND</td>
<td>1</td>
<td>1</td>
<td>EXPND</td>
<td>OP</td>
<td>FLT-DA</td>
<td>REAL</td>
</tr>
<tr>
<td>ATAN</td>
<td>2</td>
<td>1</td>
<td>EXPND</td>
<td>OP</td>
<td>FLT-DA</td>
<td>REAL</td>
</tr>
<tr>
<td>BIT</td>
<td>1</td>
<td>2</td>
<td>EXPND</td>
<td>CONST</td>
<td>STRING-DA</td>
<td>T</td>
</tr>
<tr>
<td>CHAR</td>
<td>1</td>
<td>2</td>
<td>EXPND</td>
<td>CONST</td>
<td>STRING-DA</td>
<td>T</td>
</tr>
<tr>
<td>SUBSTR</td>
<td>1</td>
<td>2</td>
<td>EXPND</td>
<td>TEXT</td>
<td>STRING-DA</td>
<td>T</td>
</tr>
<tr>
<td>INDEX</td>
<td>1</td>
<td>2</td>
<td>EXPND</td>
<td>OP</td>
<td>STRING-DA</td>
<td>T</td>
</tr>
<tr>
<td>LENGTH</td>
<td>1</td>
<td>1</td>
<td>EXPND</td>
<td>OP</td>
<td>STRING-DA</td>
<td>T</td>
</tr>
</tbody>
</table>

11.5.2
<table>
<thead>
<tr>
<th>id</th>
<th>rt-builtin(id,list)</th>
<th>eval-builtin-2(id,list,env)</th>
<th>Ref</th>
</tr>
</thead>
<tbody>
<tr>
<td>CONJG</td>
<td>( d_1 )</td>
<td>( \text{eval-conjg}(\text{op}_1) )</td>
<td>11-38(83)</td>
</tr>
<tr>
<td>EXP</td>
<td>( d_1 )</td>
<td>( \text{eval-exp}(\text{op}_1) )</td>
<td>11-39(86)</td>
</tr>
<tr>
<td>LOG</td>
<td>( d_1 )</td>
<td>( \text{eval-log}(\text{op}_1) )</td>
<td>11-40(113)</td>
</tr>
<tr>
<td>LOG 10</td>
<td>( d_1 )</td>
<td>( \text{eval-log-10}(\text{op}_1) )</td>
<td>11-40(114)</td>
</tr>
<tr>
<td>LOG 2</td>
<td>( d_1 )</td>
<td>( \text{eval-log-2}(\text{op}_1) )</td>
<td>11-40(115)</td>
</tr>
<tr>
<td>TAND</td>
<td>( d_1 )</td>
<td>( \text{eval-tand}(\text{op}_1) )</td>
<td>11-39(91)</td>
</tr>
<tr>
<td>TAN</td>
<td>( d_1 )</td>
<td>( \text{eval-tan}(\text{op}_1) )</td>
<td>11-39(92)</td>
</tr>
<tr>
<td>SIND</td>
<td>( d_1 )</td>
<td>( \text{eval-sind}(\text{op}_1) )</td>
<td>11-39(93)</td>
</tr>
<tr>
<td>SIN</td>
<td>( d_1 )</td>
<td>( \text{eval-sin}(\text{op}_1) )</td>
<td>11-39(94)</td>
</tr>
<tr>
<td>COSD</td>
<td>( d_1 )</td>
<td>( \text{eval-cosd}(\text{op}_1) )</td>
<td>11-39(95)</td>
</tr>
<tr>
<td>COS</td>
<td>( d_1 )</td>
<td>( \text{eval-cos}(\text{op}_1) )</td>
<td>11-39(96)</td>
</tr>
<tr>
<td>TANH</td>
<td>( d_1 )</td>
<td>( \text{eval-tanh}(\text{op}_1) )</td>
<td>11-39(97)</td>
</tr>
<tr>
<td>ERF</td>
<td>( d_1 )</td>
<td>( \text{eval-erf}(\text{op}_1) )</td>
<td>11-39(98)</td>
</tr>
<tr>
<td>SQRT</td>
<td>( d_1 )</td>
<td>( \text{eval-sqrt}(\text{op}_1) )</td>
<td>11-40(116)</td>
</tr>
<tr>
<td>ERFC</td>
<td>( d_1 )</td>
<td>( \text{eval-erfc}(\text{op}_1) )</td>
<td>11-39(101)</td>
</tr>
<tr>
<td>COSH</td>
<td>( d_1 )</td>
<td>( \text{eval-cosh}(\text{op}_1) )</td>
<td>11-39(102)</td>
</tr>
<tr>
<td>SINGH</td>
<td>( d_1 )</td>
<td>( \text{eval-sinh}(\text{op}_1) )</td>
<td>11-39(103)</td>
</tr>
<tr>
<td>ATANH</td>
<td>( d_1 )</td>
<td>( \text{eval-atanh}(\text{op}_1) )</td>
<td>11-41(117)</td>
</tr>
<tr>
<td>ATAND</td>
<td>( \text{pref}(d_1,d_2) )</td>
<td>( \text{eval-atand}(\text{op}_1,\text{op}_2) )</td>
<td>11-41(118)</td>
</tr>
<tr>
<td>ATAN</td>
<td>( \text{pref}(d_1,d_2) )</td>
<td>( \text{eval-atan}(\text{op}_1,\text{op}_2) )</td>
<td>11-41(119)</td>
</tr>
<tr>
<td>BIT</td>
<td>( \text{new-1}(\text{BIT},d_1,k_2,Q) )</td>
<td>( \text{eval-bit}(\text{op}_1,k_2) )</td>
<td>11-42(120)</td>
</tr>
<tr>
<td>CHAR</td>
<td>( \text{new-1}(\text{CHAR},d_1,k_2,Q) )</td>
<td>( \text{eval-char}(\text{op}_1,k_2) )</td>
<td>11-42(121)</td>
</tr>
<tr>
<td>SUBSTR</td>
<td>( \text{new}(\text{VARYING},d_1) )</td>
<td>( \text{eval-substr}(t_1,k_2,k_3,\text{env}) )</td>
<td>11-42(122)</td>
</tr>
<tr>
<td>INDEX</td>
<td>BINTG-DA</td>
<td>( \text{eval-index}(\text{op}_1,\text{op}_2) )</td>
<td>11-43(125)</td>
</tr>
<tr>
<td>LENGTH</td>
<td>BINTG-DA</td>
<td>( \text{eval-length}(\text{op}_1) )</td>
<td>11-43(127)</td>
</tr>
<tr>
<td>id</td>
<td>i</td>
<td>lmin(id)</td>
<td>arg-class(id,i)</td>
</tr>
<tr>
<td>-------</td>
<td>---</td>
<td>----------</td>
<td>-----------------</td>
</tr>
<tr>
<td>HIGH</td>
<td>1</td>
<td>1</td>
<td>CONST</td>
</tr>
<tr>
<td>LOW</td>
<td>1</td>
<td>1</td>
<td>CONST</td>
</tr>
<tr>
<td>REPEAT</td>
<td>1</td>
<td>2</td>
<td>EXPND</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td></td>
<td>CONST</td>
</tr>
<tr>
<td>UNSPEC</td>
<td>1</td>
<td>1</td>
<td>EXPND</td>
</tr>
<tr>
<td>BOOL</td>
<td>1</td>
<td>3</td>
<td>EXPND</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td></td>
<td>EXPND</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td></td>
<td>EXPND</td>
</tr>
<tr>
<td>SUM</td>
<td>1</td>
<td>1</td>
<td>AGGR</td>
</tr>
<tr>
<td>PROD</td>
<td>1</td>
<td>1</td>
<td>AGGR</td>
</tr>
<tr>
<td>ALL</td>
<td>1</td>
<td>1</td>
<td>AGGR</td>
</tr>
<tr>
<td>ANY</td>
<td>1</td>
<td>1</td>
<td>AGGR</td>
</tr>
<tr>
<td>POLY</td>
<td>1</td>
<td>2</td>
<td>AGGR</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td></td>
<td>AGGR</td>
</tr>
<tr>
<td>LBOUND</td>
<td>1</td>
<td>2</td>
<td>IRREL</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td></td>
<td>IRREL</td>
</tr>
<tr>
<td>HBOUND</td>
<td>1</td>
<td>2</td>
<td>IRREL</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td></td>
<td>IRREL</td>
</tr>
<tr>
<td>DIM</td>
<td>1</td>
<td>2</td>
<td>IRREL</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td></td>
<td>IRREL</td>
</tr>
<tr>
<td>ONFILE</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ONLOC</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ONSOURCE</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ONCHAR</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ONKEY</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ONCODE</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DATAFIELD</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ONCOUNT</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ADDR</td>
<td>1</td>
<td>1</td>
<td>IRREL</td>
</tr>
<tr>
<td>POINTER}</td>
<td>1</td>
<td>2</td>
<td>EXPND</td>
</tr>
<tr>
<td>PTR</td>
<td>2</td>
<td></td>
<td>EXPND</td>
</tr>
<tr>
<td>OFFSET</td>
<td>1</td>
<td>2</td>
<td>EXPND</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td></td>
<td>EXPND</td>
</tr>
<tr>
<td>id</td>
<td>rt-builtin(id,list)</td>
<td>eval-builtin-2(id,list,env)</td>
<td>Ref</td>
</tr>
<tr>
<td>---------</td>
<td>---------------------</td>
<td>-----------------------------</td>
<td>---------</td>
</tr>
<tr>
<td>HIGH</td>
<td>char-da(k₁)</td>
<td>eval-high(k₁)</td>
<td>11-43(128)</td>
</tr>
<tr>
<td>LOW</td>
<td>char-da(k₁)</td>
<td>eval-low(k₁)</td>
<td>11-43(129)</td>
</tr>
<tr>
<td>REPEAT</td>
<td>rt-repeat(da₁,k₂)</td>
<td>eval-repeat(op₁,k₂)</td>
<td>11-44(130)</td>
</tr>
<tr>
<td>UNSPEC</td>
<td>rt-unspec(da₁)</td>
<td>eval-unspec(gen₁)</td>
<td>11-44(133)</td>
</tr>
<tr>
<td>BOOL</td>
<td>pref(da₁,da₂)</td>
<td>eval-bool(op₁,op₂,op₃)</td>
<td>11-44(135)</td>
</tr>
<tr>
<td>SUM</td>
<td>da₁</td>
<td>eval-sum(op₁)</td>
<td>11-45(138)</td>
</tr>
<tr>
<td>PROD</td>
<td>da₁</td>
<td>eval-prod(op₁)</td>
<td>11-45(139)</td>
</tr>
<tr>
<td>ALL</td>
<td>da₁</td>
<td>eval-all(op₁)</td>
<td>11-45(140)</td>
</tr>
<tr>
<td>ANY</td>
<td>da₁</td>
<td>eval-any(op₁)</td>
<td>11-45(141)</td>
</tr>
<tr>
<td>POLY</td>
<td>pref(da₁,da₂)</td>
<td>eval-poly(op₁,op₂)</td>
<td>11-46(143)</td>
</tr>
<tr>
<td>LBOUND</td>
<td>BINTG-DA</td>
<td>eval-lbound(eda₁,k₂)</td>
<td>11-46(145)</td>
</tr>
<tr>
<td>HBOUND</td>
<td>BINTG-DA</td>
<td>eval-hbound(eda₁,k₂)</td>
<td>11-46(146)</td>
</tr>
<tr>
<td>DIM</td>
<td>BINTG-DA</td>
<td>eval-dim(eda₁,k₂)</td>
<td>11-47(147)</td>
</tr>
<tr>
<td>ONFILE</td>
<td>ID-CHAR-DA</td>
<td>eval-onfile</td>
<td>11-47(148)</td>
</tr>
<tr>
<td>ONLOC</td>
<td>ID-CHAR-DA</td>
<td>eval-onloc</td>
<td>11-47(149)</td>
</tr>
<tr>
<td>ONSOURCE</td>
<td>ONSOURCE-DA</td>
<td>eval-onsource</td>
<td>11-47(150)</td>
</tr>
<tr>
<td>ONCHAR</td>
<td>char-da(l)</td>
<td>eval-onchar</td>
<td>11-47(151)</td>
</tr>
<tr>
<td>ONKEY</td>
<td>ONKEY-DA</td>
<td>eval-onkey</td>
<td>11-48(152)</td>
</tr>
<tr>
<td>ONCODE</td>
<td>BINTG-DA</td>
<td>eval-oncode</td>
<td>11-48(154)</td>
</tr>
<tr>
<td>DATAFIELD</td>
<td>ID-CHAR-DA</td>
<td>eval-datafield</td>
<td>11-48(155)</td>
</tr>
<tr>
<td>ONCOUNT</td>
<td>BINTG-DA</td>
<td>eval-oncount</td>
<td>11-48(156)</td>
</tr>
<tr>
<td>ADDR</td>
<td>PTR</td>
<td>eval-addr(gen₁)</td>
<td>11-48(157)</td>
</tr>
<tr>
<td>POINTER</td>
<td>PTR</td>
<td>eval-ptr(op₁,gen₂)</td>
<td>11-49(158)</td>
</tr>
<tr>
<td>OFFSET</td>
<td>OFFSET</td>
<td>eval-offset(op₁,gen₂)</td>
<td>11-49(159)</td>
</tr>
</tbody>
</table>

11.5.2
<table>
<thead>
<tr>
<th>id</th>
<th>i</th>
<th>lmin(id)</th>
<th>arg-class(id,i)</th>
<th>arg-type(id,i)</th>
<th>arg-da(id,i)</th>
<th>arg-test(id,i)</th>
</tr>
</thead>
<tbody>
<tr>
<td>NULL</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NULLO</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>EMPTY</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DATE</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TIME</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ALLOCATION</td>
<td>1</td>
<td>1</td>
<td>IRREL</td>
<td>TEXT</td>
<td>*</td>
<td>SIMP-REF</td>
</tr>
<tr>
<td>LINENO</td>
<td>1</td>
<td>1</td>
<td>IRREL</td>
<td>TEXT</td>
<td>*</td>
<td>SIMP-REF</td>
</tr>
<tr>
<td>COUNT</td>
<td>1</td>
<td>1</td>
<td>IRREL</td>
<td>TEXT</td>
<td>*</td>
<td>SIMP-REF</td>
</tr>
<tr>
<td>ROUND</td>
<td>1</td>
<td>2</td>
<td>EXPND</td>
<td>TEXT</td>
<td>*</td>
<td>S-INTG</td>
</tr>
<tr>
<td>STRING</td>
<td>1</td>
<td>1</td>
<td>AGGR</td>
<td>GEN</td>
<td>*</td>
<td>T</td>
</tr>
<tr>
<td>STATUS</td>
<td>1</td>
<td>1</td>
<td>EXPND</td>
<td>OP</td>
<td>EVENT</td>
<td>T</td>
</tr>
<tr>
<td>COMPLETION</td>
<td>1</td>
<td>1</td>
<td>EXPND</td>
<td>OP</td>
<td>EVENT</td>
<td>T</td>
</tr>
<tr>
<td>PRIORITY</td>
<td>1</td>
<td>1</td>
<td>EXPND</td>
<td>OP</td>
<td>TASK</td>
<td>T</td>
</tr>
<tr>
<td>T</td>
<td>1</td>
<td>Ω</td>
<td>Ω</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

11.5.2
<table>
<thead>
<tr>
<th>id</th>
<th>rt-builtin(id,list)</th>
<th>eval-builtin-2(id,list,env)</th>
<th>Ref</th>
</tr>
</thead>
<tbody>
<tr>
<td>NULL</td>
<td>PTR</td>
<td>eval-null</td>
<td>11-49(160)</td>
</tr>
<tr>
<td>NULLO</td>
<td>OFFSET</td>
<td>eval-nullo</td>
<td>11-49(161)</td>
</tr>
<tr>
<td>EMPTY</td>
<td>μ₀(«s-size:0»)</td>
<td>eval-empty</td>
<td>11-49(163)</td>
</tr>
<tr>
<td>DATE</td>
<td>char-da(5)</td>
<td>eval-date</td>
<td>11-69(164)</td>
</tr>
<tr>
<td>TIME</td>
<td>char-da(9)</td>
<td>eval-time</td>
<td>11-50(166)</td>
</tr>
<tr>
<td>ALLOCATION</td>
<td>bit-da(1)</td>
<td>eval-allocation(t₁,env)</td>
<td>11-50(167)</td>
</tr>
<tr>
<td>LINENO</td>
<td>BINTG-DA</td>
<td>eval-lineno(t₁,env)</td>
<td>11-50(168)</td>
</tr>
<tr>
<td>COUNT</td>
<td>BINTG-DA</td>
<td>eval-count(t₁,env)</td>
<td>11-51(169)</td>
</tr>
<tr>
<td>ROUND</td>
<td>rt-round(da₁)</td>
<td>eval-round(t₁,k₂,env)</td>
<td>11-51(171)</td>
</tr>
<tr>
<td>STRING</td>
<td>rt-string(eda₁)</td>
<td>eval-string(ena₁)</td>
<td>11-52(175)</td>
</tr>
<tr>
<td>STATUS</td>
<td>BINTG-DA</td>
<td>eval-status(op₁)</td>
<td>11-53(177)</td>
</tr>
<tr>
<td>COMPLETION</td>
<td>bit-da(1)</td>
<td>eval-completion(op₁)</td>
<td>11-53(178)</td>
</tr>
<tr>
<td>CPLN</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PRIORITY</td>
<td>PRI-DA</td>
<td>eval-priority(op₁)</td>
<td>11-53(179)</td>
</tr>
</tbody>
</table>
11.6 Evaluation of the Individual Builtin Functions

Section 11.6.2 defines the instructions listed in the last but one column of the table given in section 11.5.2. These instructions evaluate the individual builtin functions.

Section 11.6.1 defines some general auxiliary functions, instructions, and objects used in 11.6.2 and especially the functions used in the last but two column of the table for determination of the return types.

Metavariables

\begin{align*}
\text{op} & \quad \text{is-op} \\
\text{opl} & \quad \text{is-op-list} \\
\text{da} & \quad \text{is-scalar-eda} \\
\text{eda} & \quad \text{is-eda} \\
p & \quad \text{is-intg-val} \lor \text{is-\( \varnothing \)} \quad \text{a precision (or string length), if specified} \\
q & \quad \text{is-intg-val} \lor \text{is-\( \varnothing \)} \quad \text{a scale factor, if specified} \\
i,j,k & \quad \text{is-intg-val} \lor \text{is-\( \varnothing \)} \quad \text{an integer value, sometimes optional} \\
t & \quad \text{is-expr} \\
\text{env} & \quad \text{is-e} \\
\text{gen} & \quad \text{is-gen}
\end{align*}

11.6.1 General auxiliary definitions

\begin{align*}
(41) \quad \text{new}(x, da) = \\
\begin{array}{ll}
x \in \{ \text{REAL,CPLX} \} & \quad \mu(da;<s\text{-mode}:x>) \\
x \in \{ \text{BIN,DEC,BIT,CHAR} \} & \quad \mu(da;<s\text{-base}:x>) \\
x \in \{ \text{FIX,FLT} \} & \quad \mu(da;<s\text{-scale}:x>) \\
\text{is-VARYING}(x) & \quad \mu(da;<s\text{-varying}:T>) 
\end{array}
\end{align*}

Note: This instruction is used to change a single attribute of \( da \); it is never used to change from arithmetic to string or vice versa.
\[ \text{prec}(da,p,q) = \]
\[
\begin{align*}
\text{is-FIX\#s-scale}(da) & \& \text{is-intg-val}(p) & \& \text{is-intg-val}(q) & \& p \leq \text{max-prec}(da) \rightarrow \\
\mu(da;\{s-prec:p\};\{s-scale-f:q\}) \\
\text{is-FIX\#s-scale}(da) \& \text{is-\Ø}(p) \& \text{is-\Ø}(q) \rightarrow \\
\mu(da;\{s-prec: \text{def-prec}(da)\};\{s-scale-f:0\}) \\
\text{is-FLT\#s-scale}(da) \& \text{is-intg-val}(p) \& \text{is-\Ø}(q) \& p \leq \text{max-prec}(da) \rightarrow \\
\mu(da;\{s-prec:p\}) \\
\text{is-FLT\#s-scale}(da) \& \text{is-\Ø}(p) \& \text{is-\Ø}(q) \rightarrow \\
\mu(da;\{s-prec: \text{def-prec}(da)\}) \\
T \rightarrow \text{error}
\end{align*}
\]

Ref.: max-prec 8-13(62)

Note: This function changes the precision and scale factor of arithmetic data attributes to either specified or default values.

\[ \text{def-prec}(da) = \text{def-prec-1}(s-base(da), s-scale(da)) \]

This definition expresses that def-prec(da), the default precision associated with da, depends only on the base and scale of da. The four possible values of def-prec-1 are implementation-defined integers. For fixed scale they are DEF-PREC-DEC and DEF-PREC-BIN (cf. 8-12(51/52)).

\[ \text{new-1}(base,da,p,q) = \]
\[
\begin{align*}
\text{is-\Ø}(p) \& \text{is-\Ø}(q) \rightarrow \text{complete-tg}(da_base,da) \\
\text{is-arithm}(da) \rightarrow \text{prec}(\text{new}(base,da)p,q) \\
\text{is-string}(da) \rightarrow \mu(\text{new}(base,da);\{s-length:p\})
\end{align*}
\]

where: \[ da_base = \{ \text{base} \in \{ \text{BIN, DEC} \} \rightarrow \mu(\text{AR-DA};\{s-base:base\}), \]
\[ \{ \text{BIT, CHAR} \} \rightarrow \mu(\text{STRING-DA};\{s-base:base\}) \]

for: \[ \text{base} \in \{ \text{BIN, DEC, BIT, CHAR} \} \]

Ref.: complete-tg 8-37(143), AR-DA 8-13(56), STRING-DA 8-13(57)

Note: This definition changes base, precision, and scale factor or base and length of da. In the contrary to the function prec, precision, scale-factor and length are set to those resulting from conversion rules instead of implementation-defined constants, if not specified. Also this function is never used to change from arithmetic to string type or vice versa.
(45) \( \text{pref}(\text{da}-1,\text{da}-2) = \)

\[ \begin{align*}
\text{is-} & \mathcal{Q}(\text{da}-2) \longrightarrow \text{da}-1 \\
\text{is-arithm(} & \text{da}-1) \text{ & is-arithm(} \text{da}-2) \longrightarrow \\
\mu(\text{pref-ar-da(} & \text{da}-1,\text{da}-2); <\text{s-prec:} \text{pref-prec(} \text{da}-1,\text{da}-2)>, \\
\text{<} & \text{s-scale-f:} \text{pref-scale-f(} \text{da}-1,\text{da}-2) >) \\
\text{is-string(} & \text{da}-1) \text{ & is-string(} \text{da}-2) \longrightarrow \\
\mu(\text{pref-str-da(} & \text{da}-1,\text{da}-2); <\text{s-length:} \max(\text{s-length(} \text{da}-1),\text{s-length(} \text{da}-2) >) \\
\text{for: is-scalar-ed(} & \text{da}-1,\text{da}-2) \vee \text{is-} \mathcal{Q}(\text{da}-2) \\
\text{Ref.: pref-ar-da 8-23(94), pref-str-da 8-24(98)}
\end{align*} \]

Note: This function completes the functions pref-ar-da and pref-str-da by specifying also the preferred precision and scale-factor or length of two data attributes. It merges two arithmetic or two string data attributes to a complete data attribute with the higher characteristics of both of them.

(46) \( \text{pref-prec(da}-1,\text{da}-2) = \)

\[ \begin{align*}
\text{is-FIX-s-scale(} & \text{da}-1) \text{ & is-FIX-s-scale(} \text{da}-2) \longrightarrow \\
\min( & N_0, \max(p_1-\text{s-prec(} \text{da}-1), p_2-\text{s-prec(} \text{da}-2)) \\
\text{max(} & p_1,p_2) \longrightarrow \max(q_1,q_2)) \\
\text{where: } p_1 & = \text{s-prec(} \text{da}-1), p_2 = \text{s-prec(} \text{da}-2) \\
q_1 & = \text{s-scale-f(} \text{da}-1), q_2 = \text{s-scale-f(} \text{da}-2) \\
N_0 & = \max \text{-prec-} \text{pref-ar-da(} \text{da}-1,\text{da}-2) \\
\text{Ref.: max-prec 8-13(62), pref-ar-da 8-23(94)}
\end{align*} \]

(47) \( \text{pref-scale-f(da}-1,\text{da}-2) = \)

\[ \begin{align*}
\text{is-FIX-s-scale(} & \text{da}-1) \text{ & is-FIX-s-scale(} \text{da}-2) \longrightarrow \\
\max( & \text{s-scale-f(} \text{da}-1), \text{s-scale-f(} \text{da}-2)) \\
\text{T} & \longrightarrow \mathcal{Q} \\
\text{for: is-arithm(} & \text{da}-1), \text{is-arithm(} \text{da}-2)
\end{align*} \]
(48) \[ \text{pref-1}(da, da\text{-list}) = \]
\[
\text{is-<} (da\text{-list}) \rightarrow da
\]
\[
T \rightarrow \text{pref-1}(\text{pref(head(da\text{-list}),da)}, \text{tail(da\text{-list}))}
\]
for: is-scalar-edda(da) ∨ is-∅(da), is-scalar-edda-list(da\text{-list})

Note: Used with da=∅, this function computes the highest characteristics of a list of data attributes.

(49) \[ \text{FLT-DA} = \]
\[
\mu(\text{AR-DA}; <\text{s-scale:FLT}>)
\]
Ref.: AR-DA 8-13(56)

(50) \[ \text{REAL-DA} = \]
\[
\mu(\text{AR-DA}; <\text{s-mode:REAL}>)
\]

(51) \[ \text{CPLX-DA} = \]
\[
\mu(\text{AR-DA}; <\text{s-mode:CPLX}>)
\]

(52) \[ \text{ID-CHAR-DA} = \]
\[
\mu_0 (<\text{s-base:CHAR}>,
\quad <\text{s-length:MAX-LENGTH-ID}>,
\quad <\text{s-varying:T}>)
\]

Note: This varying length character string data attribute with an implementation defined maximum length MAX-LENGTH-ID is used to store character strings giving the names of identifiers.

(53) \[ \text{val-op-1}(da,v) = \]
\[
\text{mk-op}(da,v_r);
\quad v_r: \text{test-rep}(da,v)
\]
for: is-value(v)
Ref.: test-rep 8-9(29), mk-op 8-11(39)

Note: This instruction constructs an operand from given data attributes and value, including testing for the size condition.
11.6.2 Evaluating instructions

In the definitions of the individual builtin functions in this section the following definitions are used frequently without being referenced each time:

new 11-30(41), prec 11-31(42), new-1 11-31(44), pref 11-32(45),
val-op-1 11-33(53), mk-op 8-11(39), op-val 8-11(40), val-op 8-11(41),
bintg-op 8-11(43), convert 8-30(118), infix-op 8-14(66), CHAR-DA 8-13(58),
BIT-DA 8-13(59), char-da 8-13(60), bit-da 8-13(61), pref-ar-da 8-23(94),
FLT-DA 11-33(49), REAL-DA 11-33(50), CPLX-DA 11-33(51), STRING-DA 8-13(57),
ID-CHAR-DA 11-33(52), complete-tg 8-37(143), chl 10-76(215), AR-DA 8-13(56).

Furthermore, the primitive arithmetic functions listed in section 1.1.8.3 are used frequently.

11.6.2.1 Arithmetic generic builtin functions

(54) \texttt{eval-abs}(op) =
    \texttt{is-REAL} \cdot \texttt{s-mode} \cdot \texttt{s-da}(op) \rightarrow \texttt{val-op-1}(s-da(op), \texttt{abs} \cdot \texttt{op-val}(op))
    \texttt{is-CPLX} \cdot \texttt{s-mode} \cdot \texttt{s-da}(op) \rightarrow \texttt{eval-cplx-abs}(op)

The instruction \texttt{eval-cplx-abs}(op) is implementation-defined. Its result is an operand \texttt{op-res} satisfying

(55) \texttt{op-res} = \texttt{val-op}(\rt-abs \cdot \texttt{s-da}(op), \texttt{v-res})

where: \texttt{is-num-val}(v-res) \& v-res \geq 0

Note: \texttt{v-res} is (an approximation of) \( + \sqrt{v_1^2 + v_2^2} \), where \( v_1 \) and \( v_2 \) are the real and imaginary part of \texttt{op-val}(op).

(56) \rt-abs(da) =
    (\texttt{is-CPLX} \cdot \texttt{s-mode} \& \texttt{is FIX} \cdot \texttt{s-scale})(da) \rightarrow \texttt{new}(\texttt{REAL}, \texttt{prec}(da, \texttt{min}(N_{da}, P_{da} + 1), Q_{da}))
    \texttt{T} \rightarrow \texttt{new}(\texttt{REAL}, \texttt{da})

where: \( N_{da} = \texttt{max-prec}(da), P_{da} = \texttt{s-prec}(da), Q_{da} = \texttt{s-scale-\{}(da) \)

(57) \texttt{eval-max}(opl) =
    \texttt{eval-max-1}(\texttt{head}(opl), \texttt{tail}(opl))
(58) \( \text{eval-max-1}(\text{op}, \text{opl}) = \)
\[
\text{is-<>}(\text{opl}) \quad \text{PASS: op} \\
\text{T} \quad \text{eval-max-1}(\text{op-1}, \text{tail}(\text{opl})); \\
\quad \text{op-1: val-op-1} (\text{pref}(\text{s-da}(\text{op}), \text{s-da}\text{head}(\text{opl})), \\
\quad \quad \text{max}(\text{op-val}(\text{op}), \text{op-val}\text{head}(\text{opl})))
\]

(59) \( \text{eval-min}(\text{opl}) = \)
\[
\quad \text{eval-min-1}(\text{head}(\text{opl}), \text{tail}(\text{opl}))
\]

(60) \( \text{eval-min-1}(\text{op}, \text{opl}) = \)
\[
\text{is-<>}(\text{opl}) \quad \text{PASS: op} \\
\text{T} \quad \text{eval-min-1}(\text{op-1}, \text{tail}(\text{opl})); \\
\quad \text{op-1: val-op-1} (\text{pref}(\text{s-da}(\text{op}), \text{s-da}\text{head}(\text{opl})), \\
\quad \quad \text{min}(\text{op-val}(\text{op}), \text{op-val}\text{head}(\text{opl})))
\]

(61) \( \text{eval-mod}(\text{op-1}, \text{op-2}) = \)
\[
\quad \text{val-op-1}(\text{da}_1, v); \\
\quad v: \text{pass-modulo-1}(\text{op-1}, \text{op-2}); \\
\quad \text{op-1}: \text{convert}(\text{da}_0, \text{op-1}), \\
\quad \text{op-2}: \text{convert}(\text{da}_0, \text{op-2})
\]

where: \( \text{da}_0 = \text{pref-ar-da}(\text{s-da}(\text{op-1}), \text{s-da}(\text{op-2})) \)

\( \text{da}_1 = \text{rt-mod}(\text{s-da}(\text{op-1}), \text{s-da}(\text{op-2})) \)

(62) \( \text{modulo-1}(\text{op-1}, \text{op-2}) = \)
\[
\quad v_2 = 0 \quad \longrightarrow \quad v_1 \\
\text{T} \quad \longrightarrow \quad \text{modulo}(v_1, \text{abs}(v_2))
\]

where: \( v_1 = \text{op-val}(\text{op-1}) \)

\( v_2 = \text{op-val}(\text{op-2}) \)

(63) \( \text{rt-mod}(\text{da-1}, \text{da-2}) = \)
\[
\quad \text{is-FLT-s-scale}(\text{da-1}) \vee \text{is-FLT-s-scale}(\text{da-2}) \quad \longrightarrow \quad \text{pref}(\text{da-1}, \text{da-2}) \\
\text{T} \quad \longrightarrow \quad \text{prec}(\text{pref}(\text{da-1}, \text{da-2}), p_o, q_o)
\]

cont'd
where: \( P_o = \min(N_0, P_2 - q_2 + q_o) \)
\( q_o = \max(q_1, q_2) \)
\( P_2 = s\text{-prec}(da-2) \)
\( q_1 = s\text{-scale-f}(da-1) \)
\( q_2 = s\text{-scale-f}(da-2) \)
\( N_0 = \max\text{-prec}\text{-pref}(da-1, da-2) \)

Ref.: max-prec 8-13(62)

(64) \text{eval-sign}(op) =
\text{PASS:} bintg\text{-op} \cdot sign\text{-l}\text{-op} = val(op)
where: sign\text{-l}(v) = (v=0 \rightarrow O, T \rightarrow sign(v))

(65) \text{eval-fixed}(op, p, q) =
convert(rt\text{-fixed}(s\text{-da}(op), p, q), op)

(66) rt\text{-fixed}(da, p, q) =
is-intg-val(p) & is-Q(q) \rightarrow prec(new(FIX, da), p, 0)
T \rightarrow prec(new(FIX, da), p, q)

(67) \text{eval-float}(op, p) =
convert(prec(new(FLT, s\text{-da}(op)), p, q), op)

(68) \text{eval-floor}(op) =
val-op1(rt\text{-trunc-s\text{-da}(op)}, floor\text{-op}\text{-val}(op))

(69) \text{eval-ceil}(op) =
val-op1(rt\text{-trunc-s\text{-da}(op)}, ceil\text{-op}\text{-val}(op))

(70) \text{eval-trunc}(op) =
val-op1(rt\text{-trunc-s\text{-da}(op)}, trunc\text{-op}\text{-val}(op))
(71) \[ \text{rt-trunc}(\text{da}) = \]
\[ \text{is-FLT}\cdot\text{s-scale}(\text{da}) \rightarrow \text{da} \]
\[ \text{is-FIX}\cdot\text{s-scale}(\text{da}) \rightarrow \text{prec}(\text{da}, p_o, 0) \]

where:
\[ p_o = \min(N_0, \max(p_1 - q_1 + 1, 1)) \]
\[ p_1 = \text{s-prec}(\text{da}) \]
\[ q_1 = \text{s-scale-f}(\text{da}) \]
\[ N_0 = \text{max-prec}(\text{da}) \]

Ref.: max-prec 8-13(62)

(72) \[ \text{eval-bin}(\text{op}, p, q) = \]
\[ \text{convert}(\text{new-l}(\text{BIN}, \text{s-da}(\text{op}), p, q), \text{op}) \]

(73) \[ \text{eval-dec}(\text{op}, p, q) = \]
\[ \text{convert}(\text{new-l}(\text{DEC}, \text{s-da}(\text{op}), p, q), \text{op}) \]

(74) \[ \text{eval-prec}(\text{op}, p, q) = \]
\[ \text{convert}(\text{prec}(\text{s-da}(\text{op}), p, q), \text{op}) \]

(75) \[ \text{eval-add}(\text{op-1}, \text{op-2}, p, q) = \]
\[ \text{eval-infix-convert}(\text{op-1}, \text{op-2}, p, q, \text{ADD}) \]

(76) \[ \text{eval-multiply}(\text{op-1}, \text{op-2}, p, q) = \]
\[ \text{eval-infix-convert}(\text{op-1}, \text{op-2}, p, q, \text{MULT}) \]

(77) \[ \text{eval-divide}(\text{op-1}, \text{op-2}, p, q) = \]
\[ \text{eval-infix-convert}(\text{op-1}, \text{op-2}, p, q, \text{DIV}) \]

(78) \[ \text{eval-infix-convert}(\text{op-1}, \text{op-2}, p, q, \text{opor}) = \]
\[ \text{convert}(\text{da}_0, \text{op}); \]
\[ \text{op:infix-op}(\text{op-11}, \text{op-21}, \text{opor}); \]
\[ \text{op-11:convert}(\text{da}_0, \text{op-1}), \]
\[ \text{op-21:convert}(\text{da}_0, \text{op-2}) \]

where:
\[ \text{da}_0 = \text{prec}(\text{pref}(\text{s-da}(\text{op-1}), \text{s-da}(\text{op-2})), p, q) \]

for: \text{is-infix-operator}(\text{opor})
11.6.2.2 Mathematical generic builtin functions

The builtin 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 following table lists implementation defined instructions for evaluation of mathematical functions. They are defined for:

\[ \text{is-op}(op), \text{is-op}(op-1), \text{s-da}(op-1) = \text{s-da}(op), \]
\[ \text{is-FLT}s\cdot\text{scale}\cdot\text{s-da}(op) \]

\[ \text{eval-cplx}(op-1,op-2) = \]
\[ \text{eval-cplx-1}(op-11,op-21); \]
\[ \text{op-11:convert}(\text{pref}(\text{s-da}(op-1), \text{s-da}(op-2)), op-1), \]
\[ \text{op-21:convert}(\text{pref}(\text{s-da}(op-1), \text{s-da}(op-2)), op-2) \]

\[ \text{eval-cplx-1}(op-1,op-2) = \]
\[ \text{mk-op}(\text{new}(\text{CPLX}, \text{s-da}(op-1)), \text{cplx-vr}(\text{s-vr}(op-1), \text{s-vr}(op-2))) \]
Ref.: cplx-vr 8-9(30)

\[ \text{eval-real}(op) = \]
\[ \text{mk-op}(\text{new}(\text{REAL}, \text{s-da}(op)), \text{s-real}\cdot\text{s-vr}(op)) \]

\[ \text{eval-imag}(op) = \]
\[ \text{mk-op}(\text{new}(\text{REAL}, \text{s-da}(op)), \text{s-imag}\cdot\text{s-vr}(op)) \]

\[ \text{eval-conjg}(op) = \]
\[ \text{eval-cplx-1}(op-1,op-2); \]
\[ \text{op-1:eval-real}(op), \]
\[ \text{op-2:prefix-op}(op-21, \text{MINUS}); \]
\[ \text{op-21:eval-imag}(op) \]
Ref.: prefix-op 8-28(109)
Each of these instructions returns an operand op-res of the following form:

\[(84) \text{op-res} = \text{val-op}(s-da(op), v-res),\]

\[(85) \text{is-num-val}(v-res),\]

where the value v-res is an approximation of the mathematical expression given in the second column of the table, satisfying the condition stated in the third column, if the expression is not unique.

**Abbreviations**

\[v_{op} = \text{op-val}(op)\]

\[v_1 = \text{op-val}(op-1)\]

\[\pi = 3.1415...\]

<table>
<thead>
<tr>
<th>instruction</th>
<th>result value</th>
<th>condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>(86) eval-exp(op)</td>
<td>(\exp(v_{op}))</td>
<td>(-\pi \leq \text{imag}(v-res) \leq \pi)</td>
</tr>
<tr>
<td>(87) eval-log-real(op)</td>
<td>(\log(v_{op}))</td>
<td>(-\pi \leq \text{imag}(v-res) \leq \pi)</td>
</tr>
<tr>
<td>(88) eval-log-cplx(op)</td>
<td>(\log(v_{op}))</td>
<td>(-\pi \leq \text{imag}(v-res) \leq \pi)</td>
</tr>
<tr>
<td>(89) eval-log-10-1(op)</td>
<td>(\log_{10}(v_{op}))</td>
<td>(-\pi \leq \text{imag}(v-res) \leq \pi)</td>
</tr>
<tr>
<td>(90) eval-log-2-1(op)</td>
<td>(\log_2(v_{op}))</td>
<td>(-\pi \leq \text{imag}(v-res) \leq \pi)</td>
</tr>
<tr>
<td>(91) eval-tand(op)</td>
<td>(\tan(v_{op}, 180/\pi))</td>
<td>(-\pi \leq \text{imag}(v-res) \leq \pi)</td>
</tr>
<tr>
<td>(92) eval-tan(op)</td>
<td>(\tan(v_{op}))</td>
<td>(-\pi \leq \text{imag}(v-res) \leq \pi)</td>
</tr>
<tr>
<td>(93) eval-sind(op)</td>
<td>(\sin(v_{op}, 180/\pi))</td>
<td>(-\pi \leq \text{imag}(v-res) \leq \pi)</td>
</tr>
<tr>
<td>(94) eval-sin(op)</td>
<td>(\sin(v_{op}))</td>
<td>(-\pi \leq \text{imag}(v-res) \leq \pi)</td>
</tr>
<tr>
<td>(95) eval-cosd(op)</td>
<td>(\cos(v_{op}, 130/\pi))</td>
<td>(-\pi \leq \text{imag}(v-res) \leq \pi)</td>
</tr>
<tr>
<td>(96) eval-cos(op)</td>
<td>(\cos(v_{op}))</td>
<td>(-\pi \leq \text{imag}(v-res) \leq \pi)</td>
</tr>
<tr>
<td>(97) eval-tanh(op)</td>
<td>(\tanh(v_{op}))</td>
<td>(-\pi \leq \text{imag}(v-res) \leq \pi)</td>
</tr>
<tr>
<td>(98) eval-erf(op)</td>
<td>(\text{erf}(v_{op}))</td>
<td>(-\pi \leq \text{imag}(v-res) \leq \pi)</td>
</tr>
<tr>
<td>(99) eval-sqrt-real(op)</td>
<td>(\sqrt{v_{op}})</td>
<td>(v-res \geq 0)</td>
</tr>
<tr>
<td>(100) eval-sqrt-cplx(op)</td>
<td>(\text{sqrt}(v_{op}))</td>
<td>(\text{real}(v-res) \geq 0 \land \text{imag}(v-res) \geq 0)</td>
</tr>
<tr>
<td>(101) eval-erfc(op)</td>
<td>(1-\text{erf}(v_{op}))</td>
<td>(-\pi \leq \text{imag}(v-res) \leq \pi)</td>
</tr>
<tr>
<td>(102) eval-cosh(op)</td>
<td>(\cosh(v_{op}))</td>
<td>(-\pi \leq \text{imag}(v-res) \leq \pi)</td>
</tr>
<tr>
<td>(103) eval-sinh(op)</td>
<td>(\sinh(v_{op}))</td>
<td>(-\pi \leq \text{imag}(v-res) \leq \pi)</td>
</tr>
<tr>
<td>(104) eval-atanh-real(op)</td>
<td>(\text{arctanh}(v_{op}))</td>
<td>(-\pi/2 \leq \text{imag}(v-res) \leq \pi/2)</td>
</tr>
<tr>
<td>(105) eval-atanh-cplx(op)</td>
<td>(\text{arctanh}(v_{op}))</td>
<td>(-\pi/2 \leq \text{imag}(v-res) \leq \pi/2)</td>
</tr>
<tr>
<td>(106) eval-atand-1(op)</td>
<td>(\text{arctan}(v_{op}, 130/\pi))</td>
<td>(-\pi \leq \text{imag}(v-res) \leq \pi)</td>
</tr>
</tbody>
</table>

Table cont'd
The functions erf and arctan-2 are defined in the following, the other functions have their usual mathematical meaning.

\[ \text{erf}(x) = \frac{2}{\sqrt{\pi}} \int_{0}^{x} e^{-t^2} \, dt \]

\[ \text{arctan-2}(x, y) = \begin{cases} 
  \text{arctan}(x/y) & y > 0 \\
  \text{arctan}(x/y) + \pi \cdot \text{sign}(x) & y < 0 \\
  \frac{\pi}{2} \cdot \text{sign}(x) & y = 0 
\end{cases} \]

By means of the so defined implementation dependent instructions, the following instructions for evaluation of built-in functions are defined:

\[ \text{eval-log}(op) = \begin{cases} 
  \text{eval-log-real}(op) & \text{is-REAL}s \cdot \text{mode}s \cdot \text{da}(op) \land \text{op-val}(op) > 0 \\
  \text{eval-log-cplx}(op) & \text{is-CPLX}s \cdot \text{mode}s \cdot \text{ia}(op) \land \text{op-val}(op) \neq 0 \\
  \text{error} & \text{T} 
\end{cases} \]

\[ \text{eval-log-10}(op) = \begin{cases} 
  \text{eval-log-10-1}(op) & \text{op-val}(op) > 0 \\
  \text{error} & \text{T} 
\end{cases} \]

\[ \text{eval-log-2}(op) = \begin{cases} 
  \text{eval-log-2-1}(op) & \text{op-val}(op) > 0 \\
  \text{error} & \text{T} 
\end{cases} \]

\[ \text{eval-sqrt}(op) = \begin{cases} 
  \text{eval-sqrt-real}(op) & \text{is-REAL}s \cdot \text{mode}s \cdot \text{da}(op) \land \text{op-val}(op) > 0 \\
  \text{eval-sqrt-cplx}(op) & \text{is-CPLX}s \cdot \text{mode}s \cdot \text{da}(op) \\
  \text{error} & \text{T} 
\end{cases} \]
(117) \texttt{eval-atanh}(op) =

- \text{is-REAL\(\cdot\)s-mode\(\cdot\)s-da}(op) \& \text{abs\(\cdot\)op-val}(op) < 1 \quad \text{\texttt{eval-atanh-real}(op)}
- \text{is-CPLX\(\cdot\)s-mode\(\cdot\)s-da}(op) \& \text{op-val}(op) \neq 1 \& \text{op-val}(op) \neq -1 \quad \text{\texttt{eval-atanh-cplx}(op)}

(118) \texttt{eval-atan}(op, op-1) =

- \text{is-} \mathcal{Q}(op-1) \quad \text{\texttt{eval-atan-1}(op)}
- \text{op-val}(op) = 0 \& \text{op-val}(op-1) = 0 \quad \text{error}

\begin{align*}
\text{T} & \quad \text{eval-atan-2(op-0, op-1}); \\
& \quad \text{op-0: convert}(da_o, op), \\
& \quad \text{op-10: convert}(da_o, op-1)
\end{align*}

where: $da_o = \text{pref}(s-da(op), s-da(op-1))$

(119) \texttt{eval-atan}(op, op-1) =

\begin{align*}
\text{is-} \mathcal{Q}(op-1) \& \text{is-REAL\(\cdot\)s-mode\(\cdot\)s-da}(op) \quad \text{\texttt{eval-atan-real}(op)} \\
\text{is-} \mathcal{Q}(op-1) \& \text{is-CPLX\(\cdot\)s-mode\(\cdot\)s-da}(op) \& \text{op-val}(op) \neq \text{cplx}(0,1) \& \text{op-val}(op) \neq \text{cplx}(0,-1) \quad \text{\texttt{eval-atan-cplx}(op)}
\end{align*}

\begin{align*}
\text{is-CPLX\(\cdot\)s-mode\(\cdot\)s-da}(op) \& \text{op-val}(op) = 0 \& \text{op-val}(op-1) = 0 \quad \text{error}
\end{align*}

\begin{align*}
\text{T} & \quad \text{eval-atan-2(op-0, op-10)}; \\
& \quad \text{op-0: convert}(da_o, op), \\
& \quad \text{op-10: convert}(da_o, op-1)
\end{align*}

where: $da_o = \text{pref}(s-da(op), s-da(op-1))$
### 11.6.2.3 String generic builtin functions

#### eval-bit(op,i) =

1. `is-intg-val(i) → convert(\mu(BIT-DA; <s-length:i>),op)`
2. `is-Ω(i) → convert(BIT-DA,op)`

#### eval-char(op,i) =

1. `is-intg-val(i) → convert(\mu(CHAR-DA; <s-length:i>),op)`
2. `is-Ω(i) → convert(CHAR-DA,op)`

#### eval-substr(t,i,j,env) =

1. `eval-substr-l(op,t,i,j,env);`
2. `op: convert(STRING-DA,op-1);`
3. `op-1: eval-expr(t,env)`

Ref.: [eval-expr 7-18(43)]

#### eval-substr-l(op,t,i,j,env) =

1. `is-Ω(j) & lsk_{op} → pass-val-op(\frac{da_{rt}}{i} \text{LIST} \text{elem}(n,op-val(op)))`
2. `is-Ω(j) & lsk_{op} & i+j-lsk_{op} → pass-val-op(\frac{da_{rt}}{i} \text{LIST} \text{elem}(n,op-val(op)))`

T →

1. `eval-substr-2(op-1,i,j);`
2. `op-1: convert(STRING-DA,op-0);`
3. `op-0: eval-expr(t,env);`

Ref.: [eval-expr 7-18(43)]

Where: `da_{rt} = \text{new}($\text{VARYING, s-da$(op)$})`

`k_{op} = \text{length} \cdot op-val(op)`
(124) eval-substr-2(op,i,j) =

\[
\begin{align*}
\text{is-} Q(j) & \rightarrow \text{pass-val-op}(\text{da}_r, \text{LIST} \text{elem}(n, \text{op-val}(op))) \\
T & \rightarrow \text{pass-val-op}(\text{da}_r, \text{LIST} \text{elem}(n, \text{op-val}(op)))
\end{align*}
\]

where: \( \text{da}_r = \text{new}(\text{VARYING}, s-\text{da}(op)) \)
\( k_{op} = \text{length} \cdot \text{op-val}(op) \)
\( i_m = \max(1, i) \)
\( j_m = \min(j, k_{op} - i + 1) \)

(125) eval-index(op-1,op-2) =

\[
\begin{align*}
\text{s-base} \cdot s-\text{da}(op-1) & \neq s-\text{base} \cdot s-\text{da}(op-2) \\
\text{eval-index}(op-11,op-21); \\
\text{op-11:} & \text{convert}(\text{CHAR-DA}, op-1), \\
\text{op-21:} & \text{convert}(\text{CHAR-DA}, op-2)
\end{align*}
\]

\[
\begin{align*}
T & \rightarrow \text{PASS: bintg-op-index}(1, \text{op-val}(op-1), \text{op-val}(op-2))
\end{align*}
\]

(126) index(i,vl-1,vl-2) =

\[
\begin{align*}
\text{is-} <\!(vl-2) \vee \text{length}(vl-1) & < \text{length}(vl-2) \rightarrow 0 \\
\delta(vl-1; \{ \text{elem}(i) \mid i > \text{length}(vl-2) \}) & = vl-2 \rightarrow 1 \\
T & \rightarrow \text{index}(i+1, \text{tail}(vl-1), vl-2)
\end{align*}
\]

(127) eval-length(op) =

\[
\begin{align*}
\text{PASS: bintg-op-length} \cdot \text{op-val}(op)
\end{align*}
\]

(128) eval-high(i) =

\[
\begin{align*}
\text{pass-val-op}(\text{char-da}(i), \text{LIST} \text{CHAR}_l)
\end{align*}
\]

where: \( \text{CHAR}_l = (\text{char}) (\text{is-char-val}(\text{char}) \& \\
\neg (\exists \text{char-1}) (\text{collat}(\text{char-1}) > \text{collat}(\text{char})) \)

Ref.: collat 8-20(83)

(129) eval-low(i) =

\[
\begin{align*}
\text{pass-val-op}(\text{char-da}(i), \text{LIST} \text{CHAR}_o)
\end{align*}
\]

where: \( \text{CHAR}_o = (\text{char}) (\text{is-char-val}(\text{char}) \& \\
\neg (\exists \text{char-1}) (\text{collat}(\text{char-1}) < \text{collat}(\text{char})) \)

Ref.: collat 8-20(83)
(130) \( \text{eval-repeat}(\text{op}, i) = \)
\[
\text{eval-repeat-1}(\text{op}, \text{op}, i)
\]

(131) \( \text{eval-repeat-1}(\text{op-1}, \text{op}, i) = \)
\[
\begin{align*}
\text{isO} & \rightarrow \text{PASS: op-1} \\
\text{T} & \rightarrow \text{eval-repeat-1}(\text{op-2}, \text{op}, i - 1); \\
\text{op-2: infix-op}(\text{op-1}, \text{op}, \text{CAT})
\end{align*}
\]

(132) \( \text{rt-repeat}(\text{da}, i) = \)
\[
\begin{align*}
\text{isO} & \rightarrow \text{da} \\
\text{T} & \rightarrow \mu(\text{da}; <\text{s-length:}(i+1).\text{s-length}(\text{da})>)
\end{align*}
\]

(133) \( \text{eval-unspec}(\text{gen}) = \)
\[
\begin{align*}
\text{is-e-ret-type* s-da}(\text{gen}) & \rightarrow \\
\text{mk-op}(\text{rt-unspec* s-da}(\text{gen}), \text{el-rf}(\text{s-pp}(\text{gen}) (\text{S}))) \\
\text{T} & \rightarrow \text{error}
\end{align*}
\]

Ref.: \text{is-e-ret-type 2-19(47), el-rf 2-22(68)}

(134) \( \text{rt-unspec}(\text{da}) = \)
\[
\mu_0(<\text{s-base:BIT}, <\text{s-length:unspec-length}(\text{da})>)
\]

Ref.: \text{unspec-length 7-17(42)}

(135) \( \text{eval-bool}(\text{op-1}, \text{op-2}, \text{op-3}) = \)
\[
\begin{align*}
\text{pass-val-op}(\text{pref}(\text{s-da}(\text{op-1}), \text{s-da}(\text{op-2})), \\
\text{bool}(\text{op-val}(\text{op-1}), \text{op-val}(\text{op-2}), \text{op-val}(\text{op-3})))
\end{align*}
\]

11.6.2.3
(136) bool(v-1,v-2,v-3) =

\[
\text{length}_m \text{ LIST } \prod_{i=1}^n \text{ elem-bool}(\text{elem}(i,v_1),\text{elem}(i,v_2),v_3)
\]

where: \( \text{length}_m = \max(\text{length}(v-1),\text{length}(v-2)) \)

\[v_1 = \text{adjust-string}(\text{BIT}, \text{length}_m, v-1)\]
\[v_2 = \text{adjust-string}(\text{BIT}, \text{length}_m, v-2)\]

for: \( \text{is-bit-val-list}(v-1), \text{is-bit-val-list}(v-2), \text{is-bit-val-list}(v-3) \)

Ref.: adjust-string 8-10(36)

(137) elem-bool(bit-1,bit-2,bit-list) =

\[
is-0-\text{BIT}(bit-1) \& is-0-\text{BIT}(bit-2) \rightarrow elem(1,bit-list)
is-0-\text{BIT}(bit-1) \& is-1-\text{BIT}(bit-2) \rightarrow elem(2,bit-list)
is-1-\text{BIT}(bit-1) \& is-0-\text{BIT}(bit-2) \rightarrow elem(3,bit-list)
is-1-\text{BIT}(bit-1) \& is-1-\text{BIT}(bit-2) \rightarrow elem(4,bit-list)
\]

for: \( \text{is-bit-val}(bit-1), \text{is-bit-val}(bit-2), \text{is-bit-val-list}(bit-list) \)

11.6.2.4 Generic builtin functions for array manipulation

(138) eval-sum(opl) =

\[
\text{iterate-infix-op}(\text{head}(opl),\text{tail}(opl),ADD)
\]

(139) eval-prod(opl) =

\[
\text{iterate-infix-op}(\text{head}(opl),\text{tail}(opl),MULT)
\]

(140) eval-all(opl) =

\[
\text{iterate-infix-op}(\text{head}(opl),\text{tail}(opl),AND)
\]

(141) eval-any(opl) =

\[
\text{iterate-infix-op}(\text{head}(opl),\text{tail}(opl),OR)
\]
iterate-infix-op(op,opl,opor) =

\[\begin{align*}
\text{is-<>}(opl) & \quad \text{PASS: op} \\
T & \quad \text{iterate-infix-op}(op-1,\text{tail}(opl),opor) ; \\
\text{op-1:infix-op}(op,\text{head}(opl),opor)
\end{align*}\]

for: is-infix-operator(opor)

\[\begin{align*}
eval-poly(opl-1,opl-2) &= \\
eval-poly-1(op-1,op-2,\text{tail}(opl-1),opl-2) ; \\
\text{op-1:convert}(s\_da_0,\text{head}(opl-1)) , \\
\text{op-2:eval-op-1}(s\_da_0,1)
\end{align*}\]

where: \(s\_da_0 = \text{pref}(s\_da_0 \text{head}(opl-1),s\_da_0 \text{head}(opl-2))\)

\[\begin{align*}
eval-poly-1(op-1,op-2,opl-1,opl-2) &= \\
eval-poly-1(op-11,op-21,\text{tail}(opl-1),\text{tail}_2); \\
\text{op-11:infix-op}(op-1,op-0,ADD), \\
\text{op-0:eval-infix-expr}(op-21,\text{head}(opl-1),MULT) ; \\
\text{op-21:eval-infix-expr}(op-2,\text{head}(opl-2),MULT)
\end{align*}\]

where: \(\text{tail}_2 = (\text{length}(opl-2) > 1 \quad \text{tail}(opl-2) , \\
T \quad \text{op-2})\)

Ref.: eval-infix-expr 8-14(65)

eval-lbound(eda,i) =

\[\begin{align*}
is\_array(eda) \& i = 1 & \quad \text{pass-bintg-op}(s\_lbd(eda)) \\
is\_array(eda) \& i > 1 & \quad \text{eval-lbound}(s\_elem(eda),i-1) \\
T & \quad \text{error}
\end{align*}\]

eval-hbound(eda,i) =

\[\begin{align*}
is\_array(eda) \& i = 1 & \quad \text{pass-bintg-op}(s\_ubd(eda)) \\
is\_array(eda) \& i > 1 & \quad \text{eval-hbound}(s\_elem(eda),i-1) \\
T & \quad \text{error}
\end{align*}\]
11.6.2.5 Condition builtin functions

The values of condition builtin functions, if used in an on-unit, are determined from the contents of the condition builtin function part of the condition state s-cbif(CS), (cf. 9.3). If used outside on-units, the functions return standard values. For the function chl(id), cf. 10-76(215)

(147) \[ \text{eval-dim}(eda, i) = \]
\[ \text{is-array}(eda) \& i = 1 \rightarrow \text{pass-bintg-op}(s-\text{ubd}(eda) - s-lbd(eda) + 1) \]
\[ \text{is-array}(eda) \& i > 1 \rightarrow \text{eval-dim}(s-\text{elem}(eda), i - 1) \]
\[ T \rightarrow \text{error} \]

(11.6.2.5)
(152) **eval-onkey** =

\[ \text{is-} \left( \text{s-onkey}\cdot \text{s-cbif}(CS) \right) \rightarrow \text{pass-val-op}(\text{ONKEY-DA}, \langle \rangle) \]
\[ T \rightarrow \text{pass-val-op}(\text{ONKEY-DA}, \text{s-onkey}\cdot \text{s-cbif}(CS)) \]

(153) **ONKEY-DA** =

\[ \mu_0(\langle \text{s-base:CHAR}, \text{s-length:ONKEY-LENGTH}, \text{s-varying:T} \rangle) \]

**Note**: ONKEY-LENGTH is an implementation defined integer.

(154) **eval-oncode** =

\[ \text{is-} \left( \text{s-oncode}\cdot \text{s-cbif}(CS) \right) \rightarrow \text{pass-bintg-op}(O) \]
\[ T \rightarrow \text{pass-bintg-op}(\text{s-oncode}\cdot \text{s-cbif}(CS)) \]

(155) **eval-datafield** =

\[ \text{is-} \left( \text{s-datafield}\cdot \text{s-cbif}(CS) \right) \rightarrow \text{pass-val-op}(\text{ID-CHAR-DA}, \langle \rangle) \]
\[ T \rightarrow \text{pass-val-op}(\text{ID-CHAR-DA}, \text{s-datafield}\cdot \text{s-cbif}(CS)) \]

(156) **eval-oncount** =

\[ \text{is-} \left( \text{s-oncount}\cdot \text{s-cbif}(CS) \right) \rightarrow \text{pass-bintg-op}(O) \]
\[ T \rightarrow \text{pass-bintg-op}(\text{s-oncount}\cdot \text{s-cbif}(CS)) \]

11.6.2.6 **List-processing builtin functions**

(157) **eval-addr(gen)** =

\[ \text{is-connected}(gen) \rightarrow \text{PASS:s-pp}(gen) \]
\[ T \rightarrow \text{error} \]

**Ref.**: is-connected 7-33(87)
(158) **eval-ptr(op, gen)** =

\[
\text{is-area} \times s-da(gen) \land \text{is-applic-1(op-val(op),size-1\times s-pp(gen))} \rightarrow \\
\text{pass-val-op(PTR,(op-val(op))\times(s-pp(gen)))}
\]

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

Ref.: is-applic-1 2-22(61), size-1 2-22(62)

(159) **eval-offset(op, gen)** =

\[
\text{is-area} \times s-da(gen) \rightarrow \\
\text{pass-val-op(OFFSET,(lo)\times s-pp(gen))}
\]

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

(160) **eval-null** =

\[
\text{pass-val-op(PTR,NPTR)}
\]

(161) **eval-nullo** =

\[
\text{pass-val-op(OFFSET,NPTR)}
\]

(162) **is-ptr-val(NPTR) \land (∀stg)( \neg \text{is-applic(NPTR,stg)})**

Ref.: is-ptr-val 2-21(52), is-applic 2-21(57)

Note: NPTR is that pointer which is not applicable to any storage.

(163) **eval-empty** =

\[
\text{PASS: } \mu_e(\{ <s-da: \mu_e(\{ <s-size:O> \}), <s-vr:EMPTY-AREA> \})
\]

where: (is-vr \ & \ is-\{ \} \times alloc-state)(EMPTY-AREA)

Ref.: alloc-state 2-23(73), is-vr 2-22(69)

Note: EMPTY-AREA is the area with empty allocation state.

11.6.2.7 Other built-in functions

(164) **eval-date** =

\[
\text{is-valid-date\times s-date(TD)} \rightarrow \\
\text{pass-val-op(char-da(6),num-char-list(6,s-date(TD)))}
\]

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

11.6.2.7
Ref.: num-char-list 8-33(127)

(165) is-valid-date(d) =

\[ d > 0 \land (\text{month}_d \in \{1,3,5,7,8,10,12\} \land 1 \leq \text{day}_d \leq 31 \lor \text{month}_d \in \{4,6,9,11\} \land 1 \leq \text{day}_d \leq 30 \lor \text{month}_d = 2 \land 1 \leq \text{day}_d \leq 28 \lor \text{modulo}(\text{year}_d, 4) = 0 \land \text{month}_d = 2 \land \text{day}_d = 29) \]

where: \( \text{year}_d = \text{trunc}(d/10000) \)

\( \text{month}_d = \text{modulo}(\text{trunc}(d/100), 100) \)

\( \text{day}_d = \text{modulo}(d, 100) \)

Note: It is assumed that the program is interpreted between March 1, 1900 and February 28, 2100.

(166) eval-time =

\[ \text{is-time}(\text{td}) > 0 \]

\[ \text{pass-val-op}(\text{char-da}(9), \text{num-char-list}(9, \text{time}_t)) \]

\[ \text{T} \]

where: \( \text{time}_t = 10^7 \cdot \text{h}_t + 10^5 \cdot \text{min}_t + \text{msec}_t \)

\( \text{h}_t = \text{modulo}(\text{trunc}(\text{tm}_t/3600000), 24) \)

\( \text{min}_t = \text{modulo}(\text{trunc}(\text{tm}_t/60000), 60) \)

\( \text{msec}_t = \text{modulo}(\text{tm}_t, 60000) \)

\( \text{tm}_t = \text{s-time}(\text{T}) \)

Ref.: num-char-list 8-33(127)

(167) eval-assignment(t, env) =

\[ \text{is-CTL}\cdot\text{s-ctg-cl}(\text{attr}_t) \land \text{is-gen-head}(n_t(DN)(AG)) \]

\[ \text{pass-val-op}(\text{bit-da}(1), 1\cdot\text{BIT}) \]

\[ \text{is-CTL}\cdot\text{s-ctg-cl}(\text{attr}_t) \]

\[ \text{pass-val-op}(\text{bit-da}(1), 0\cdot\text{BIT}) \]

\[ \text{T} \]

where: \( \text{attr}_t = \text{s-attr}_t(\text{AT}) \)

\( n_t = \text{sel}\cdot\text{s-id-list}(t)(\text{env}) \)

(168) eval-lineno(t, env) =

\[ \text{pass-file-bif}(\text{s-line}, t, \text{env}) \]

11.6.2.7
(169) \( \text{eval-count}(t, \text{env}) = \text{pass-file-bif}(s\text{-count}, t, \text{env}) \)

(170) \( \text{file-bif}(s, t, \text{env}) = \)

\[
\text{is-bintg-op} \ast s \cdot u_t(FU) \rightarrow s \cdot u_t(FU)
\]

\( T \rightarrow \text{error} \)

where: \( f d_t = s\text{-fu}(\text{sel}\cdot s\text{-id-list}(t)(\text{env})(\text{DN})(\text{FD})) \)

\( u_t = (\text{is-STAND-PRINT}(f d_t) \rightarrow s\text{-fu} \cdot \text{s-stand-print}(\text{FD}), T \rightarrow f d_t) \)

for: \( s \in \{s\text{-line}, s\text{-count}\} \)

(171) \( \text{eval-round}(t, i, \text{env}) = \)

\( \text{eval-round-1}(\text{op}, i) \);

\( \text{op: eval-expr}(t, \text{env}) \)

Ref.: \( \text{eval-expr} 7-18(43) \)

(172) \( \text{eval-round-1}(\text{op}, i) = \)

\( \text{is-num-pic}(d a_\text{op}) \rightarrow \text{eval-round-1}(\text{c-op}, i); \)

\( \text{c-op: convert(AR-DA, op)} \)

\( (\text{is-arithm} \& \text{is-CPLX}\cdot s\text{-mode})(d a_\text{op}) \rightarrow \)

\( \text{eval-cplx}(\text{op-1}, \text{op-2}); \)

\( \text{op-1: eval-round-1}(\text{op}_{\text{real}}, i), \)

\( \text{op-2: eval-round-1}(\text{op}_{\text{imag}}, i) \)

\( \text{is-arithm}(d a_\text{op}) \rightarrow \text{pass-val-op}(d a_{\text{res}}, v); \)

\( v: \text{test-fl}(d a_{\text{res}}, \text{round}(d a, v_{\text{op}}, i)) \)

\( \text{is-string-type}(d a_\text{op}) \rightarrow \text{PASS:op}, T \rightarrow \text{error} \)

where: \( d a_\text{op} = s\text{-da}(\text{op}) \)

\( v_{\text{op}} = s\text{-vr}(\text{op}) \)

\( v_{\text{op}} = \text{op-val}(\text{op}) \)

\( \text{op}_{\text{real}} = \mu_{o}(\text{<s\text{-da:da}_{\text{real}}, s\text{-vr:s\text{-real}}(v_{\text{op}})>}) \)

\( \text{op}_{\text{imag}} = \mu_{o}(\text{<s\text{-da:da}_{\text{real}}, s\text{-vr:s\text{-imag}}(v_{\text{op}})>}) \)

\( d a_{\text{real}} = \mu(d a_{\text{op}}; \text{<s\text{-mode:REAL}>}) \)

\( d a_{\text{res}} = \text{rt-round}(d a_{\text{op}}) \)

cont'd
Ref.: is-num-pic 8-44(166), test-fl 8-17(73),
is-string-type 8-23(93).

(173) $(\forall da, v, i) \{(is-arithm \& is-REAL\#s-mode \& is-FIX\#s-scale)(da) \& v \in v-O-set(da) \Rightarrow$
\[ \text{round}(da, v, i) = \text{sign}(v) \cdot (v_{\text{round}}/\text{base}_{da}^i) \}\]

where: 
\[ v_{\text{round}} = \text{mod}(v_1, 1) \geq 0.5 \Rightarrow v_1 + 1, \quad T \Rightarrow v_1 \]
\[ v_1 = \text{abs}(v) \cdot \text{base}_{da}^i \]
\[ \text{base}_{da} = (is-BIN\#s-base(da) \Rightarrow 2, \quad T \Rightarrow 10) \]

Ref.: v-O-set 8-7(20).

Note: For floating-point da, the function round(da,v,i) is implementation-defined, since no particular normalization rules have been assumed.

(174) \[ \text{rt-round}(da) = \]
\[ \text{is-num-pic}(da) \rightarrow \text{rt-round\#complete-tg}(AR-DAid a) \]
\[ (is-arithm \& is-FIX\#s-scale)(da) \rightarrow \quad \text{prec}(da, \text{min}(\max-\text{prec}(da),\text{\#s-prec}(da)+1),\text{\#s-scale-f}(da)) \]
\[ \text{is-arithm}(da) \lor \text{is-string-type}(da) \rightarrow da \]
\[ T \rightarrow error \]

Ref.: is-num-pic 8-44(166), complete-tg 8-37(143),
\[ \max-\text{prec} 8-13(62), \text{is-string-type} 8-23(93). \]

(175) \[ \text{eval-string}(gen) = \]
\[ \text{is-PACKED\#s-dens\#s-mi}(gen) \& \text{is-connected}(gen) \& \text{is-non-varying\#s-da}(gen) \rightarrow \]
\[ \text{mk-op}(\text{rt-string\#s-da}(gen),\text{el-rf}(s-\text{pp}(gen)(g))) \]
\[ T \rightarrow error \]

Ref.: is-connected 7-33(87), el-rf 2-22(68), is-non-varying 7-29(75).

(176) \[ \text{rt-string}(eda) = \]
\[ \text{is-bit-aggr}(eda) \rightarrow \text{bit-da\#s-string-extent}(eda) \]
\[ \text{is-char-aggr}(eda) \rightarrow \text{char-da\#s-string-extent}(eda) \]
\[ T \rightarrow error \]

Ref.: is-bit-aggr 7-30(76), is-char-aggr 7-30(77), string-extent 7-30(78)
(177) **eval-status**(op) = 
    pass-val-op(BINTG-DA,s-status*op-val(op))

(178) **eval-completion**(op) = 
    pass-val-op(bit-da(1),<s-compl*op-val(op)>)

(179) **eval-priority**(op) =
    convert(PRI-DA,op-0);
    op-0:eval-infix-expr(op-1,op-2,SUBTR);
    op-1:convert-2(PRI-DA,op),
    op-2:convert-2(PRI-DA,gen-op(s-tv(TE),E))

Ref.: convert-2 4-5(9) , PRI-DA 4-5(6), gen-op 7-19(47),
     eval-infix-expr 8-14(65)
12. ABSTRACT SYNTAX OF TEXT

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

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

\[\text{is-program}.\]

The contents of the abstract syntax is the definition of this predicate. The definition is 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 are the following elementary objects:

(a) A finite class of constant elementary objects, usually denoted by names written in capital letters (e.g. EXT, A-CHAR, 9-CHAR, T), including the two special elementary objects \(<>\) (empty list) and \(.*\).

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

(c) The infinite class of values of (positive and negative) integers, characterized by the predicate \(\text{is-intg-val}\).

(d) The infinite class of implementation defined value representations (cf. 2.4.1), characterized by the predicate \(\text{is-vr}\).

(e) Some classes of finite sets of objects. The elements of these sets are themselves described by predicates defined in the abstract syntax (e.g. \(\text{is-init-set}\)), including the empty set \(\{\\}\).
12.1 Notations

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).

(a) \( \text{is-pred} = \text{is-pred}_1 \lor \ldots \lor \text{is-pred}_n \)
means that is-pred is defined by:
\[
\text{is-pred}(x) = \text{is-pred}_1(x) \lor \ldots \lor \text{is-pred}_n(x)
\]

(b) \( \text{is-pred} = (\langle s\text{-sel}_1: \text{is-pred}_1 \rangle, \ldots, \langle s\text{-sel}_n: \text{is-pred}_n \rangle) \)
means that is-pred is defined by:
\[
\text{is-pred}(x) = \exists x_1, \ldots, x_n (\bigwedge_{i=1}^n \text{is-pred}_i(x_i) \land x = \mu_0(\{ \langle s\text{-sel}_1:x_1 \rangle, \ldots, \langle s\text{-sel}_n:x \rangle \}))
\]

(c) \( \text{is-pred} = \text{is-OBJ} \)
where OBJ is the name of a constant elementary object or \( \emptyset \), means that is-pred is defined by:
\[
\text{is-pred}(x) = x = \text{OBJ}
\]

(d) \( \text{is-pred} = \text{is-pred}_1\text{-list} \)
means that is-pred is defined by:
\[
\text{is-pred}(x) = \exists x_1, \ldots, x_n (\bigwedge_{i=1}^n \text{is-pred}_i(x_i) \land \neg \text{is-OBJ}(x_i) \land x = \langle x_1, \ldots, x_n \rangle)
\]
Note: This includes especially (for \( n = 0 \)) is-pred(\langle \rangle).

(e) \( \text{is-pred} = \text{is-pred}_1\text{-set} \)
means that is-pred is defined by:
\[
\text{is-pred}(x) = \exists x_1, \ldots, x_n (\bigwedge_{i=1}^n \text{is-pred}_i(x_i) \land x = \{ x_1, \ldots, x_n \})
\]
Note: This includes especially (for \( n = 0 \)) is-pred(\{ \}).
(f) \( \text{is-pred} = \{ <f(y): \text{is-pred}_2 > | \text{is-pred}_1 (y) \} \)

means that \( \text{is-pred} \) is defined by:

\[
\text{is-pred}(x) = \left( \exists y_1, z_1, \ldots, y_n, z_n \left( \bigwedge_{i=1}^{n} (\text{is-pred}_1 (y_i) \land \text{is-pred}_2 (z_i)) \land x = \mu_{\sigma} (<f(y_1):z_1>, \ldots, <f(y_n):z_n>) \right) \right)
\]

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

(g) \( \text{is-pred} = \{ \langle \text{sel-fct}(1): \text{is-pred}_0 \rangle, \ldots \} \)

where \( \text{sel-fct} \) is one of the selector functions \( \text{elem} \), \( \text{sub} \) mapping integer values into selectors, means that the predicate \( \text{is-pred} \) is defined by:

\[
\text{is-pred}(x) = \left( \exists m, y_1, \ldots, y_m \left( m \geq 1 \land \bigwedge_{i=1}^{m} \text{is-pred}_0 (y_i) \land x = \mu_{\sigma} (<\text{sel-fct}(1):y_1>, \ldots, <\text{sel-fct}(m):y_m>) \right) \right)
\]

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 (a); each of these classes may be defined by giving the selectors of its objects and the object classes of the corresponding components (b); these two kinds of definitions are iterated until one finally comes down to classes consisting of only one elementary object (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):

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

(3) The most frequent use of notation (f) is that the function \( f \) is the identity and \( \text{is-pred}_1 \) is a predicate characterizing a class of objects which in the same time are used as selectors: \( \text{is-id} \) or (not in the abstract syntax of text) \( \text{is-n} \).
Also the following combination of notations (b) and (g) is used:

\[
\text{is-pred} = (\langle s\text{-sel}_1: \text{is-pred}_1 \rangle, \ldots, \langle s\text{-sel}_n: \text{is-pred}_n \rangle, \\
\langle \text{sel-fct}(1): \text{is-pred}_0 \rangle, \ldots)
\]

which means

\[
\text{is-pred}(x) \equiv (\exists x_1, \ldots, x_n, m, y_1, \ldots, y_m) (m \geq 1 \land \bigwedge_{i=1}^{m} \text{is-pred}_1(x_i) \land \bigwedge_{i=1}^{m} \text{is-pred}_0(y_i) \land \\
x = \mu_0 (\langle s\text{-sel}_1:x_1, \ldots, s\text{-sel}_n:x_n, \\
\langle \text{sel-fct}(1):y_1, \ldots, \text{sel-fct}(m):y_m \rangle))
\]

i.e., \(x\) has besides the fixed set of components \(s\text{-sel}_1, \ldots, s\text{-sel}_n\) belonging to the classes \(\text{is-pred}_1, \ldots, \text{is-pred}_n\) an indefinite but finite set of components \(\text{sel-fct}(i)\) belonging to the class \(\text{is-pred}_0\).

### 12.2 Predicate Definitions. Program

1. \(\text{is-program} = (\langle s\text{-decl-part}: \text{is-program-decl-part} \rangle, \\
\langle s\text{-body-list}: \text{is-body-list} \rangle)
2. \(\text{is-program-decl-part} = \{\langle \text{id}: \text{is-ext-entry} \rangle \mid \text{id} = \text{id}(\text{id})\}
3. \(\text{is-ext-entry} = (\langle s\text{-scope}: \text{EXT} \rangle, \\
\langle s\text{-param-list}: \text{is-}^* \rangle, \\
\langle s\text{-ret-type}: \text{is-ret-type} \rangle, \\
\langle s\text{-den}: \text{is-intg-val} \rangle)
4. \(\text{is-body} = (\langle s\text{-decl-part}: \text{is-decl-part} \rangle, \\
\langle s\text{-body-list}: \text{is-body-list} \rangle, \\
\langle s\text{-cond-part}: \text{is-cond-part} \rangle, \\
\langle s\text{-st-list}: \text{is-proc-st-list} \rangle)
5. \(\text{is-opt} = \text{is-T} \lor \text{is-}^*\)
6. \(\text{is-intg-val} = \)

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 \(\text{is-num-val}\) (cf. 8.1.2.1).
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.

12.2.1 Declarations

is-decl-part = ( [id:is-decl] || is-id(id) )

is-decl = is-prop-var v is-entry v is-file v is-defined v is-based v is-BUILTIN v is-LABEL v is-format-attr v is-generic v is-COND

is-prop-var = ( <s-scope:is-scope>, <s-stg-cl:is-stg-cl v is-ԑ>, <s-da:is-named-da>, <s-dens:is-dens>, <s-init-set:is-init-set> )

is-scope = is-INT v is-EXT v is-PARAM

is-stg-cl = is-STATIC v is-CTL v is-AUTO

is-dens = is-PACKED v is-ALIGNED

is-init = ( <s-ref:is-basic-ref>, <s-init-spec:is-init-elem-list v is-call-st> )

is-init-elem = is-init-iter v is-expr.v is-*

is-init-iter = ( <s-rep-factor:is-expr>, <s-item-list:is-init-elem-list> )

is-entry = ( <s-scope:is-scope>, <s-param-list:is-descr-list v is-*>, <s-ret-type:is-ret-type>, <s-den:is-intg-val v is-ԑ> )
(18) \( \text{is-descr} = \text{is-var-descr} \lor \text{is-entry-descr} \lor \text{is-file-descr} \lor \text{is-*} \)

(19) \( \text{is-var-descr} = \langle \text{s-stg-cl:is-CTL} \lor \text{is-Q} \rangle, \)
\( \quad \langle \text{s-da:is-da-descr} \rangle, \)
\( \quad \langle \text{s-dens:is-dens} \rangle \)

(20) \( \text{is-entry-descr} = \langle \text{s-param-list:is-descr-list} \lor \text{is-*} \rangle \)

(21) \( \text{is-file-descr} = \langle \text{s-file-attr:is-file-attr-set} \rangle \)

(22) \( \text{is-ret-type} = \text{is-arithm} \lor \text{is-string-da} \lor \text{is-piC} \lor \text{is-area-da} \lor \text{is-PTR} \lor \text{is-OFFSET} \)

(23) \( \text{is-file} = \langle \text{s-scope:is-scope} \rangle, \)
\( \quad \langle \text{s-file-attr:is-file-attr-set} \rangle, \)
\( \quad \langle \text{s-title:is-id} \rangle, \)
\( \quad \langle \text{s-io-env:is-io-opt} \rangle \)

(24) \( \text{is-file-attr} = \text{is-STREAM} \lor \text{is-RECORD} \lor \text{is-IN} \lor \text{is-OUT} \lor \text{is-UPD} \lor \text{is-SEQ} \lor \text{is-DIR} \lor \text{is-BUF} \lor \text{is-UNBUF} \lor \text{is-PRINT} \lor \text{is-BACK} \lor \text{is-EXCL} \lor \text{is-KEYED} \)

(25) \( \text{is-io-opt} = \)
\( \text{Note: This predicate is implementation defined.} \)

(26) \( \text{is-defined} = \langle \text{s-base:is-basic-ref} \rangle, \)
\( \quad \langle \text{s-pos:is-intg-val} \lor \text{is-Q} \rangle, \)
\( \quad \langle \text{s-da:is-named-da} \rangle, \)
\( \quad \langle \text{s-dens:is-dens} \rangle \)

(27) \( \text{is-based} = \langle \text{s-ptr:is-opt-ref} \rangle, \)
\( \quad \langle \text{s-da:is-named-da} \rangle, \)
\( \quad \langle \text{s-dens:is-dens} \rangle, \)
\( \quad \langle \text{s-init-set:is-init-set} \rangle \)

(28) \( \text{is-format-attr} = \langle \text{s-format-list:is-format-list} \rangle, \)
\( \quad \langle \text{s-cond-part:is-cond-part} \rangle \)

(29) \( \text{is-generic} = \langle \text{s-generic-ref-set:is-generic-ref-set} \rangle \)

(30) \( \text{is-generic-ref} = \langle \text{s-id:is-id} \rangle, \)
\( \quad \langle \text{s-param-list:is-descr-list} \rangle \)
12.2.1.1 Data attributes

(31) is-named-da = is-named-array v is-named-struct v is-named-cell v is-scalar-da

(32) is-named-array = (s-lbd:is-extent, s-ubd:is-extent, s-elem:is-named-da)

(33) is-extent = is-expr v is-

(34) is-named-struct = is-named-succ-list

(35) is-named-succ = (s-id:is-id, s-da:is-named-da)

(36) is-named-cell = (s-cell:is-T, elem(i):is-named-succ,...)

(37) is-scalar-da = is-arithm v is-string-da v is-pic v is-area-da v is-label v is-PTR v is-OFFSET v is-TASK v is-EVENT

(38) is-arithm = (s-mode:is-REAL v is-CPLX, s-base:is-BIN v is-DEC, s-scale:is-FIX v is-FLT, s-prec:is-intg-val, s-scale-f:is-intg-val v is-\mathbb{Q})

(39) is-string-da = (s-base:is-BIT v is-CHAR, s-length:is-extent, s-varying:is-opt)

(40) is-area-da = is-AREA v is-spec-area-da

(41) is-spec-area-da = (s-size:is-extent)

(42) is-label = is-LABEL v is-rest-label

(43) is-rest-label = (s-label-set:is-id-set)

(44) is-da-descr = is-array-descr v is-struct-descr v is-cell-descr v is-scalar-da
(45) \( \text{is-array-descr} = (\text{is-lbd:is-extent}, \)
\( \text{is-ubd:is-extent}, \)
\( \text{is-elem:is-da-descr}) \)

(46) \( \text{is-struct-descr} = \text{is-succ-descr-list} \)

(47) \( \text{is-succ-descr} = (\text{is-da:is-da-descr}) \)

(48) \( \text{is-cell-descr} = (\text{is-cell:is-T}, \)
\( \text{elem(i):is-succ-descr},...) \)

12.2.1.2 Pictures

(49) \( \text{is-pic} = \text{is-dec-pic} \lor \text{is-sterling-pic} \lor \text{is-bin-pic} \lor \text{is-char-pic} \)

(50) \( \text{is-dec-pic} = (\text{is-mode:is-REAL} \lor \text{is-CPLX}, \)
\( \text{is-mt-field:is-dec-spec-list}, \)
\( \text{is-mt-unit:is-intg-val} \lor \text{is-Q}, \)
\( \text{is-scale-f:is-intg-val} \lor \text{is-Q}, \)
\( \text{is-exp-sep:is-E-CHAR} \lor \text{is-Q}, \)
\( \text{is-exp-field:is-dec-spec-list} \lor \text{is-Q}) \)

(51) \( \text{is-dec-spec} = \text{is-9-CHAR} \lor \text{is-2-CHAR} \lor \text{is-ASTER} \lor \text{is-Y-CHAR} \lor \text{is-T-CHAR} \lor \text{is-I-CHAR} \lor \text{is-R-CHAR} \lor \text{is-SIGN} \lor \text{is-PLUS} \lor \text{is-MINUS} \lor \text{is-DOLLAR} \lor \text{is-BLANK} \lor \text{is-COMMA} \lor \text{is-SLASH} \lor \text{is-POINT} \lor \text{is-C-CHAR} \lor \text{is-D-CHAR} \lor \text{is-B-CHAR} \)

(52) \( \text{is-sterling-pic} = (\text{is-mode:is-REAL}, \)
\( \text{is-mt-field:is-sterling-spec-list}, \)
\( \text{is-stat-part-end:is-intg-val}, \)
\( \text{is-pound-end:is-intg-val}, \)
\( \text{is-shill-end:is-intg-val}, \)
\( \text{is-mt-unit:is-intg-val} \lor \text{is-Q}) \)

(53) \( \text{is-sterling-spec} = \text{is-6-CHAR} \lor \text{is-7-CHAR} \lor \text{is-8-CHAR} \lor \text{is-S-CHAR} \lor \text{is-dec-spec} \)
(54) is-bin-pic = (is-mode:is-REAL v is-CPLX),
   (is-mt-field:is-bin-spec-list),
   (is-mt-unit:is-intq-val v is-Q),
   (is-scale-f:is-intq-val v is-Q),
   (is-exp-field:is-bin-spec-list v is-Q)

(55) is-bin-spec = is-1-CHAR v is-2-CHAR v is-3-CHAR v is-SIGN

(56) is-char-pic = (is-field:is-char-spec-list)

(57) is-char-spec = is-X-CHAR v is-A-CHAR v is-9-CHAR

12.2.1.3 Formats

(58) is-format = is-format-iter v is-format-item

(59) is-format-iter = (is-rep-factor:is-expr),
   (is-format-list:is-format-list)

(60) is-format-item = is-data-format v is-control-format v is-remote-format

(61) is-data-format = is-real-format v is-cplx-format v is-string-format v is-pic-format

(62) is-real-format = (is-format-type:is-FIX v is-FLT),
   (is-w:is-expr),
   (is-d:is-expr),
   (is-p:is-opt-expr),
   (is-s:is-opt-expr)

(63) is-cplx-format = (is-format-type:is-CPLX),
   (is-real-part:is-real-format v is-pic-format),
   (is-imag-part:is-real-format v is-pic-format)

(64) is-string-format = (is-format-type:is-BIT v is-CHAR),
   (is-w:is-opt-expr)

(65) is-pic-format = (is-format-type:is-PIC),
   (is-pic:is-pic)
12.2.2 Statements

(69) \text{is-proc-st} = \text{is-entry-point} \lor \text{is-st}

(70) \text{is-entry-point} = (\text{<s-id:is-id>},
\text{<s-param-list:is-id-list>},
\text{<s-ret-type:is-ret-type>})

(71) \text{is-st} = (\text{<s-cond-part:is-cond-part>},
\text{<s-label-list:is-id-list>},
\text{<s-prop-st:is-prop-st>})

(72) \text{is-cond-part} = (\text{<s-conv:is-ON} \lor \text{is-NO} \lor \text{is-\textless{\textgreater}},
\text{<s-fofl:is-ON} \lor \text{is-NO} \lor \text{is-\textless{\textgreater}},
\text{<s-ofl:is-ON} \lor \text{is-NO} \lor \text{is-\textless{\textgreater}},
\text{<s-size:is-ON} \lor \text{is-NO} \lor \text{is-\textless{\textgreater}},
\text{<s-strg:is-ON} \lor \text{is-NO} \lor \text{is-\textless{\textgreater}},
\text{<s-subrg:is-ON} \lor \text{is-NO} \lor \text{is-\textless{\textgreater}},
\text{<s-ufl:is-ON} \lor \text{is-NO} \lor \text{is-\textless{\textgreater}},
\text{<s-zdiv:is-ON} \lor \text{is-NO} \lor \text{is-\textless{\textgreater}},
\text{<s-check:is-check-cond-part>})

(73) \text{is-check-cond-part} = (\{\text{<sel(idl):is-ON} \lor \text{is-NO} \mid \text{is-id-list(idl)}\})

(74) \text{is-prop-st} = \text{is-block} \lor \text{is-group} \lor \text{is-proc-st-list} \lor \text{is-if-st} \lor \text{is-goto-st} \lor
\text{is-call-st} \lor \text{is-return-st} \lor \text{is-wait-st} \lor \text{is-delay-st} \lor
\text{is-exit-st} \lor \text{is-stop-st} \lor \text{is-assign-st} \lor \text{is-allocate-st} \lor
\text{is-free-st} \lor \text{is-on-st} \lor \text{is-revert-st} \lor \text{is-signal-st} \lor
\text{is-open-st} \lor \text{is-close-st} \lor \text{is-get-st} \lor \text{is-put-st} \lor \text{is-read-st} \lor
\text{is-write-st} \lor \text{is-rewrite-st} \lor \text{is-locate-st} \lor \text{is-delete-st} \lor
\text{is-unlock-st} \lor \text{is-display-st} \lor \text{is-null-st}

(75) \text{is-null-st} = (\text{<s-st:is-NUL\textless{\textgreater}})
12.2.2.1 Block, group

(76) \text{is-block} = (\text{s-decl-part} : \text{is-decl-part},
\text{s-body-list} : \text{is-body-list},
\text{s-cond-part} : \text{is-cond-part},
\text{s-st-list} : \text{is-st-list})

(77) \text{is-group} = \text{is-while-group} \lor \text{is-contr-group}

(78) \text{is-while-group} = (\text{s-while-expr} : \text{is-expr},
\text{s-st-list} : \text{is-st-list})

(79) \text{is-contr-group} = (\text{s-contr-var} : \text{is-ref},
\text{s-spec-list} : \text{is-do-spec-list},
\text{s-do-list} : \text{is-st-list})

(80) \text{is-do-spec} = (\text{s-init-expr} : \text{is-expr},
\text{s-by-expr} : \text{is-opt-expr},
\text{s-to-expr} : \text{is-opt-expr},
\text{s-while-expr} : \text{is-expr})

12.2.2.2 Flow of control statements

(81) \text{is-if-st} = (\text{s-st} : \text{is-IF},
\text{s-expr} : \text{is-expr},
\text{s-then-st} : \text{is-st},
\text{s-else-st} = \text{is-st} \lor \text{is-\text{Q}})

(82) \text{is-goto-st} = (\text{s-st} : \text{is-GOTO},
\text{s-ref} : \text{is-ref})

(83) \text{is-call-st} = (\text{s-st} : \text{is-CALL},
\text{s-id} : \text{is-id},
\text{s-arg-list} : \text{is-expr-list},
\text{s-pa-opt} : \text{is-pa-opt} \lor \text{is-\text{Q}})

(84) \text{is-pa-opt} = (\text{s-task} : \text{is-ref} \lor \text{is-*},
\text{s-event} : \text{is-ref} \lor \text{is-**},
\text{s-pri} : \text{is-opt-expr})
12.2.2.3 Storage manipulating statements

(90) is-assign-st = (is-st:is-ASSIGN>,
     <is-lp:is-ref-list>,
     <is-rp:is-expr>,
     <is-bbyname:is-opt>)

(91) is-allocate-st = (is-st:is-ALLOCATE>,
     <is-allocate-list:is-allocate-list>)

(92) is-allocate = (is-id:is-id>,
     <is-al:is-al>,
     <is-set-ref:is-opt-ref>,
     <is-area:is-opt-ref>,
     <is-init-set:is-init-set>)

(93) is-al = is-array-al v is-struct-al v is-cell-al v is-string-al v
     is-area-da v is-

(94) is-array-al = (is-lbd:is-extent>,
     <is-ubd:is-extent>,
     <is-elem:is-al>)

(95) is-struct-al = is-al-list

(96) is-cell-al = (is-cell:is-T>,
     <elem(i):is-al>,...)

12.2.2.3
12.2.2.4 Condition handling statements

(97) is-string-al = (<s-base:is-BIT v is-CHAR>,
<\s-length:is-extent>)

(98) is-free-st = (<s-st:is-FREE>,
<\s-free-list:is-free-list>)

(99) is-free = (<s-id:is-id>,
<\s-set-ref:is-opt-ref>,
<\s-area:is-opt-ref>)

(100) is-on-st = (<s-st:is-ON>,
<s-on-cond:is-cond-list>,
<s-snap:is-opt>,
<s-on-unit:is-st v is-SYSTEM>)

(101) is-revert-st = (<s-st:is-REVERT>,
<s-on-cond:is-cond-list>)

(102) is-signal-st = (<s-st:is-SIGNAL>,
<s-on-cond:is-cond-list>)

(103) is-cond = is-CONV v is-FOFL v is-OFL v is-SIZE v is-STRG v is-SUBRG v
is-UFL v is-ZDIV v is-check v is-\^\&L.A v is-named-io-cond v
is-ERROR v is-FINISH v is-progr-named-cond

(104) is-check = (<s-id-list:is-id-list>,
<s-arg-list:is-<->>)

(105) is-named-io-cond = (<s-file:is-id>,
<s-io-cond:is-io-cond>)

(106) is-io-cond = is-ENDFILE v is-ENDPAGE v is-KLY v is-NAME v is-RECORD v
is-TRANSMIT v is-UNDF

(107) is-progr-named-cond = (<s-cond:is-id>)
12.2.2.5 Input and output statements

(108) is-open-st = (<s-st:is-OPEN>,
               <s-open-list:is-open-list>)

(109) is-open = (<s-file:is-id>,
               <s-ident:is-opt-expr>,
               <s-title:is-opt-expr>,
               <s-linesize:is-opt-expr>,
               <s-pagesize:is-opt-expr>,
               <s-open-attr:is-file-attr-set>)

(110) is-close-st = (<s-st:is-CLOSE>,
                    <s-close-list:is-close-list>)

(111) is-close = (<s-file:is-id>,
                  <s-ident:is-opt-expr>)

(112) is-get-st = is-file-get v is-string-get

(113) is-file-get = (<s-st:is-GET>,
                    <s-file:is-id>,
                    <s-spec:is-data-spec-list>,
                    <s-copy:is-opt>,
                    <s-skip:is-opt-expr>)

(114) is-string-get = (<s-st:is-GET>,
                      <s-string:is-ref>,
                      <s-spec:is-data-spec-list>)

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

(116) is-file-put = (<s-st:is-PUT>,
                    <s-file:is-id>,
                    <s-spec:is-data-spec-list>,
                    <s-page:is-opt>,
                    <s-line:is-opt-expr>,
                    <s-skip:is-opt-expr>)

(117) is-string-put = (<s-st:is-PUT>,
                      <s-string:is-ref>,
                      <s-spec:is-data-spec-list>)
(118) is-data-spec = is-data-directed v is-edit-directed v is-list-directed

(119) is-data-directed = (<s-d-data-list:is-item-list>)

(120) is-edit-directed = (<s-data-list:is-item-list>,
               <s-format-list:is-format-list>)

(121) is-list-directed = (<s-data-list:is-item-list>)

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

(123) is-contr-item = (<s-contr-var:is-ref>,
               <s-spec-list:is-do-spec-list>,
               <s-do-list:is-item-list>)

(124) is-read-st = is-into-read v is-set-read v is-ignore

(125) is-into-read = (<s-st:is-READ>,
               <s-file:is-id>,
               <s-into:is-ref>,
               <s-key:is-opt-expr>,
               <s-keyto:is-opt-ref>,
               <s-nolock:is-opt>,
               <s-event:is-opt-ref>)

(126) is-set-read = (<s-st:is-READ>,
               <s-file:is-id>,
               <s-set-ref:is-ref>,
               <s-key:is-opt-expr>,
               <s-keyto:is-opt-ref>)

(127) is-ignore = (<s-st:is-READ>,
               <s-file:is-id>,
               <s-ignore:is-expr>,
               <s-event:is-opt-ref>)

(128) is-write-st = (<s-st:is-WRITE>,
               <s-file:is-id>,
               <s-from:is-ref>,
               <s-keyfrom:is-opt-expr>,
               <s-event:is-opt-ref>)
(129) is-rewrite-st = (<s-st:is-REWRITE>,
            <s-file:is-id>,
            <s-from:is-opt-ref>,
            <s-key:is-opt-expr>,
            <s-event:is-opt-ref>)

(130) is-locate-st = (<s-st:is-LOCATE>,
            <s-file:is-id>,
            <s-id:is-id>,
            <s-set-ref:is-opt-ref>,
            <s-keyfrom:is-opt-expr>)

(131) is-delete-st = (<s-st:is-DELETE>,
            <s-file:is-id>,
            <s-key:is-opt-expr>,
            <s-event:is-opt-ref>)

(132) is-unlock-st = (<s-st:is-UNLOCK>,
            <s-file:is-id>,
            <s-key:is-expr>)

(133) is-display-st = (<s-st:is-DISPLAY>,
            <s-display:is-expr>,
            <s-reply:is-opt-ref>,
            <s-event:is-opt-ref>)

12.2.3 Expressions

(134) is-expr = is-infix-exp r ∨ is-prefix-exp r ∨ is-paren-exp r ∨ is-ref ∨
                is-const ∨ is-isub

(135) is-infix-exp r = (<s-operator:is-infix-operator>,
            <s-op-1:is-exp r>,
            <s-op-2:is-exp r>)

(136) is-infix-operator = is-OK ∨ is-AND ∨ is-GT ∨ is-GL ∨ is-LQ ∨ is-LE ∨ is-LT ∨
                        is-NE ∨ is-CAT ∨ is-ADD ∨ is-SUBL TR ∨ is-MULT ∨ is-DIV ∨
                        is-EXP

(137) is-prefix-exp r = (<s-operator:is-prefix-operator>,
            <s-op:is-exp r>)

12.2.3
(138) is-prefix-operator = is-NOT v is-PLUS v is-MINUS

(139) is-paren-expr = (<s-op:is-expr>)

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

(141) is-opt-expr = is-expr v is-

12.2.3.1 References

(142) is-ref = is-ptr-qual-ref v is-basic-ref

(143) is-ptr-qual-ref = (<s-ptr-qual:is-ref>,
    <s-id-list:is-id-list>,
    <s-arg-list:is-arg-expr-list>)

(144) is-basic-ref = (<s-id-list:is-id-list>,
    <s-arg-list:is-arg-expr-list>)

(145) is-arg-expr = is-expr v is-

(146) is-simple-ref = (<(elem(1)):is-id-list:is-id>,
    <s-arg-list:is-<>>) 

(147) is-opt-ref = is-ref v is-

12.2.3.2 Constants

(148) is-const = is-arithm-const v is-string-const

(149) is-arithm-const = (<s-da:is-arithm>,
    <s-vr:is-vr>)

(150) is-string-const = (<s-da:is-fixed-string-eda>,
    <s-vr:is-vr>)

(151) is-fixed-string-eda = (<s-base:is-5IT v is-CHAR>,
    <s-length:is-intg-val>)

Ref.: is-vr 2-22(69)
12.3 Auxiliary Functions and Predicates on Data Attributes

This section defines some predicates and functions frequently used for manipulation of data attributes. Though they do not describe abstract syntax of program text itself, most of them are closely related to objects occurring in the program text.

12.3.1 Evaluated data attributes

The following predicates describe evaluated data attributes as produced by the instruction eval-da (cf. 5.1.6) from data attributes as defined in 12.2.1.1.

(152) is-eda = is-array-eda v is-struct-eda v is-cell-eda v is-scalar-eda

(153) is-array-eda = (<s-lbd:is-intg-val v is-*>,
                      <s-ubd:is-intg-val v is-*>,
                      <s-elem:is-eda>)

(154) is-struct-eda = is-succ-eda-list

(155) is-succ-eda = (<s-da:is-eda>)

(156) is-cell-eda = (<s-cell:is-T>,
                      <elem(1):is-succ-eda>, ...)

(157) is-scalar-eda = is-arithm v is-string-eda v is-pic v is-area-eda v
                      is-LABEL v is-PTR v is-OFFSET v is-TASK v is-EVENT

(158) is-string-eda = (<s-base:is-BIT v is-CHAR>,
                      <s-length:is-intg-val v is-*>,
                      <s-varying:is-opt>)

(159) is-area-eda = is-AREA v is-spec-area-eda

(160) is-spec-area-eda = (<s-size:is-intg-val v is-*>)
12.3.2 Summarizing predicates

The following predicates are defined to have common terms for different, but similar predicates, especially for unevaluated and evaluated data attributes.

(161) \text{is-var} = \text{is-prop-var} \lor \text{is-defined} \lor \text{is-based}

(162) \text{is-da} = \text{is-named-da} \lor \text{is-da-descr} \lor \text{is-eda}

(163) \text{is-array} = \text{is-named-array} \lor \text{is-array-descr} \lor \text{is-array-eda}

(164) \text{is-struct} = \text{is-named-struct} \lor \text{is-struct-descr} \lor \text{is-struct-eda}

(165) \text{is-cell} = \text{is-named-cell} \lor \text{is-cell-descr} \lor \text{is-cell-eda}

(166) \text{is-scalar} = \text{is-scalar-da} \lor \text{is-scalar-eda}

(167) \text{is-string} = \text{is-string-da} \lor \text{is-string-eda}

(168) \text{is-area} = \text{is-area-da} \lor \text{is-area-eda}

(169) \text{is-spec-area} = \text{is-spec-area-da} \lor \text{is-spec-area-eda}

12.3.3 Functions on data attributes

The following functions are often used for manipulation of data attributes.

(170) \text{order(da)} = 

\text{length}(\emptyset(da; \text{s-cell}))

for: \text{is-struct(da)} \lor \text{is-cell(da)} \lor \text{is-struct-al(da)} \lor \text{is-cell-al(da)}

(171) \text{dim(da)} =

\text{is-} \emptyset \ast \text{s-elem(da)} \rightarrow 0

\text{T} \rightarrow 1 + \text{dim} \ast \text{s-elem(da)}

for: \text{is-da(da)} \lor \text{is-eda(da)} \lor \text{is-al(da)}
(172) \( \text{lb}(\text{eda}) = \)

\[
\begin{align*}
\text{is-array(eda)} & \rightarrow \text{s-lb}(\text{eda}) \\
\text{T} & \rightarrow 1
\end{align*}
\]

for: is-eda(eda)

(173) \( \text{ub}(\text{eda}) = \)

\[
\begin{align*}
\text{is-array(eda)} & \rightarrow \text{s-ub}(\text{eda}) \\
\text{is-struct(eda)} \lor \text{is-cell(eda)} & \rightarrow \text{order(eda)} \\
\text{is-string(eda)} & \rightarrow \text{s-length(eda)} \\
\text{is-scalar(eda)} & \rightarrow 1
\end{align*}
\]

for: is-eda(eda)

(174) \( \text{da-part}(d,a,i) = \)

\[
\begin{align*}
\text{is-array}(d,a) & \rightarrow \text{s-elem}(d,a) \\
\text{is-struct}(d,a) \lor \text{is-cell}(d,a) & \rightarrow \text{elem}(i,d,a) \\
\text{is-string}(d,a) & \rightarrow \mu_\emptyset(\langle \text{s-base}(d,a)>, \\
& \langle \text{s-length}:1\rangle) \\
\text{is-scalar}(d,a) & \rightarrow d
\end{align*}
\]

for: is-da(d,a) \lor is-eda(d,a), is-intg-val(i)

(175) \( \text{extents-set}(d,a) = \)

\[
\{ \chi(d,a) \mid \neg\text{is-}Q\ast\chi(d,a) \land (\exists \chi-1)(\chi = \text{s-lb}\ast\chi-1 \lor \chi = \text{s-ub}\ast\chi-1 \lor \\
\chi = \text{s-length}\ast\chi-1 \lor \chi = \text{s-size}\ast\chi-1) \}
\]

for: is-da(d,a)
APPENDIX I: CROSS-REFERENCE INDEX

This index lists all names used in the document, with the exception of names of selectors (i.e., names prefixed by $s$-), meta-variables and abbreviations. 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. For names used in the abbreviations part of a chapter the character A is written instead of the formula number. 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 or in an abbreviation are given. Multiple occurrences within a formula are indicated by multiple reference to the formula. The defining formula is indicated by an underlined reference.

There are names of functions and instructions which are implementation defined, or which are only partially defined by a number of logical statements. For these cases either the reference to a place in the text describing these functions or instructions is given, or the reference to a defining statement.

(3) Occurrences of names of the form pass-function (cf. 1.4) are listed under the entry function.

Occurrences of names of the form is-pred-list (cf. 12.1) are listed under the entry is-pred.

Occurrences of names of the form is-pred-set (cf. 12.1) are listed under the entry is-pred.

Occurrences of names of the form is-OBJ (cf. 12.1) are listed under the entry OBJ.

Occurrences of names defined by the function and predicate schemes in chapter 10.5.2.3 are listed under the entries of the corresponding predicates.

(4) Names of elementary objects (having no defining formula) are listed also in Appendix II.
A-CHAR                      8-4 (11), 8-59 (225), 12-9 (57)
ABNL                        4-9 (17), 4-10 (24), 4-12 (30), 4-20 (49), 10-22 (56)
abs                         1-111111
acc-infix-num                8-16 (71), 8-17 (72)
acc-prefix-num               8-28 (112), 8-29 (113)
activate-tasks               2-12 (25), 4-11 (25), 4-12 (30), 4-18941, 4-20 (49), 7-9 (23), 7-16 (37),
                             10-33 (91), 10-34 (92)
activate-tasks-1            10-33 (90), 10-33 (91), 10-107 (291)
active-descendants          1-10 (19), 4-10 (20), 4-10 (21), 4-10 (22)
ADD                          1-5 (8), 5-25 (44), 8-12 (47), 8-18 (72), 8-25 (102), 8-26 (103), 10-101 (276),
                             11-37 (75), 11-45 (133), 11-46 (144), 12-16 (136)
add-op-expr                  10-68 (192), 10-92 (254), 10-101 (275), 10-102 (276)
adjust-char-pic              10-27 (70), 10-27 (71)
adjust-str                   10-26 (68), 10-27 (70), 10-108 (293)
adjust-string                8-9 (33), 8-10 (36), 8-15 (67), 8-18 (78), 8-18 (79), 8-51 (176),
                             11-22 (130), 11-22 (130)
adjust-substr                10-27 (70), 10-27 (72)
AGGR                         11-8 (17), 11-10 (23), 11-18 (35)
aggr-part                    7-6 (11), 7-6 (11), 7-6 (11)
ALIGNED                      5-44 (79), 5-55 (103), 12-5 (13)
alloc-assign                 7-11 (23), 7-11 (24)
alloc-space                  3-9 (12), 5-11 (13), 5-44 (80), 6-4 (4), 6-5 (9), 6-8 (13), 6-11 (19), 6-11 (19),
                             6-19 (43), 6-19 (43)
alloc-state                  2-23 (73), 2-23 (74), 2-24 (79), 2-24 (79), 2-25 (85), 2-25 (85), 2-25 (88),
                             2-25 (89), 2-26 (91), 3-4 (4), 6-5 (8), 6-9 (15), 6-11 (19), 6-13 (25), 6-19 (43),
                             6-19 (44), 7-10 (21), 7-11 (23), 11-49 (163)
ALLOCATE                     2-33 (108), 12-12 (91)
alocate                      6-3 (2), 6-4 (4)
alocater-1                   5-43 (77), 5-44 (80), 7-20 (49), 7-20 (49), 9-17 (34)
allst-part                   2-24 (78), 2-24 (78), 2-24 (80), 2-25 (86), 2-25 (89)
AND                          8-4 (9), 8-12 (49), 8-21 (87), 11-45 (140), 12-16 (136)
APOSTR                       8-4 (9), 10-56 (145), 10-56 (145), 10-56 (146), 10-56 (146), 10-57 (159),
                             10-57 (160), 10-57 (160), 10-72 (204), 10-98 (266), 10-99 (266), 10-98 (266),
                             10-98 (266), 10-98 (266), 10-98 (266)
AR-DA

AREA

area-alloc-space

area-allocate

area-assign

area-assign-1

area-free

area-initialize

arg-check-list

arg-class

arg-class-1

arg-da

arg-da-1

arg-test

arg-type

ARR

array-elem

ASSIGN

assign

assign-completion

assign-complex

assign-imag

assign-uchar

assign-priority

assign-priority-1

assign-real

assign-status
assign-substr
assign-substr-1
assign-substr-2
assign-unassoc
assoc-csa
assoc-idl
assoc-idl-set
ASTER
ATTAN
ATAND
attach-io-event
attach-task
attr-arg
attr-expr
AUTO
auto-allocate
auto-free
B-CHAR
BACK
base-elem
based-allocate
based-free
BIN
bintg-bit-conv
bintg-da
BINTG-DA
bintg-op
<p>| BIT       | 6-11(24), 6-11(24), 7-12(29), 7-12(29), 7-30(76), 8-3(2), 8-5(15), 8-10(37), 8-13(59), 8-13(61), 8-15(67), 8-15(67), 8-20(82), 8-22(91), 8-22(91), 8-24(99), 8-24(99), 8-24(99), 8-31(120), 8-31(120), 8-31(120), 8-37(143), 8-38(149), 8-38(150), 8-39(153), 8-39(153), 8-39(156), 8-21(51), 8-69(195), 8-81(227), 8-82(229), 8-82(229), 10-29(259), 10-97(265), 10-97(265), 10-98(266), 10-98(266), 10-98(266), 10-102(280), 10-103(281), 10-103(281), 11-20(39), 11-30(41), 11-31(44), 11-44(134), 11-45(136), 11-65(136), 12-7(39), 12-9(64), 12-13(97), 12-17(151), 12-18(158) |
| bit-char-conv | 8-31(120), 8-36(138), 8-59(226), 10-103(281) |
| bit-da      | 8-13(51), 9-25(101), 10-72(204), 10-83(230), 11-19(37), 11-20(39), 11-20(39), 11-50(167), 11-52(176), 11-53(178) |
| BIT-DA     | 5-20(31), 8-13(51), 8-22(91), 8-29(117), 11-19(37), 11-42(120), 11-42(120) |
| bit-length  | 8-34(129), 8-39(153), 8-39(154) |
| bit-num-conv | 8-31(120), 8-35(136) |
| bit-op      | 10-82(229), 10-83(230) |
| BLANK      | 8-4(9), 8-10(37), 8-32(124), 8-32(124), 8-32(124), 8-32(124), 8-49(193), 8-49(193), 8-49(193), 8-55(210), 8-56(216), 8-56(216), 8-57(217), 8-59(225), 8-59(225), 10-60(170), 10-61(174), 10-61(174), 10-72(201), 10-75(209), 10-77(220), 10-82(229), 10-82(229), 10-84(232), 10-84(233), 10-84(233), 10-84(235), 10-84(235), 10-98(266), 10-98(266), 10-98(266), 10-100(272), 10-105(285), 11-47(151), 12-8(51) |
| BLOCK      | 2-15(31), 5-14(19), 5-53(98) |
| block-ei   | 5-3(1), 5-14(19) |
| bool       | 11-44(135), 11-46(136) |
| BREAK      | 8-4(10) |
| buffer-allocate | 10-28(176), 10-45(118), 10-47(121) |
| buffer-into | 10-29(81), 10-45(118) |
| buffer-output | 10-42(112), 10-43(113) |
| buffer-seq  | 10-28(178), 10-40(108), 10-43(113) |
| buffer-space | 6-13(25), 6-13(25), 6-13(25), 6-13(25), 10-28(76), 10-28(76), 10-46(120), 10-46(120), 10-47(121) |
| buffered-set | 6-13(25), 6-13(25), 6-13(25), 6-13(25), 10-28(76), 10-28(76), 10-46(120), 10-46(120), 10-47(121) |
| BUILTIN    | 2-18(40), 5-9(12), 5-38(64), 5-38(64), 5-38(64), 5-46(85), 7-24(58), 9-16(31), 9-16(31), 11-2(1), 11-4(4), 11-8(16), 12-5(9) |
| builtin-arg | 11-4(6), 11-5(12) |</p>
<table>
<thead>
<tr>
<th>Command</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>comment</td>
<td>9-4 (1), 9-4 (1), 9-4 (1), 9-2 (12)</td>
</tr>
<tr>
<td>comment-f</td>
<td>9-4 (2), 9-4 (13), 9-4 (4)</td>
</tr>
<tr>
<td>comp-operator</td>
<td>5-24 (40), 5-24 (41)</td>
</tr>
<tr>
<td>compl-event</td>
<td>4-16 (39), 4-19 (45)</td>
</tr>
<tr>
<td>compl-event-1</td>
<td>4-19 (45), 4-20 (48), 5-28 (47)</td>
</tr>
<tr>
<td>compl-val</td>
<td>8-53 (202), 8-54 (205)</td>
</tr>
<tr>
<td>complete-ar</td>
<td>8-37 (143), 8-37 (149)</td>
</tr>
<tr>
<td>complete-m-b-s</td>
<td>8-37 (144), 8-37 (145)</td>
</tr>
<tr>
<td>complete-prec</td>
<td>8-37 (144), 8-37 (146)</td>
</tr>
<tr>
<td>complete-str</td>
<td>8-37 (143), 8-38 (148)</td>
</tr>
<tr>
<td>complete-tg</td>
<td>5-43 (78), 8-22 (88), 8-22 (88), 8-26 (105), 8-26 (105), 8-27 (106), 8-29 (116), 8-30 (118), 8-32 (125), 8-37 (143), 8-39 (152), 11-10 (23), 11-10 (23), 11-31 (48), 11-52 (174)</td>
</tr>
<tr>
<td>completion</td>
<td>7-12 (27), 7-12 (29)</td>
</tr>
<tr>
<td>complex</td>
<td>7-12 (27), 7-12 (29)</td>
</tr>
<tr>
<td>conv</td>
<td>8-52 (198), 9-15 (29), 9-16 (30), 11-6 (13)</td>
</tr>
<tr>
<td>conv-gen-list</td>
<td>4-15 (33), 9-16 (39)</td>
</tr>
<tr>
<td>cond</td>
<td>2-18 (39), 5-9 (12), 5-55 (102), 9-6 (9), 12-5 (9)</td>
</tr>
<tr>
<td>cond-dyn-sel</td>
<td>2-30 (103), 2-31 (104), 5-7 (9), 5-7 (9), 9-6 (11), 9-7 (14), 9-12 (19), 9-12 (19), 9-12 (21), 9-12 (21), 9-13 (22)</td>
</tr>
<tr>
<td>cond-infix</td>
<td>8-16 (4), 8-16 (59)</td>
</tr>
<tr>
<td>cond-io-sel</td>
<td>2-30 (100), 2-30 (100), 2-31 (101)</td>
</tr>
<tr>
<td>cond-sel</td>
<td>2-29 (92), 2-30 (100), 2-30 (100), 2-30 (100), 5-7 (9), 5-7 (9)</td>
</tr>
<tr>
<td>conn-refer</td>
<td>7-32 (85), 7-33 (85), 7-33 (86), 7-33 (86)</td>
</tr>
<tr>
<td>const</td>
<td>11-8 (17), 11-18 (35)</td>
</tr>
<tr>
<td>conv</td>
<td>2-29 (99), 2-31 (103), 9-4 (1), 9-8 (15), 9-8 (15), 9-8 (15), 9-12 (19), 9-17 (34), 12-13 (103)</td>
</tr>
<tr>
<td>conv-scale-f</td>
<td>8-32 (125), 8-33 (126)</td>
</tr>
<tr>
<td>convert</td>
<td>4-4 (5), 4-5 (8), 5-20 (31), 7-9 (19), 7-12 (29), 7-12 (29), 7-12 (29), 7-12 (29), 7-14 (34), 7-14 (34), 7-15 (36), 7-15 (36), 7-21 (52), 8-14 (65), 8-14 (65), 8-27 (108), 8-30 (118), 10-12 (30), 10-92 (256), 10-92 (256), 10-97 (265), 10-101 (276), 10-102 (280), 11-5 (8), 11-13 (29), 11-35 (61), 11-35 (61), 11-36 (65), 11-36 (67), 11-37 (72), 11-37 (72)</td>
</tr>
<tr>
<td>da-part</td>
<td>6-11(21), 6-14(27), 6-14(28), 6-14(28), 7-5(7), 7-30(76), 7-30(77), 7-30(80), 7-30(80), 7-33(86), 7-33(88), 10-68(191), 10-92(255), 12-2011741</td>
</tr>
<tr>
<td>da-part-list</td>
<td>7-5(6), 7-5(17)</td>
</tr>
<tr>
<td>da-ref</td>
<td>7-3(1), 7-5(8), 7-5(19), 7-5(9)</td>
</tr>
<tr>
<td>da-type</td>
<td>8-31(120), 8-31(120), 8-31(120), 8-31(120), 8-31(120), 8-31(120), 8-31(120), 8-31(120), 8-31(120), 8-31(120), 8-31(120), 8-31(120), 8-31(120), 8-31(120)</td>
</tr>
<tr>
<td>datadir-idl-set</td>
<td>10-66(185), 10-66(185), 10-67(186), 10-91(253)</td>
</tr>
<tr>
<td>datadir-n-set</td>
<td>10-67(187), 10-67(187), 10-89(249)</td>
</tr>
<tr>
<td>datadir-ref-parsing</td>
<td>10-59(169)</td>
</tr>
<tr>
<td>datadir-rp-parsing</td>
<td>10-59(169), 10-77(219)</td>
</tr>
<tr>
<td>DEC</td>
<td>8-1(A), 8-12(50), 8-23(96), 8-23(96), 8-32(125), 8-36(141), 8-38(147), 8-38(147), 8-39(152), 8-64(239), 10-73(206), 10-102(280), 11-20(39), 11-30(41), 11-31(44), 11-37(73), 12-7(38)</td>
</tr>
<tr>
<td>decipher</td>
<td>2-41(135), 2-41(137), 2-42(139), 2-42(140), 2-43(142), 10-7(18), 10-17(39), 10-19(46)</td>
</tr>
<tr>
<td>def-copy</td>
<td>10-86(243), 10-86(249)</td>
</tr>
<tr>
<td>DEF-ENVIRONMENT</td>
<td>10-8(A)</td>
</tr>
<tr>
<td>DEF-LINESIZE</td>
<td>10-20(48)</td>
</tr>
<tr>
<td>DEF-PAGESIZE</td>
<td>10-20(49)</td>
</tr>
<tr>
<td>def-prec</td>
<td>11-31(42), 11-31(42), 11-31(43)</td>
</tr>
<tr>
<td>DEP-PREC-BIN</td>
<td>11-31(42), 11-31(42), 11-31(43)</td>
</tr>
<tr>
<td>DEP-PREC-DEC</td>
<td>8-12(53)</td>
</tr>
<tr>
<td>default-attr-set</td>
<td>8-12(52)</td>
</tr>
<tr>
<td>DEL-KEYED</td>
<td>10-9(A), 10-13(33)</td>
</tr>
<tr>
<td>DEL-UNBUF</td>
<td>10-33(92), 10-48(126)</td>
</tr>
<tr>
<td>DELAY</td>
<td>10-33(92), 10-48(126)</td>
</tr>
<tr>
<td>delete</td>
<td>10-48(126), 10-92(127), 10-50(130)</td>
</tr>
<tr>
<td>DELETE</td>
<td>10-48(126), 10-92(127), 10-50(130)</td>
</tr>
<tr>
<td>delete-buffer-event</td>
<td>2-33(108), 12-12(87)</td>
</tr>
<tr>
<td>delete-event</td>
<td>10-22(55), 10-22(55)</td>
</tr>
<tr>
<td>delete-kered</td>
<td>4-10(24), 4-20(48), 8-20(142), 10-22(56)</td>
</tr>
<tr>
<td>delete-kered-1</td>
<td>10-33(92), 10-49(128)</td>
</tr>
<tr>
<td>delete-kered-2</td>
<td>10-41(109), 10-49(128), 10-49(129)</td>
</tr>
<tr>
<td>Function</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------</td>
<td>------------------------------</td>
</tr>
<tr>
<td>delete-task</td>
<td></td>
</tr>
<tr>
<td>delete-nd</td>
<td></td>
</tr>
<tr>
<td>dense-xpr</td>
<td></td>
</tr>
<tr>
<td>dense-match</td>
<td></td>
</tr>
<tr>
<td>dig-index</td>
<td></td>
</tr>
<tr>
<td>dig-pos</td>
<td></td>
</tr>
<tr>
<td>digit-list</td>
<td></td>
</tr>
<tr>
<td>dim</td>
<td></td>
</tr>
<tr>
<td>dir-match</td>
<td></td>
</tr>
<tr>
<td>direct-into</td>
<td></td>
</tr>
<tr>
<td>DISPLAY</td>
<td></td>
</tr>
<tr>
<td>display-message</td>
<td></td>
</tr>
<tr>
<td>DIV</td>
<td></td>
</tr>
<tr>
<td>do-continue</td>
<td></td>
</tr>
<tr>
<td>DOLLAR</td>
<td></td>
</tr>
<tr>
<td>qr-set</td>
<td></td>
</tr>
<tr>
<td>drift-pv</td>
<td></td>
</tr>
<tr>
<td>DUMMY</td>
<td></td>
</tr>
<tr>
<td>dummy-arq</td>
<td></td>
</tr>
<tr>
<td>dummy-assign</td>
<td></td>
</tr>
<tr>
<td>dummy-dens</td>
<td></td>
</tr>
<tr>
<td>dummy-eda</td>
<td></td>
</tr>
<tr>
<td>E-CHAR</td>
<td></td>
</tr>
<tr>
<td>e-data-parsing</td>
<td></td>
</tr>
<tr>
<td>EDA</td>
<td></td>
</tr>
<tr>
<td>eda-builtin</td>
<td></td>
</tr>
<tr>
<td>eda-builtin-1</td>
<td></td>
</tr>
</tbody>
</table>
IBMLAB VIENNA
TR 25.082

ep .......................... 5-50(90), 5-50(90), 5-50(91)
epilogue ..................... 4-9 (17), 5-3 (1), 5-14 (29), 5-28 (47), 5-28 (47), 5-28 (47), 5-53 (98), 5-53 (98), 9-13 (24)
EO .......................... 8-4 (9), 8-12 (49), 8-18 (78), 8-18 (78), 8-18 (78), 8-19 (81), 8-20 (82), 10-60 (170), 10-74 (207), 10-98 (268), 12-16 (136)
equal-intg .......................... 5-40 (67), 5-40 (67), 5-40 (68)
erase-buffer .................... 10-29 (77), 10-42 (112), 10-45 (118), 10-47 (121), 10-48 (126)
error .......................... 1-20 (123)
ERROR .......................... 2-29 (99), 8-16 (69), 9-4 (1), 9-4 (1), 9-11 (18), 9-12 (19), 10-77 (219), 10-78 (222), 10-96 (261), 12-13 (103)
error .......................... 1-20 (123)
es-dataset ...................... 10-4 (A), 10-5 (A), 10-9 (A)
es-descr .......................... 10-4 (101), 10-5 (A), 10-9 (A)
es-ds-label .......................... 10-4 (191), 10-5 (A)
ev-vr .......................... 4-6 (10), 8-21 (121), 4-20 (49), 10-32 (89), 10-107 (290)
eval-abs .......................... 11-21 (40), 11-21 (40), 11-21 (40), 11-21 (40), 11-21 (40), 11-21 (40), 11-21 (40), 11-21 (40), 11-21 (40), 11-21 (40)
eval-add .......................... 11-21 (40), 11-21 (40), 11-21 (40), 11-21 (40), 11-21 (40), 11-21 (40), 11-21 (40), 11-21 (40), 11-21 (40), 11-21 (40)
eval-addr .......................... 11-21 (40), 11-21 (40), 11-21 (40), 11-21 (40), 11-21 (40), 11-21 (40), 11-21 (40), 11-21 (40), 11-21 (40), 11-21 (40)
eval-all .......................... 11-21 (40), 11-21 (40), 11-21 (40), 11-21 (40), 11-21 (40), 11-21 (40), 11-21 (40), 11-21 (40), 11-21 (40), 11-21 (40)
eval-allocation .................. 11-21 (40), 11-21 (40), 11-21 (40), 11-21 (40), 11-21 (40), 11-21 (40), 11-21 (40), 11-21 (40), 11-21 (40), 11-21 (40)
eval-any .......................... 11-21 (40), 11-21 (40), 11-21 (40), 11-21 (40), 11-21 (40), 11-21 (40), 11-21 (40), 11-21 (40), 11-21 (40), 11-21 (40)
eval-arg .......................... 5-36 (58), 5-36 (58), 5-36 (58), 5-36 (58)
eval-arg-list .................... 3-5 (6), 5-33 (52), 5-36 (58), 5-36 (58), 5-36 (58)
eval-atan .......................... 11-21 (40), 11-21 (40), 11-21 (40), 11-21 (40), 11-21 (40), 11-21 (40), 11-21 (40), 11-21 (40), 11-21 (40), 11-21 (40)
eval-atan-complex .......................... 11-49 (103), 11-49 (103), 11-49 (103), 11-49 (103)
eval-atan-real .......................... 11-49 (103), 11-49 (103), 11-49 (103), 11-49 (103)
eval-atan-2 .......................... 11-49 (103), 11-49 (103), 11-49 (103)
eval-atan-2d .......................... 11-49 (103), 11-49 (103), 11-49 (103), 11-49 (103)
eval-atan-1 .......................... 11-39 (106), 11-41 (118)
eval-atan-2e .......................... 11-39 (106), 11-41 (118)


eval descr-list 5-9(12), 5-11(14), 5-51(92)

eval dim 11-21(40), 11-47(147), 11-47(147)

eval dir-arg 5-36(59), 5-38(59)

eval divide 11-21(40), 11-47(147)

eval dummy-arg 5-36(59), 5-43(77)

eval empty 11-21(40), 11-47(163)

eval entry-den 5-9(12), 5-12(161)

eval erf 11-21(40), 11-39(191)

eval erf ce 11-21(40), 11-39(191)

eval exp 11-21(40), 11-39(191)

eval exp 4-4(5), 5-20(31), 5-23(37), 5-45(81), 5-53(97), 5-53(97), 5-53(97), 5-6(17), 5-4(3), 7-6(10), 7-18(43), 7-18(43), 7-18(43), 7-18(43), 7-18(43), 7-18(43), 7-21(52), 7-21(53), 10-12(30), 10-92(256), 11-5(8), 11-13(29), 11-13(29), 11-42(122), 11-42(123), 11-51(171)

eval fixed 11-21(40), 11-36(151)

eval float 11-21(40), 11-36(151)

eval floor 11-21(40), 11-36(151)


eval function 7-19(46), 7-19(46)

eval gen 7-24(61), 7-25(60)

eval gen-arg 5-36(59), 5-36(59)

eval gen-arg-l 5-41(71), 5-41(71)

eval generic 7-19(46), 7-19(46)

eval hbound 11-21(40), 11-36(146), 11-46(146)

eval high 11-21(40), 11-43(129)

eval ident 10-12(25), 10-12(25), 10-22(54)

eval isag 11-21(40), 11-38(121), 11-38(121)

eval index 11-21(40), 11-43(125), 11-43(125)

eval-infix-conver 11-37(75), 11-37(76), 11-37(77), 11-37(78)

eval-infix-exp 4-5(8), 5-24(40), 5-24(40), 5-25(44), 7-18(43), 8-14(65), 10-101(276), 11-46(144), 11-46(144), 11-53(179)

eval-inte-exp 4-15(33), 4-21(50), 5-13(18), 5-13(18), 5-13(18), 6-17(37), 7-12(27), 7-21(52), 7-26(62), 10-12(25), 10-12(25), 10-47(122), 10-53(138), 10-53(138), 10-64(179), 10-88(245), 10-88(245), 11-6(14), 11-8(17)
eval-bound

11-21(40), 11-46(145), 11-49(195)

eval-length

11-21(40), 11-49(197)

eval-lineno

11-21(40), 11-49(198)

eval-log

11-21(40), 11-40(113)

eval-log-complex

11-39(191), 11-40(113)

eval-log-real

11-39(197), 11-40(113)

eval-log-10

11-21(40), 11-40(114)

eval-log-10-1

11-39(193), 11-40(114)

eval-log-2

11-21(40), 11-40(115)

eval-log-2-1

11-39(192), 11-40(115)

eval-low

11-21(40), 11-39(129)

eval-up

5-23(36), 7-4(4), 7-9(59), 10-12(29), 10-68(190), 10-77(219), 10-88(245)

eval-up-list

7-4(3), 7-24(36), 7-4(4), 7-6(10)

eval-max

11-21(40), 11-34(127)

eval-max-1

11-34(57), 11-35(58), 11-35(128)

eval-min

11-21(40), 11-35(129)

eval-min-1

11-35(59), 11-35(129), 11-35(161)

eval-mod

11-21(40), 11-35(161)

eval-multiply

11-21(40), 11-37(161)

eval-null

11-21(40), 11-49(169)

eval-nulloc

11-21(40), 11-49(169)

eval-offset

11-21(40), 11-49(159)

eval-onchar

11-21(40), 11-57(151)

eval-oncode

11-21(40), 11-48(159)

eval-oncount

11-21(40), 11-48(159)

eval-onfile

11-21(40), 11-47(149)

eval-onkey

11-21(40), 11-49(152)

eval-onloc

11-21(40), 11-47(149)

eval-onresource

11-21(40), 11-47(150)

eval-opt-exit

5-24(38), 5-24(38), 7-21(153)
| eval-paran-built-in | 11-4 (4), 11-12 (27) |
| eval-pb-arg | 11-13 (28), 11-13 (28), 11-13 (28), 11-13 (28), 11-13 (29) |
| eval-pb-arg-list | 11-12 (27), 11-12 (28) |
| eval-poly | 11-21 (40), 11-46 (133) |
| eval-poly-1 | 11-46 (143), 11-46 (144), 11-46 (193) |
| eval-proc | 11-21 (40), 11-37 (191) |
| eval-prefix-exp | 7-18 (43), 8-27 (108) |
| eval-pri | 4-3 (1), 4-5 (7) |
| eval-pri-1 | 4-3 (2), 9-5 (8) |
| eval-pri-2 | 4-5 (7), 8-5 (8), 7-16 (39) |
| eval-priority | 11-21 (40), 11-53 (129) |
| eval-prod | 11-21 (40), 11-45 (138) |
| eval-prop-gen | 7-24 (51), 7-27 (167) |
| eval-ps-le | 7-4 (5), 7-12 (27) |
| eval-ptr | 11-21 (40), 11-49 (159) |
| eval-ptr-ref | 7-24 (61), 7-31 (82) |
| eval-ql | 5-26 (46), 5-35 (A), 6-16 (36), 7-2 (A), 7-27 (66), 7-27 (66), 9-16 (31), 10-91 (253), 10-92 (256) |
| eval-real | 11-21 (40), 11-38 (81), 11-38 (83) |
| eval-ref | 5-26 (45), 7-18 (43), 7-19 (46), 10-53 (138), 10-64 (179) |
| eval-ref-gen | 4-4 (3), 4-15 (33), 5-41 (71), 6-3 (2), 6-8 (14), 6-8 (14), 6-9 (15), 6-18 (41), 7-4 (5), 7-12 (27), 7-12 (27), 7-14 (34), 7-19 (46), 7-24 (61), 7-25 (59), 7-25 (60), 7-31 (82), 7-31 (82), 10-38 (101), 10-43 (114), 11-5 (7) |
| eval-repeat | 11-21 (40), 11-44 (130), 11-44 (131), 11-44 (131) |
| eval-repeat-1 | 11-44 (130), 11-44 (131), 11-44 (131) |
| eval-round | 11-21 (40), 11-51 (171) |
| eval-round-1 | 11-51 (171), 11-51 (172), 11-51 (172), 11-51 (172), 11-51 (172) |
| eval-set-gen | 6-3 (2), 6-3 (2), 6-8 (119), 6-18 (41), 6-18 (41), 10-42 (111), 10-46 (119) |
| eval-sig | 11-21 (40), 11-36 (54) |
| eval-sin | 11-21 (40), 11-39 (94) |
IBM LAB VIENNA

eval-sind

eval-sinh

eval-s1

10-92 (256), 10-93 (257), 10-93 (257)

eval-sqrt

11-21 (40), 11-39 (103)

eval-sqrt-colx

11-39 (103), 11-40 (116)

eval-sqrt-real

11-39 (103), 11-40 (116)

eval-status

11-21 (40), 11-53 (177)

eval-string

11-21 (40), 11-53 (177)

eval-sub-gen

7-25 (59), 7-25 (60), 10-92 (256)

eval-substr

11-21 (40), 11-42 (122)

eval-substr-1

11-42 (122), 11-42 (123)

eval-substr-2

11-42 (123), 11-94 (124)

eval-sum

11-21 (40), 11-45 (138)

eval-tan

11-21 (40), 11-39 (123)

eval-tand

11-21 (40), 11-39 (123)

eval-tanh

11-21 (40), 11-39 (123)

eval-te-gen

4-4 (2), 4-4 (2), 4-4 (2)

eval-time

11-21 (40), 11-59 (166)

eval-trunc

11-21 (40), 11-36 (70)

eval-truth

5-20 (29), 5-20 (31), 5-21 (34), 5-24 (42)

eval-unspec

11-21 (40), 11-49 (133)

eval-while

5-24 (39), 5-24 (42)

EVENT

3-3 (3), 4-4 (2), 4-7 (12), 4-16 (35), 7-9 (20), 7-16 (37), 7-16 (37), 7-16 (37), 7-16 (37), 7-16 (37), 7-16 (37), 8-3 (3), 8-5 (15), 10-32 (88), 10-107 (290), 11-19 (37), 12-7 (37), 12-18 (157)

event-reply

10-107 (290), 10-107 (290)

event-transmission

10-32 (89), 10-33 (90)

RXCL


eexec-time

2-10 (15), 2-12 (28), 2-12 (27), 2-12 (27)

eexec-locate

10-22 (56), 10-22 (56), 10-38 (102), 10-42 (111), 10-42 (112)
<p>| EXIT   | 2-33(108), 12-12(88) |
| EXP   | 8-12(47), 8-16(69), 8-16(71), 8-17(72), 8-22(89), 8-22(90), 8-25(101), 8-26(104), 8-27(106), 8-27(107), 12-16(136) |
| exp-sep | 8-58(221), 9-59(222) |
| EXP-SIZE | 8-39(152), 8-60(229), 10-105(284), 10-105(284) |
| expand-basic-ref | 10-74(207), 10-75(210) |
| expand-idl | 9-16(31), 2-16(32), 9-16(32), 9-16(32), 9-10(251) |
| expand-part | 10-75(212), 10-76(213), 10-76(216) |
| expand-ref | 9-15(29), 9-16(31), 9-16(31) |
| expand-reflist | 9-5(7), 9-7(12), 9-15(27), 9-15(29), 9-16(31) |
| EXPND | 11-2(2), 11-2(2), 11-8(17), 11-9(18), 11-9(18), 11-9(18), 11-9(18), 11-9(18), 11-10(20), 11-10(21), 11-10(23), 11-18(35) |
| EXT | 2-30(100), 3-7(8), 3-7(9), 3-7(10), 3-10(16), 5-12(16), 10-9(8), 12-4(3), 12-5(11) |
| extents-set | 3-8(11), 5-9(12), 5-47(87), 12-20(175) |
| F-CHAR | 8-4(11), 8-33(126) |
| f-data-parsing | 10-59(169), 10-83(231) |
| f-e-da | 10-82(229), 10-84(231) |
| f-e-val | 10-82(229), 10-84(231) |
| FILE | 5-38(64), 5-46(85), 9-6(9) |
| file-bif | 11-50(168), 11-51(169), 11-51(170) |
| FILEMARK | 2-40(130), 2-40(131), 2-40(132), 2-42(140), 2-43(141), 2-43(142), 2-44(145), 10-30(85), 10-79(223), 10-80(225), 10-80(226), 10-86(243) |
| fill-char | 8-10(36), 8-10(137) |
| FINISH | 2-29(99), 4-9(14), 4-9(16), 5-53(97), 5-53(97), 9-4(1), 9-11(18), 9-12(19), 12-13(103) |
| PIX | 8-7(20), 8-9(32), 8-12(50), 8-12(51), 8-16(71), 8-16(71), 8-17(73), 8-19(80), 8-19(80), 8-24(97), 8-24(97), 8-24(97), 8-25(101), 8-25(101), 8-27(106), 8-27(107), 8-27(107), 8-28(112), 8-32(125), 8-32(125), 8-36(141), 8-37(146), 8-39(154), 8-39(154), 8-64(239), 10-69(195), 10-73(206), 10-81(227), 10-81(227), 10-82(229), 10-83(231), 10-83(231), 10-83(231), 10-83(231), 10-93(259), 10-102(280), 10-103(281), 10-105(285), 10-105(285), 11-30(41), 11-31(42), 11-32(46), 11-32(46), 11-32(47), 11-32(47), 11-34(56), 11-36(66), 11-36(66), 11-37(711), 11-52(173), 11-52(174), 12-7(38), 12-9(62) |
| floor | 1-11(151) |</p>
<table>
<thead>
<tr>
<th>PLT</th>
<th>5-39 (65), 8-7 (20), 8-8 (28), 8-9 (32), 8-17 (73), 8-17 (73), 8-22 (89), 8-22 (90), 8-24 (97), 8-25 (101), 8-32 (125), 8-36 (141), 8-37 (146), 8-39 (152), 8-64 (239), 10-69 (195), 10-73 (206), 10-81 (227), 10-81 (227), 10-82 (229), 10-82 (229), 10-83 (231), 10-83 (231), 10-84 (237), 10-93 (259), 10-102 (280), 10-103 (281), 10-105 (285), 10-105 (285), 11-20 (39), 11-30 (41), 11-31 (42), 11-33 (49), 11-35 (63), 11-35 (63), 11-36 (67), 11-37 (71), 12-7 (38), 12-9 (62)</th>
</tr>
</thead>
<tbody>
<tr>
<td>PLT-DA</td>
<td>11-19 (37), 11-33 (92)</td>
</tr>
<tr>
<td>PQPL</td>
<td>2-29 (99), 2-31 (103), 8-17 (73), 9-4 (1), 9-11 (18), 9-12 (19), 12-13 (103)</td>
</tr>
<tr>
<td>FREE</td>
<td>2-33 (108), 12-13 (98)</td>
</tr>
<tr>
<td>free</td>
<td>4-11 (28), 6-18 (41), 6-19 (42)</td>
</tr>
<tr>
<td>free-ag</td>
<td>4-11 (26), 4-11 (26), 4-11 (26), 4-11 (26), 4-11 (26)</td>
</tr>
<tr>
<td>free-based-set</td>
<td>5-14 (20), 5-19 (21)</td>
</tr>
<tr>
<td>free-block</td>
<td>3-5 (7), 5-33 (53), 5-33 (53), 5-33 (53)</td>
</tr>
<tr>
<td>free-set</td>
<td>4-9 (18), 4-11 (26)</td>
</tr>
<tr>
<td>free-task</td>
<td>10-25 (65), 10-39 (104)</td>
</tr>
<tr>
<td>from-direct</td>
<td>10-25 (65), 10-39 (106)</td>
</tr>
<tr>
<td>from-segu</td>
<td>10-18 (44), 10-19 (55)</td>
</tr>
<tr>
<td>fu-elem</td>
<td>10-19 (45), 10-20 (48)</td>
</tr>
<tr>
<td>fu-lsz</td>
<td>10-19 (45), 10-19 (45), 10-19 (45)</td>
</tr>
<tr>
<td>fu-pos</td>
<td>10-19 (45), 10-19 (45), 10-19 (45)</td>
</tr>
<tr>
<td>fu-psz</td>
<td>10-19 (45), 10-20 (48)</td>
</tr>
<tr>
<td>G-CHAR</td>
<td>8-4 (11)</td>
</tr>
<tr>
<td>GE</td>
<td>5-24 (40), 5-24 (41), 8-12 (48), 8-18 (78), 12-16 (136)</td>
</tr>
<tr>
<td>GEN</td>
<td>5-42 (73), 5-46 (85), 11-5 (7), 11-18 (36)</td>
</tr>
<tr>
<td>gen-base</td>
<td>10-20 (50), 10-21 (51), 10-21 (51), 10-44 (115), 10-44 (115), 10-46 (120), 10-68 (192), 10-68 (192), 10-89 (247), 10-89 (247), 10-107 (290), 10-107 (290)</td>
</tr>
<tr>
<td>gen-da-match</td>
<td>5-40 (66), 5-40 (67), 5-40 (67), 5-40 (67)</td>
</tr>
<tr>
<td>gen-da-match-1</td>
<td>5-42 (74), 5-42 (75), 5-42 (75), 5-42 (75), 5-51 (95)</td>
</tr>
<tr>
<td>gen-length</td>
<td>10-96 (261), 10-96 (261), 10-96 (261)</td>
</tr>
<tr>
<td>gen-list</td>
<td>4-16 (34), 5-16 (34), 6-16 (34), 6-16 (34), 6-17 (39)</td>
</tr>
<tr>
<td>gen-match</td>
<td>5-36 (59), 5-40 (66)</td>
</tr>
<tr>
<td>gen-match-1</td>
<td>5-42 (73), 5-42 (74), 5-48 (88)</td>
</tr>
</tbody>
</table>
IB
1'1
UB
VIE NNA AI -21
TR
25. 082

gen-op

2-11(20), 3-3(3), 4-5(7), 4-5(8), 4-12(30), 4-12(39),
4-20(49), 5-24(40), 5-25(44), 6-18(41), 6-18(41), 7-14(31), 7-14(32),
7-14(33), 7-14(34), 7-15(35), 7-15(36), 7-16(37), 7-16(37),
7-16(38), 7-19(42), 7-20(50), 10-92(256), 11-47(150), 11-53(179)
generic-match

5-54(101), 5-55(101)
generic-sel

5-32(51), 5-54(99), 7-20(49), 7-24(58)
gen generic-sel-1

5-38(64), 5-54(99), 5-54(100)
gen generic-sel-2

5-54(100), 5-54(101)
get

2-33(108), 10-52(132), 10-52(133), 12-14(113), 12-14(114)
get-char-list

10-69(193), 10-92(228), 10-82(228)
get-column

10-69(195), 10-80(225), 10-80(225), 10-80(225)
get-contra-format

10-69(194), 10-69(195)
get-copy

10-65(182), 10-65(182), 10-65(182), 10-66(184), 10-69(193), 10-69(194),
10-70(197), 10-77(220), 10-89(243)
get-data

10-74(207), 10-77(219)
get-data-field

10-69(190), 10-69(193)
get-data-format

10-69(193), 10-69(194), 10-69(195), 10-69(195)
get-data-lp

10-66(184), 10-74(207), 10-74(207), 10-74(207), 10-74(207), 10-77(220)
get-data-rp

10-74(207), 10-75(208), 10-75(208), 10-75(208)
get-data-spec

10-64(179), 10-64(179), 10-66(184), 10-66(184)
get-endfile

10-64(179), 10-64(180)
get-file-space

10-78(221), 10-79(223), 10-79(223), 10-79(223)
get-fa-lg

10-69(193), 10-81(A), 10-81(A), 10-91(228), 10-102(280)
get-list-edit

10-52(132), 10-66(184), 10-66(184), 10-67(189), 10-67(189), 10-67(189)
get-listdir

10-69(193), 10-71(199), 10-71(199), 10-71(199)
get-next-char

10-20(196), 10-71(199), 10-74(207), 10-75(208), 10-82(228)
get-scalor

10-67(189), 10-68(190), 10-68(191)
get-skip

10-65(182), 10-69(195), 10-80(225), 10-80(225), 10-80(226), 10-80(226)
get-space

10-69(195), 10-70(196), 10-78(221)
get-string-space

10-78(221), 10-78(222), 10-78(222), 10-78(222)
goto

2-33(108), 12-11(82)
goto-jump

5-28(47), 5-29(50), 5-29(50), 5-29(50), 5-29(50), 5-52(96)

goto-search

5-26(46), 5-28(47), 5-28(47), 5-28(47), 5-28(47), 5-28(47)

GT

8-4(9), 8-12(48), 8-18(78), 8-18(78), 8-19(81), 12-16(136)

H-CHAR

8-4(11)

head

1-6(8)

I

2-21(54), 2-21(56), 2-21(56), 6-19(42)

I-CHAR

8-4(11), 8-32(123), 8-45(174), 8-45(176), 8-55(210), 10-55(142), 10-55(143), 10-72(204), 12-8(51)

ID-CHAR-DA


id-ref

5-17(25), 5-17(25), 5-33(52)

idl-string

10-98(268), 10-98(268), 10-98(269)

IP

2-33(108), 12-11(81)

IGN

10-33(92), 10-47(123)

ignore

10-33(92), 10-48(124), 10-48(124)

imag

imag

IMAG

7-12(27), 7-12(29)

IN


index

11-43(125), 11-43(126), 11-43(126)

infix-bit

8-15(67), 8-21(86)

infix-num

8-16(68), 8-16(70), 8-16(71)

infix-op

8-14(65), 8-14(66), 11-37(78), 11-44(131), 11-46(142), 11-46(144)

infix-operator

7-18(43), 7-18(44)

infix-val

8-14(66), 8-14(66)

INH

2-34(110), 4-6(10), 10-8(A), 10-22(54)

init

6-15(32), 6-15(32), 6-16(35), 6-16(35)

init-expl-close

10-21(52), 10-22(53)

init-expl-open

10-11(20), 10-11(20), 10-11(20)

init-file-1

10-64(179), 10-65(181)

init-file-2

10-64(179), 10-65(182)
int-delete-1 ............................................. 10-48(125), 10-48(126)
int-display-st ........................................... 5-18(28), 10-106(128)
int-do-list ............................................... 5-24(39), 5-25(43)
int-exit-st ................................................ 9-2(161), 5-18(28)
int-expl-open ............................................ 10-11(21), 10-12(25)
int-free-list ............................................. 6-18(40), 6-18(41), 5-18(41), 6-18(41)
int-free-st ................................................ 5-18(28), 5-19(40)
int-get-st .................................................. 5-18(28), 10-64(139)
int-goto-st ................................................ 5-26(45), 5-26(64)
int-goto-st-1 ............................................. 5-18(45), 5-21(33)
int-group ..................................................... 5-18(28), 5-20(29)
int-if-st .................................................... 5-20(29), 5-20(30)
int-ignore ................................................... 10-37(100), 10-47(122)
int-ignore-1 .............................................. 10-47(122), 10-47(123)
int-impl-open ............................................. 10-13(121), 10-34(93), 10-38(101), 10-40(107), 10-42(111), 10-43(114), 10-46(119), 10-47(122), 10-48(125), 10-64(179), 10-88(245)
int-into-read ............................................. 10-37(100), 10-43(114)
int-into-1 .................................................. 10-43(114), 10-44(115)
int-item-list .............................................. 5-25(43), 10-52(132)
int-locate-st .............................................. 5-18(28), 10-42(133)
int-next-st ................................................ 5-16(23), 5-16(24), 5-16(24), 5-16(24), 5-22(35), 5-29(50), 5-29(50), 5-29(50), 10-31(87), 10-74(207), 10-77(220)
int-on-st ................................................... 5-18(28), 10-11(20)
int-open-st ................................................ 5-18(28), 10-11(20)
int-proc-body ............................................. 3-5(6), 4-6(10), 5-50(89), 5-50(89)
int-proc-st-list .......................................... 5-50(90), 5-52(90)
int-prog .................................................... 3-2(2), 3-5(1)
int-program-call ......................................... 3-5(51), 3-5(61)
<table>
<thead>
<tr>
<th>Function</th>
<th>Pages</th>
<th>References</th>
</tr>
</thead>
<tbody>
<tr>
<td>int-proper</td>
<td>5-17</td>
<td>5-18 (28)</td>
</tr>
<tr>
<td>int-put</td>
<td>5-18</td>
<td>10-88 (28)</td>
</tr>
<tr>
<td>int-read</td>
<td>5-18</td>
<td>10-37 (109)</td>
</tr>
<tr>
<td>int-reply</td>
<td>10-107 (290), 10-107 (291), 10-107 (292)</td>
<td>10-107 (292)</td>
</tr>
<tr>
<td>int-reply-event</td>
<td>10-106 (288), 10-107 (290)</td>
<td>10-107 (290)</td>
</tr>
<tr>
<td>int-return</td>
<td>5-18 (28), 5-50 (30), 5-52 (271)</td>
<td></td>
</tr>
<tr>
<td>int-revert</td>
<td>5-18 (28), 9-71 (271)</td>
<td></td>
</tr>
<tr>
<td>int-rewrite</td>
<td>10-40 (107), 10-40 (109)</td>
<td></td>
</tr>
<tr>
<td>int-set-read</td>
<td>10-37 (100), 10-46 (109)</td>
<td></td>
</tr>
<tr>
<td>int-set-1</td>
<td>10-46 (119), 10-46 (120)</td>
<td></td>
</tr>
<tr>
<td>int-signal</td>
<td>5-18 (28), 9-14 (26)</td>
<td></td>
</tr>
<tr>
<td>int-snap</td>
<td>9-12 (21), 9-12 (21), 9-13 (21)</td>
<td></td>
</tr>
<tr>
<td>int-st</td>
<td>5-16 (24), 5-17 (25), 5-20 (30), 5-20 (30), 5-29 (50), 5-29 (50), 9-13 (24)</td>
<td></td>
</tr>
<tr>
<td>int-st-list</td>
<td>5-3 (1), 5-16 (24), 5-18 (28), 5-21 (34), 5-25 (43)</td>
<td></td>
</tr>
<tr>
<td>int-stand-check</td>
<td>9-4 (1), 9-5 (16)</td>
<td></td>
</tr>
<tr>
<td>int-stand-print-open</td>
<td>10-16 (32), 10-86 (243), 10-105 (286)</td>
<td></td>
</tr>
<tr>
<td>int-stop-st</td>
<td>4-9 (14), 5-18 (28)</td>
<td></td>
</tr>
<tr>
<td>int-task-call</td>
<td>4-11 (1), 5-50 (89)</td>
<td></td>
</tr>
<tr>
<td>int-unif</td>
<td>10-11 (20), 10-11 (21), 10-11 (23)</td>
<td></td>
</tr>
<tr>
<td>int-unlock</td>
<td>5-18 (28), 10-34 (23)</td>
<td></td>
</tr>
<tr>
<td>int-unlock-1</td>
<td>4-20 (48), 10-34 (93), 10-34 (24), 10-35 (99)</td>
<td></td>
</tr>
<tr>
<td>int-wait</td>
<td>4-16 (39), 4-17 (42), 4-17 (42)</td>
<td></td>
</tr>
<tr>
<td>int-wait-st</td>
<td>4-15 (31), 5-18 (28)</td>
<td></td>
</tr>
<tr>
<td>int-wait-st-1</td>
<td>4-15 (31), 4-15 (31)</td>
<td></td>
</tr>
<tr>
<td>int-wait-1</td>
<td>4-15 (33), 4-16 (37), 4-17 (42)</td>
<td></td>
</tr>
<tr>
<td>int-wait-2</td>
<td>4-16 (37), 4-16 (39), 4-16 (39), 4-16 (39)</td>
<td></td>
</tr>
<tr>
<td>int-while-group</td>
<td>5-21 (33), 5-21 (34), 5-21 (34)</td>
<td></td>
</tr>
<tr>
<td>int-write</td>
<td>5-18 (28), 10-38 (101)</td>
<td></td>
</tr>
<tr>
<td>int-write-1</td>
<td>10-38 (101), 10-38 (102)</td>
<td></td>
</tr>
<tr>
<td>Command</td>
<td>Page Numbers</td>
<td></td>
</tr>
<tr>
<td>------------------</td>
<td>-----------------------------------</td>
<td></td>
</tr>
<tr>
<td>INTG</td>
<td>11-5(7), 11-6(14), 11-18(36), 11-19(38)</td>
<td></td>
</tr>
<tr>
<td>intg-const-val</td>
<td>5-40(68), 5-40(68), 5-41(70)</td>
<td></td>
</tr>
<tr>
<td>intg-da</td>
<td>8-12(50), 8-12(50), 8-13(54), 8-38(149)</td>
<td></td>
</tr>
<tr>
<td>INTG-DA</td>
<td>8-11(42), 8-12(52), 8-13(54), 8-38(149)</td>
<td></td>
</tr>
<tr>
<td>intg-op</td>
<td>5-24(40), 5-25(42)</td>
<td></td>
</tr>
<tr>
<td>intg-star-extents</td>
<td>5-47(86), 5-47(87), 5-51(94)</td>
<td></td>
</tr>
<tr>
<td>intql-string</td>
<td>10-98(268), 10-98(268), 10-99(270)</td>
<td></td>
</tr>
<tr>
<td>into-direct</td>
<td>10-33(92), 10-94(116)</td>
<td></td>
</tr>
<tr>
<td>into-unbuf-unkeyd</td>
<td>10-33(92), 10-45(117)</td>
<td></td>
</tr>
<tr>
<td>invalid-open</td>
<td>10-17(39), 10-17(40)</td>
<td></td>
</tr>
<tr>
<td>io-cond-iterate</td>
<td>4-19(45), 4-19(46)</td>
<td></td>
</tr>
<tr>
<td>io-cond-iterate-1</td>
<td>4-19(46), 4-20(47), 4-20(47)</td>
<td></td>
</tr>
<tr>
<td>io-cond-return</td>
<td>10-30(86), 10-31(87)</td>
<td></td>
</tr>
<tr>
<td>io-transmission</td>
<td>10-32(88), 10-33(90), 10-33(92)</td>
<td></td>
</tr>
<tr>
<td>MREEL</td>
<td>11-8(17), 11-18(35)</td>
<td></td>
</tr>
<tr>
<td>is-active</td>
<td>4-6(10), 4-6(10), 4-6(10), 4-6(10)</td>
<td></td>
</tr>
<tr>
<td>is-active-pa</td>
<td>2-10(15), 2-10(16), 2-11(18), 2-11(19)</td>
<td></td>
</tr>
<tr>
<td>is-aq</td>
<td>2-1(1), 2-4(4), 2-27(94)</td>
<td></td>
</tr>
<tr>
<td>is-aq-entry</td>
<td>2-27(94), 2-27(94)</td>
<td></td>
</tr>
<tr>
<td>is-al</td>
<td>12-12(92), 12-12(93), 12-12(94), 12-12(95), 12-12(96)</td>
<td></td>
</tr>
<tr>
<td>is-allocate</td>
<td>12-12(91), 12-12(92)</td>
<td></td>
</tr>
<tr>
<td>is-allocate-st</td>
<td>5-18(28), 12-10(74), 12-11(92)</td>
<td></td>
</tr>
<tr>
<td>is-allocated</td>
<td>3-1(3), 3-3(3), 3-3(3), 3-3(3)</td>
<td></td>
</tr>
<tr>
<td>is-alpham-char</td>
<td>8-4(9), 8-4(10), 10-57(162)</td>
<td></td>
</tr>
<tr>
<td>is-applic</td>
<td>2-21(51), 2-21(58), 2-22(61), 2-22(62), 2-22(65), 2-22(66), 2-23(72), 6-11(19), 6-13(25), 11-49(162)</td>
<td></td>
</tr>
<tr>
<td>is-applic-1</td>
<td>2-22(61), 2-24(78), 2-24(80), 11-49(158)</td>
<td></td>
</tr>
<tr>
<td>is-area</td>
<td>5-38(62), 5-38(62), 5-40(67), 5-40(67), 5-42(75), 5-42(75), 5-43(78), 5-43(78), 5-43(78), 5-43(78), 5-43(78), 6-6(10), 6-6(10), 6-7(11), 6-9(15), 6-16(33), 6-16(33), 6-19(44), 7-9(19), 7-9(19), 7-12(29), 8-30(119), 8-30(119), 8-36(140), 11-49(158), 11-49(159), 12-19(168)</td>
<td></td>
</tr>
<tr>
<td>is-area-appl</td>
<td>6-9 (15), 6-10 (17), 7-10 (22)</td>
<td></td>
</tr>
<tr>
<td>is-area-da</td>
<td>12-6 (22), 12-7 (37), 12-7 (40), 12-12 (93), 12-19 (168)</td>
<td></td>
</tr>
<tr>
<td>is-area-eda</td>
<td>2-19 (47), 12-18 (157), 12-18 (159), 12-19 (168)</td>
<td></td>
</tr>
<tr>
<td>is-area-type</td>
<td>2-23 (74), 2-26 (93)</td>
<td></td>
</tr>
<tr>
<td>is-arg-expr</td>
<td>12-17 (143), 12-17 (144), 12-17 (145)</td>
<td></td>
</tr>
<tr>
<td>is-arithm</td>
<td>2-19 (47), 5-23 (37), 8-3 (1), 8-3 (2), 8-5 (15), 8-7 (19), 8-7 (20), 8-7 (20), 8-8 (26), 8-8 (26), 8-9 (29), 8-9 (29), 8-9 (33), 8-9 (33), 8-18 (78), 8-23 (92), 8-31 (122), 8-31 (122), 8-38 (149), 8-38 (150), 8-38 (151), 8-39 (156),</td>
<td></td>
</tr>
<tr>
<td>is-arithm-const</td>
<td>10-97 (265), 11-13 (29), 11-14 (33), 11-31 (44), 11-32 (45), 11-51 (172), 11-51 (172), 11-52 (173), 11-52 (174), 12-6 (22), 12-7 (37), 12-7 (39), 12-17 (149), 12-19 (157)</td>
<td></td>
</tr>
<tr>
<td>is-arithm-opor</td>
<td>8-12 (47), 8-15 (67), 8-22 (91), 8-25 (101)</td>
<td></td>
</tr>
<tr>
<td>is-array</td>
<td>5-5 (6), 5-13 (18), 5-38 (62), 5-40 (67), 5-40 (67), 5-42 (75), 5-42 (75), 5-43 (78), 5-43 (78), 5-43 (78), 5-44 (79), 5-44 (79), 5-55 (103), 6-5 (7), 6-6 (10), 6-6 (10), 6-7 (11), 6-7 (11), 6-17 (39), 7-4 (3), 7-6 (10), 7-6 (12), 7-7 (13), 7-7 (13), 7-7 (13), 7-22 (55), 7-23 (56), 7-23 (56), 7-23 (56), 7-23 (56), 7-23 (56), 7-28 (71), 7-30 (78), 7-31 (81), 7-32 (55), 7-32 (55), 7-33 (86), 7-33 (86), 7-33 (86), 7-33 (86), 7-33 (86),</td>
<td></td>
</tr>
<tr>
<td>is-array-al</td>
<td>6-6 (10), 6-6 (10), 12-12 (93), 12-12 (94), 12-8 (45), 12-19 (163)</td>
<td></td>
</tr>
<tr>
<td>is-array-descr</td>
<td>12-18 (152), 12-18 (153), 12-19 (163)</td>
<td></td>
</tr>
<tr>
<td>is-array-eda</td>
<td>12-18 (152), 12-18 (153), 12-19 (163)</td>
<td></td>
</tr>
<tr>
<td>is-assign-st</td>
<td>5-19 (28), 12-10 (74), 12-12 (90)</td>
<td></td>
</tr>
<tr>
<td>is-at</td>
<td>2-1 (1), 2-1 (1), 2-1 (1), 2-1 (1), 2-1 (1), 2-1 (1)</td>
<td></td>
</tr>
<tr>
<td>is-attr</td>
<td>2-18 (38), 2-18 (38)</td>
<td></td>
</tr>
<tr>
<td>is-based</td>
<td>2-18 (39), 5-9 (12), 6-3 (2), 6-3 (2), 6-18 (41), 6-18 (41), 7-2 (A), 7-24 (58), 7-24 (61), 7-24 (61), 7-25 (60), 10-42 (111), 12-5 (9), 12-6 (27), 12-19 (161)</td>
<td></td>
</tr>
<tr>
<td>is-basic-ref</td>
<td>10-66 (185), 10-77 (219), 12-5 (14), 12-6 (26), 12-17 (142), 12-17 (143), 12-17 (144)</td>
<td></td>
</tr>
<tr>
<td>is-bin-pic</td>
<td>10-9 (45), 10-9 (45), 10-9 (45), 10-9 (45), 10-9 (45), 10-9 (45)</td>
<td></td>
</tr>
<tr>
<td>is-bin-spec</td>
<td>8-45 (172), 12-9 (54), 12-9 (54), 12-9 (54), 12-9 (54)</td>
<td></td>
</tr>
<tr>
<td>is-bintg</td>
<td>8-11 (45), 8-11 (45), 8-11 (45), 8-11 (45), 8-11 (45), 8-11 (45)</td>
<td></td>
</tr>
<tr>
<td>Term</td>
<td>References</td>
<td></td>
</tr>
<tr>
<td>---------------------------</td>
<td>-------------------------------------------------</td>
<td></td>
</tr>
<tr>
<td>is-correct-edn</td>
<td>3-9(12), 3-9(12), 5-11(13), 5-44(80), 6-4(4), 6-5(7), 6-8(13), 6-9(15)</td>
<td></td>
</tr>
<tr>
<td>is-correct-key</td>
<td>10-26(68), 10-26(191)</td>
<td></td>
</tr>
<tr>
<td>is-correct-static</td>
<td>3-7(10), 2-8(111)</td>
<td></td>
</tr>
<tr>
<td>is-corresp</td>
<td>7-28(70), 7-29(73), 7-29(74), 7-31(81), 7-31(81), 7-31(81)</td>
<td></td>
</tr>
<tr>
<td>is-cplx-format</td>
<td>12-9(61), 12-2(163)</td>
<td></td>
</tr>
<tr>
<td>is-cplx-vr</td>
<td>8-6(17), 8-6(181)</td>
<td></td>
</tr>
<tr>
<td>is-csa</td>
<td>2-1(1), 2-4(4), 2-15(30), 2-31(102)</td>
<td></td>
</tr>
<tr>
<td>is-csa</td>
<td>2-37(118), 2-40(127), 10-24(2), 10-8(A)</td>
<td></td>
</tr>
<tr>
<td>is-d</td>
<td>2-1(1), 2-4(4), 2-15(101), 2-15(30)</td>
<td></td>
</tr>
<tr>
<td>is-d-char</td>
<td>8-46(181), 8-46(182)</td>
<td></td>
</tr>
<tr>
<td>is-da</td>
<td>8-11(38), 12-19(162)</td>
<td></td>
</tr>
<tr>
<td>is-da-descr</td>
<td>12-6(19), 12-7(44), 12-8(45), 12-8(47), 12-19(162)</td>
<td></td>
</tr>
<tr>
<td>is-data</td>
<td>2-40(132), 2-41(134)</td>
<td></td>
</tr>
<tr>
<td>is-data-directed</td>
<td>10-66(184), 10-89(249), 12-15(118), 12-15(119)</td>
<td></td>
</tr>
<tr>
<td>is-data-format</td>
<td>10-53(138), 12-9(60), 12-9(161)</td>
<td></td>
</tr>
<tr>
<td>is-data-spec</td>
<td>12-14(113), 12-14(114), 12-14(116), 12-14(117), 12-15(119)</td>
<td></td>
</tr>
<tr>
<td>is-data-tr</td>
<td>10-52(134), 10-52(135), 10-91(253), 10-92(256)</td>
<td></td>
</tr>
<tr>
<td>is-datadir-ref</td>
<td>10-59(169), 10-60(171), 10-75(211)</td>
<td></td>
</tr>
<tr>
<td>is-datadir-ref-corr</td>
<td>10-60(171), 10-75(210)</td>
<td></td>
</tr>
<tr>
<td>is-datadir-rp</td>
<td>10-59(169), 10-60(171), 10-60(172), 10-75(209)</td>
<td></td>
</tr>
<tr>
<td>is-datadir-rp-corr</td>
<td>10-60(171), 10-75(208)</td>
<td></td>
</tr>
<tr>
<td>is-datadir-rp-leftcorr</td>
<td>10-60(172), 10-75(208)</td>
<td></td>
</tr>
<tr>
<td>is-dataset</td>
<td>2-39(121), 2-39(122)</td>
<td></td>
</tr>
<tr>
<td>is-dec-diq-spec</td>
<td>8-45(173), 8-45(174), 8-49(193), 8-49(193), 8-49(194), 8-49(194)</td>
<td></td>
</tr>
<tr>
<td>is-dec-pic</td>
<td>8-42(158), 8-44(166), 8-44(168), 8-47(187), 10-81(227), 10-97(265), 12-8(49), 12-8(50)</td>
<td></td>
</tr>
<tr>
<td>is-dec-spec</td>
<td>12-8(50), 12-8(50), 12-8(51), 12-8(53)</td>
<td></td>
</tr>
<tr>
<td>is-decl</td>
<td>3-5(5), 3-10(17), 12-5(8), 12-5(9)</td>
<td></td>
</tr>
<tr>
<td>is-decl-part</td>
<td>3-10(17), 12-4(4), 12-5(8), 12-11(76)</td>
<td></td>
</tr>
<tr>
<td>is-defined</td>
<td>2-18(39), 5-9(12), 7-2(4), 7-24(58), 7-24(61), 7-24(61), 7-25(50), 7-28(68), 12-5(9), 12-6(26), 12-19(161)</td>
<td></td>
</tr>
</tbody>
</table>
is-delay-st: 5-18(28), 12-10(74), 12-12(107)

is-delete-st: 5-18(28), 10-13(33), 10-15(37), 10-17(40), 12-10(74), 12-16(131)

is-den: 2-19(42), 2-19(43)

is-dens: 2-19(46), 2-28(97), 12-5(10), 12-5(11), 12-6(19), 12-6(26), 12-6(27)

is-descr: 2-19(41), 2-19(48), 12-5(17), 12-6(19), 12-6(20), 12-6(30)

is-digit: 8-45(173), 8-46(183), 8-46(183), 8-47(185)

is-display-st: 5-18(28), 12-10(74), 12-15(133)

is-dn: 2-1(1), 2-19(42), 2-31(103), 2-32(104)

is-do-spec: 12-11(79), 12-11(80), 12-15(123)

is-dr-ed-spec: 8-45(179), 8-46(180), 8-48(192), 8-49(193), 9-49(193), 8-50(195), 8-50(195), 8-50(195), 8-57(217)

is-dr-spec: 8-42(158), 8-45(179)

is-ds-descr: 2-39(123), 2-39(123), 10-17(39)

is-ds-label: 10-12(26), 10-12(27)

is-dummy-descr: 5-36(59), 5-43(76)

is-dyn-pref-part: 2-19(44), 2-20(50), 2-31(102), 2-31(103), 2-31(103), 2-32(105)

is-e: 2-1(1), 2-4(4), 2-6(8), 2-15(30), 2-18(137), 2-18(38), 2-19(44), 2-20(50), 2-31(103), 2-32(104), 2-32(105)

is-e-const: 10-44(235), 10-84(236)

is-e-data: 10-59(169), 10-60(171), 10-60(172), 10-84(235)

is-e-data-corr: 10-59(169), 10-59(169), 10-60(171), 10-60(171), 10-83(231)

is-e-data-leftcorr: 10-60(171), 10-60(172), 10-60(172), 10-83(231)

is-e-exponent: 10-61(175), 10-72(202), 10-72(203)

is-e-ret-type: 2-19(44), 2-19(47), 3-3(3), 11-44(133)

is-ed-spec: 2-46(191), 8-50(195), 9-50(195), 8-50(195), 8-57(217)

is-eda: 2-19(43), 2-19(46), 2-27(96), 2-28(97), 12-19(152), 12-19(152), 12-18(153), 12-18(155), 12-19(162)

is-edescr: 2-19(44), 2-19(45)

is-edit-directed: 10-56(184), 10-66(194), 10-89(249), 10-89(249), 12-15(118), 12-15(120)

is-edit-tr: 10-52(134), 10-53(137), 10-68(192), 10-69(193)
is-f-data ........................................ 10-59(169), 10-60(171), 10-60(172), 10-84(232)
is-f-data-corr ................................... 10-60(171), 10-83(231)
is-f-data-leftcorr ................................ 10-60(172), 10-83(231)
is-f-io-cond ....................................... 2-6(7), 2-30(100), 2-32(104), 2-32(106), 9-4(1), 9-4(11), 9-12(19)
is-fd ............................................... 2-1(1), 2-4(4), 2-34(109)
is-f-elem .......................................... 2-34(109), 2-34(110), 10-8(A)
is-file ............................................... 2-18(39), 3-7(10), 3-10(17), 5-17(60), 5-17(60), 5-38(64), 5-46(85),
                                             5-55(102), 5-55(105), 10-8(A), 10-11(22), 12-5(9), 12-9(23)
is-file-attr ....................................... 10-8(A), 12-6(21), 12-6(23), 12-6(24), 12-14(109)
is-file-descr ....................................... 2-19(45), 2-19(45), 2-19(45), 5-37(60), 5-55(105), 12-6(18), 12-6(21)
is-file-get ......................................... 10-13(33), 10-14(35), 10-17(40), 10-64(179), 12-14(112), 12-14(113)
is-file-put ......................................... 10-13(33), 10-14(35), 10-17(40), 10-88(245), 12-14(115), 12-14(116)
is-fix-match ....................................... 8-62(232), 8-62(233), 8-63(238)
is-fix-pic ........................................... 8-43(163), 8-44(170), 8-47(188), 8-52(199), 8-62(232), 8-63(238),
                                             8-64(241)
is-fixed-string-edata .............................. 12-17(150), 12-17(151)
is-float-generic-builtin .......................... 5-38(64), 11-12(23)
is-flt-match ....................................... 8-62(232), 8-62(233), 8-63(238)
is-flt-pic .......................................... 8-43(163), 8-44(171), 8-47(188), 8-48(192), 8-52(199), 8-62(232),
                                             8-63(238), 8-64(239), 8-64(241)
is-free ............................................... 8-17(73), 8-17(73)
is-format ............................................ 2-20(50), 2-35(115), 2-36(116), 2-37(118), 2-38(119), 10-53(137), 12-6(28),
                                             12-9(28), 12-9(28), 12-9(28), 12-9(28), 12-9(28), 12-15(120)
is-format-attr ..................................... 2-18(39), 5-9(12), 5-12(17), 7-7(13), 7-19(46), 7-24(58), 12-5(9),
                                             12-5(9)
is-format-den ...................................... 2-19(43), 2-20(150), 8-4(6), 8-5(15), 10-54(139), 10-54(139)
is-format-item ..................................... 12-9(58), 12-9(60)
is-format-iter ..................................... 10-53(138), 12-9(58), 12-9(59)
is-free ............................................. 12-13(98), 12-13(99)
is-free-st .......................................... 5-18(28), 12-10(74), 12-13(98)
is-fu ................................................ 2-1(1), 2-35(112)
is-fu-elem ........................................ 2-35(112), 2-35(113), 10-5(A), 10-9(A), 10-21(A), 10-23(A), 10-53(138),
                                             10-63(A), 10-87(A)
<table>
<thead>
<tr>
<th>is-inner-dataset</th>
<th>2-40(127), 2-41(137)</th>
</tr>
</thead>
<tbody>
<tr>
<td>is-intg</td>
<td>8-11(64), 8-13(541), 8-16(69), 8-22(90), 8-26(104)</td>
</tr>
<tr>
<td>is-intg-const</td>
<td>3-8(11), 3-8(11), 5-9(12), 5-40(68), 5-40(68), 5-41(69), 5-47(87)</td>
</tr>
<tr>
<td>is-intg-exponent</td>
<td>7-18(44), 7-22(54), 9-26(104)</td>
</tr>
<tr>
<td>is-intg-op</td>
<td>5-41(69), 5-41(69), 5-41(69), 5-47(87), 11-6(14)</td>
</tr>
<tr>
<td>is-intg-val</td>
<td>2-6(7), 2-7(12), 2-7(12), 2-7(12), 2-12(27), 2-13(29), 2-16(32), 2-17(34)</td>
</tr>
<tr>
<td>is-into-read</td>
<td>10-15(37), 10-37(100), 12-15(124), 12-15(125)</td>
</tr>
<tr>
<td>is-io-arithm-const</td>
<td>10-55(141), 10-55(142), 10-58(166), 10-62(176), 10-62(176), 10-62(178), 10-72(204), 10-84(234), 10-84(236)</td>
</tr>
<tr>
<td>is-io-arithm-data</td>
<td>10-55(140), 10-55(141), 10-61(175)</td>
</tr>
<tr>
<td>is-io-basic-ref</td>
<td>10-52(161), 10-60(170)</td>
</tr>
<tr>
<td>is-io-basic-ref-blanks</td>
<td>10-50(170), 10-75(211)</td>
</tr>
<tr>
<td>is-io-binary-fixed</td>
<td>10-56(149), 10-57(155), 10-62(178), 10-73(206)</td>
</tr>
<tr>
<td>is-io-binary-fixed-part</td>
<td>10-57(155), 10-57(156), 10-62(178), 10-62(178)</td>
</tr>
<tr>
<td>is-io-binary-float</td>
<td>10-56(149), 10-57(158), 10-62(178), 10-72(203), 10-73(206)</td>
</tr>
<tr>
<td>is-io-binary-integer</td>
<td>10-57(156), 10-57(156), 10-57(156), 10-57(157), 10-62(178)</td>
</tr>
<tr>
<td>is-io-hit-char</td>
<td>10-56(146), 10-56(148), 10-57(157), 10-85(240)</td>
</tr>
<tr>
<td>is-io-bit-string</td>
<td>10-55(140), 10-56(145), 10-72(204)</td>
</tr>
<tr>
<td>is-io-char-string</td>
<td>10-55(140), 10-56(145), 10-72(204)</td>
</tr>
<tr>
<td>is-io-cond</td>
<td>2-32(106), 9-8(15), 9-8(15), 12-13(105), 12-13(106)</td>
</tr>
<tr>
<td>is-io-cond-arg</td>
<td>2-5(6), 2-5(7)</td>
</tr>
<tr>
<td>is-io-cplx-expr</td>
<td>10-55(141), 10-55(143), 10-62(176), 10-72(204)</td>
</tr>
<tr>
<td>is-io-data</td>
<td>10-55(140), 10-72(202)</td>
</tr>
</tbody>
</table>
is-io-decimal-fixed 10-56(149), 10-56(150), 10-56(151), 10-58(166), 10-62(178), 10-73(206), 10-73(206), 10-73(206), 10-84(234), 10-84(236)

is-io-decimal-float 10-56(149), 10-56(151), 10-62(178), 10-72(203), 10-73(206), 10-84(236)

is-io-del-subs 10-58(164), 10-58(165), 10-76(213)

is-io-exponent 10-56(151), 10-56(152), 10-57(158), 10-62(178), 10-62(178)

is-io-identifier 10-57(161), 10-57(162), 10-60(170)

is-io-identifier-blanks 10-57(163), 10-60(170)

is-io-integer 10-56(147), 10-56(147), 10-56(150), 10-56(150), 10-56(150), 10-56(153), 10-56(154), 10-62(178)

is-io-link 10-52(133)

is-io-opt 2-37(118), 10-8(11), 12-6(23), 12-6(23)

is-io-prop-arim-th-data 10-61(174), 10-61(175)

is-io-prop-data 10-60(170), 10-72(222)

is-io-prop-data-blanks 10-60(170), 10-72(222)

is-io-qual 10-57(161), 10-57(163), 10-76(213)

is-io-real-const 10-55(142), 10-55(143), 10-55(143), 10-55(143), 10-56(149)

is-io-sign 10-55(142), 10-55(143), 10-55(143), 10-56(149), 10-56(153), 10-56(154)

is-io-string-apost 10-57(159), 10-57(160), 10-72(204)

is-io-string-char 10-56(145), 10-57(152)

is-io-sub 10-58(164), 10-60(170)

is-io-sub-blanks 10-58(164), 10-58(165), 10-60(170)

is-io-sub-part 10-57(161), 10-58(169)

is-isub 7-7(13), 7-18(43), 7-22(54), 7-28(68), 12-16(134), 12-17(140)

is-isub-def 7-25(60), 7-25(60), 7-28(68)

is-item 12-15(119), 12-15(120), 12-15(121), 12-15(122), 12-15(123)

is-key-match 10-24(61), 10-24(61), 10-25(63), 10-29(82), 10-39(104), 10-39(106), 10-43(113), 10-49(129)

is-key-to-key-match 10-24(61), 10-24(62), 10-25(63), 10-29(82), 10-29(82), 10-49(129)

is-keyed-data 2-40(128), 2-40(131), 2-40(133), 10-25(63)

is-keyed-seq-data 2-40(128), 2-40(133), 10-25(63)

is-label 5-13(18), 5-38(62), 5-38(62), 5-40(67), 5-40(67), 6-16(36), 12-7(37), 12-7(37)
is-label-den ........................................ 2-19 (43), 2-20 (42), 5-26 (46), 8-4 (6), 8-5 (15)
is-LEFT-PAR-blanks .................................. 10-58 (164), 10-60 (170)
is-letter ............................................. 8-4 (10), 8-4 (11), 8-59 (225), 10-57 (162)
is-list ................................................ 1-5 (51)
is-list-directed ....................................... 10-66 (184), 10-89 (249), 12-15 (118), 12-15 (121)
is-list-tr ............................................. 10-52 (134), 10-52 (136), 10-69 (193), 10-93 (258)
is-listdir-item ....................................... 10-60 (172), 10-72 (200)
is-listdir-item-leftcorr .............................. 10-60 (172), 10-71 (199)
is-listdir-proper-item ............................... 10-59 (169), 10-60 (171), 10-72 (200), 10-72 (201)
is-listdir-proper-item-corr .......................... 10-60 (172), 10-71 (199)
is-locate-st .......................................... 5-18 (28), 10-13 (33), 10-15 (37), 10-17 (40), 12-10 (74), 12-16 (130)
is-locked-foreign ................................... 10-35 (96), 10-35 (97), 10-35 (98)
is-locked-own ........................................ 10-34 (94), 10-34 (95)
is-m .................................................. 2-1 (1), 2-99 (193)
is-mi .................................................. 2-27 (96), 2-28 (97)
is-mstg ................................................ 2-7 (1), 2-25 (93), 3-3 (3), 6-11 (19), 6-19 (43)
is-mstg-applic ...................................... 3-9 (12), 5-11 (13), 5-44 (80), 6-4 (4), 6-8 (4), 6-4 (51), 6-8 (13), 6-8 (13)
is-named-array ..................................... 12-7 (31), 12-7 (31), 12-7 (32), 12-19 (163)
is-named-cell ....................................... 12-7 (31), 12-7 (31), 12-7 (36), 12-19 (165)
is-named-da .......................................... 12-5 (10), 12-6 (26), 12-6 (27), 12-7 (31), 12-7 (32), 12-7 (35), 12-19 (162)
is-named-io-cond ................................... 2-30 (100), 9-4 (1), 9-6 (9), 9-12 (19), 9-14 (26), 12-13 (103), 12-13 (105)
is-named-message .................................. 2-44 (143), 2-44 (143), 2-44 (144), 10-108 (294)
is-named-struct .................................... 12-7 (31), 12-7 (34), 12-19 (164)
is-named-succ ...................................... 12-7 (34), 12-7 (35), 12-7 (36)
is-ncl ................................................ 10-58 (167), 10-58 (167)
is-non-varying ..................................... 6-13 (26), 6-13 (26), 7-24 (61), 7-24 (61), 7-25 (60), 7-29 (74), 7-29 (75), 11-52 (175)
is-p-if-st .................................. 2-16 (32), 3-10 (17L), 5-18 (28), 5-28 (48), 5-28 (48), 9-14 (25)

is-p-on-st .................................. 3-10 (17L), 5-18 (28), 9-14 (25)

is-p-proc-st .................................. 2-16 (32), 3-10 (17L), 5-18 (28), 5-28 (48), 9-14 (25)

is-p-prop-st .................................. 3-10 (17L)

is-p-prop-var .................................. 3-10 (17L)

is-p-st .................................. 2-32 (105), 3-10 (17L), 5-17 (25), 5-20 (30), 5-28 (48), 5-28 (48), 9-12 (21)

is-p-while-group .................................. 3-10 (17L), 5-21 (33)

is-p .................................. 2-1 (1), 2-4 (1)

is-p-act .................................. 12-11 (83), 12-11 (84)

is-param-built-in .................................. 2-18 (40), 2-19 (41L), 5-37 (60), 5-38 (64), 5-55 (102), 5-55 (105),
7-24 (58), 11-2 (1), 11-4 (4), 11-8 (16)

is-paren-exp .................................. 7-7 (13), 7-18 (43), 7-22 (54), 12-16 (134), 12-17 (139)

is-pe .................................. 2-4 (2), 2-4 (15)

is-pic .................................. 2-19 (47), 5-23 (37), 5-38 (63), 6-11 (24), 7-30 (77), 7-30 (78), 8-3 (11), 8-8 (26),
8-9 (29), 8-36 (140), 8-38 (150), 8-39 (155), 8-40 (157), 8-42 (158),
12-6 (22), 12-7 (37), 12-8 (49), 12-9 (65), 12-18 (157)

is-pic-format .................................. 12-9 (61), 12-9 (63), 12-9 (63), 12-9 (65)

is-pic-match .................................. 8-63 (237), 10-85 (249), 10-85 (249)

is-pic-match-1 .................................. 10-84 (239), 10-84 (239), 10-85 (249)

is-pic-spec .................................. 8-45 (172)

is-POINT-blanks .................................. 10-57 (163), 10-59 (179)

is-pp .................................. 2-27 (96), 2-28 (98), 2-29 (98)

is-prefix-exp .................................. 5-41 (69), 5-41 (70), 5-41 (70), 7-7 (13), 7-18 (43), 7-22 (54), 12-16 (134),
12-18 (137)

is-prefix-operator .................................. 6-12 (46), 12-16 (137), 12-17 (139)

is-print-data .................................. 2-40 (128), 2-40 (129)

is-proc-st .................................. 3-10 (17), 12-4 (4), 12-10 (74)

is-prog-named-cond .................................. 2-29 (99), 9-4 (1), 9-6 (9), 9-12 (19), 12-13 (103), 12-13 (107)

is-program .................................. 3-3 (3), 12-9 (11)

is-program-decl-part .................................. 12-4 (1), 12-4 (2)

is-prop-arithmetic .................................. 8-8 (26), 8-9 (12)

is-prop-drift .................................. 8-48 (192), 8-59 (195)
<table>
<thead>
<tr>
<th>Symbol</th>
<th>References</th>
</tr>
</thead>
<tbody>
<tr>
<td>is-prop-exponent</td>
<td>10-56(152), 10-56(153), 10-72(203)</td>
</tr>
<tr>
<td>is-prop-pic</td>
<td>8-47(112), 8-51(196), 8-61(230), 10-27(71), 10-81(227)</td>
</tr>
<tr>
<td>is-prop-pic-1</td>
<td>8-44(165), 8-47(187), 8-47(187), 8-47(188)</td>
</tr>
<tr>
<td>is-prop-sh-pc</td>
<td>8-49(193), 8-49(193), 9-49(199)</td>
</tr>
<tr>
<td>is-prop-st</td>
<td>3-10(17), 12-10(71), 12-10(74)</td>
</tr>
<tr>
<td>is-prop-sterling-subf</td>
<td>8-48(192), 8-49(193)</td>
</tr>
<tr>
<td>is-prop-subf</td>
<td>8-47(188), 8-48(192)</td>
</tr>
<tr>
<td>is-prop-var</td>
<td>2-18(39), 3-10(17), 5-7(8), 5-47(86), 7-2(34), 7-24(58), 7-24(61), 9-6(8), 9-16(31), 10-67(187), 10-91(253), 12-5(9), 12-5(10), 12-19(161)</td>
</tr>
<tr>
<td>is-proper-dataset</td>
<td>2-40(127), 2-40(128), 2-42(139), 2-42(140), 2-43(141), 2-43(142), 10-17(39)</td>
</tr>
<tr>
<td>is-ps-gen</td>
<td>2-6(10), 2-36(116), 7-9(19), 7-11(251), 10-12(27)</td>
</tr>
<tr>
<td>is-pt</td>
<td>2-4(2), 2-4(4)</td>
</tr>
<tr>
<td>is.ptr-qual-ref</td>
<td>7-31(82), 9-16(31), 12-17(142), 12-17(193)</td>
</tr>
<tr>
<td>is.ptr-val</td>
<td>2-5(6), 2-21(52), 2-21(53), 2-21(54), 2-23(74), 2-24(78), 2-28(98), 2-37(118), 2-38(119), 6-19(44), 7-32(85), 7-33(87), 8-4(6), 8-5(15), 11-49(162)</td>
</tr>
<tr>
<td>is-put-st</td>
<td>5-18(28), 12-10(74), 12-19(115)</td>
</tr>
<tr>
<td>is-r-char</td>
<td>8-45(174), 8-5(175), 8-45(176)</td>
</tr>
<tr>
<td>is-read-st</td>
<td>5-18(28), 10-13(33), 10-17(40), 12-10(74), 12-15(124)</td>
</tr>
<tr>
<td>is-real-format</td>
<td>12-9(61), 12-9(62), 12-9(63), 12-9(63)</td>
</tr>
<tr>
<td>is-real-val</td>
<td>8-4(5), 8-4(8), 8-5(15)</td>
</tr>
<tr>
<td>is-real-vr</td>
<td>8-5(16), 8-6(17), 8-6(18), 8-6(18)</td>
</tr>
<tr>
<td>is-rec-opt</td>
<td>2-6(9), 2-6(10)</td>
</tr>
<tr>
<td>is-ref</td>
<td>2-6(8), 4-4(3), 4-5(7), 5-36(59), 5-55(103), 7-18(43), 7-22(54), 7-25(59), 10-12(28), 10-67(189), 10-89(249), 10-89(249), 10-91(253), 12-10(68), 12-11(79), 12-11(82), 12-11(82), 12-11(82), 12-12(86), 12-12(90), 12-14(114), 12-14(117), 12-15(123), 12-15(125), 12-15(126), 12-15(128), 12-16(134), 12-17(142), 12-17(143), 12-17(147)</td>
</tr>
<tr>
<td>is-remote-format</td>
<td>10-53(138), 12-9(60), 12-10(69)</td>
</tr>
<tr>
<td>is-reply</td>
<td>10-107(292), 10-107(292), 10-108(294)</td>
</tr>
<tr>
<td>is-rest-fu-file</td>
<td>2-16(117), 2-36(119)</td>
</tr>
<tr>
<td>is-rest-label</td>
<td>5-26(46), 12-7(42), 12-7(43)</td>
</tr>
<tr>
<td>is-ret-type</td>
<td>2-19(41), 12-4(3), 12-5(17), 12-6(22), 12-10(70)</td>
</tr>
</tbody>
</table>
is-return-st                      5-18(28), 9-14(25), 12-10(74), 12-11(85)
is-revert-st                     5-18(28), 12-10(74), 12-13(101)
is-rewrite-st                    5-18(28), 10-13(33), 10-15(37), 10-17(40), 12-10(74), 12-16(129)
is-RIGHT-PAR-blanks              10-58(164), 10-60(170)
is-s-main                        2-4(2), 2-4(31), 2-5(6), 2-10(16), 2-37(118), 2-38(119), 4-9(15), 4-9(16)
is-scalar                        5-38(62), 5-38(63), 5-38(63), 5-40(67), 5-40(67), 5-42(75), 5-42(75), 5-43(78), 5-43(78), 5-45(81), 6-5(7), 6-6(10), 6-11(21), 6-16(36), 6-16(36), 6-17(39), 7-4(3), 7-6(10), 7-7(13), 7-19(47), 7-22(55), 7-23(56), 7-23(56), 7-23(56), 7-23(56), 7-30(76), 7-36(76), 7-36(76), 10-68(190), 10-92(258), 10-92(254), 10-93(257), 11-5(11), 11-5(11), 11-5(11), 11-6(12), 11-9(18), 12-19(166), 12-20(173), 12-20(174)
is-scalar-da                     12-7(31), 12-7(137), 12-7(44), 12-19(166)
is-scalar-edu                    12-18(152), 12-19(157), 12-19(166)
is-scope                         12-5(10), 12-5(11), 12-5(17), 12-6(23)
is-semi-coapl                    4-16(39), 4-17(41), 4-17(42), 4-18(43)
is-SEMIC-blanks                  10-60(170), 10-60(171)
is-SEMIC-blanks-corr             10-60(171), 10-74(207)
is-seq-data                      2-40(128), 2-40(132)
is-set-read                      10-15(37), 10-37(100), 12-15(124), 12-15(126)
is-sign-mag-pic                  8-46(184), 8-47(187), 8-48(189), 8-53(202)
is-sign-pos                      8-45(178), 8-45(178), 8-45(178), 8-56(216), 8-56(216), 8-63(236)
is-sign-spec                     8-45(176), 8-45(178), 8-56(216)
is-signal-st                     5-18(28), 12-10(74), 12-13(102)
is-signed-subf                   8-62(235), 8-63(239)
is-simple-ref                    5-36(59), 5-55(102), 7-19(46), 11-7(15), 12-17(148)
is-size                          2-21(60), 2-40(133), 2-41(134)
is-size-cond                     8-7(19), 8-9(31), 8-9(32)
is-spec-area                     5-13(18), 6-6(10), 6-7(11), 12-19(169)
is-spec-area-da                  12-7(40), 12-7(41), 12-19(169)
is-spec-area-edu                 12-18(159), 12-19(160), 12-19(169)
is-st                            3-10(17), 12-10(69), 12-10(71), 12-11(76), 12-11(78), 12-11(79), 12-11(81), 12-11(81), 12-13(100)
is-st-type                       2-33(107), 2-33(108)
<table>
<thead>
<tr>
<th>Symbol</th>
<th>References</th>
</tr>
</thead>
<tbody>
<tr>
<td>is-state</td>
<td>2-141</td>
</tr>
<tr>
<td>is-sterl-dig-match</td>
<td>8-56 (214), 8-56 (214), 8-56 (215)</td>
</tr>
<tr>
<td>is-sterling-pic</td>
<td>8-42 (158), 8-44 (166), 8-44 (169), 8-47 (187), 10-97 (265), 12-8 (49)</td>
</tr>
<tr>
<td>is-sterling-spec</td>
<td>8-45 (172), 12-8 (52), 12-8 (53)</td>
</tr>
<tr>
<td>is-stg</td>
<td>12-20 (131), 12-21 (158), 2-26 (93)</td>
</tr>
<tr>
<td>is-stg-cl</td>
<td>5-18 (28), 12-10 (74), 12-12 (92)</td>
</tr>
<tr>
<td>is-stream-data</td>
<td>2-40 (128), 2-40 (129), 2-90 (130)</td>
</tr>
<tr>
<td>is-string</td>
<td>5-13 (18), 5-23 (37), 5-38 (62), 5-38 (62), 5-38 (62), 5-40 (67), 5-40 (67), 5-40 (67), 5-40 (67), 5-42 (75), 5-42 (75), 5-6 (10), 5-7 (11), 5-7 (11), 6-11 (24), 6-11 (24), 6-11 (24), 7-30 (76), 7-30 (76), 7-30 (76), 7-30 (76), 8-8 (26), 8-9 (33), 8-8 (78), 8-23 (93), 8-38 (150), 8-38 (151), 8-39 (155), 8-39 (156), 8-40 (157), 11-31 (44), 11-32 (45), 11-32 (45), 12-19 (167), 12-20 (173), 12-20 (174)</td>
</tr>
<tr>
<td>is-string-al</td>
<td>6-6 (10), 12-12 (93), 12-13 (97)</td>
</tr>
<tr>
<td>is-string-const</td>
<td>12-17 (148), 12-17 (150)</td>
</tr>
<tr>
<td>is-string-da</td>
<td>12-6 (22), 12-7 (37), 12-7 (39), 12-19 (167)</td>
</tr>
<tr>
<td>is-string-eda</td>
<td>12-20 (131), 12-18 (157), 12-18 (158), 12-19 (167)</td>
</tr>
<tr>
<td>is-string-format</td>
<td>12-9 (61), 12-9 (61)</td>
</tr>
<tr>
<td>is-string-get</td>
<td>10-64 (179), 12-14 (112), 12-14 (119)</td>
</tr>
<tr>
<td>is-string-put</td>
<td>10-88 (245), 12-14 (115), 12-14 (117)</td>
</tr>
<tr>
<td>is-string-type</td>
<td>5-13 (18), 5-23 (37), 5-38 (62), 5-38 (62), 5-38 (62), 5-40 (67), 5-40 (67), 5-40 (67), 5-40 (67), 5-42 (75), 5-42 (75), 5-6 (10), 5-7 (11), 5-7 (11), 6-11 (24), 6-11 (24), 6-11 (24), 7-30 (76), 7-30 (76), 7-30 (76), 7-30 (76), 8-8 (26), 8-9 (33), 8-8 (78), 8-23 (93), 8-38 (150), 8-38 (151), 8-39 (155), 8-39 (156), 8-40 (157), 11-31 (44), 11-32 (45), 11-32 (45), 12-19 (167), 12-20 (173), 12-20 (174)</td>
</tr>
<tr>
<td>is-struct</td>
<td>5-5 (6), 5-13 (18), 5-38 (62), 5-38 (62), 5-40 (67), 5-40 (67), 5-40 (67), 5-42 (75), 5-42 (75), 5-6 (10), 5-7 (11), 5-7 (11), 6-11 (24), 6-11 (24), 6-11 (24), 7-30 (76), 7-30 (76), 7-30 (76), 7-30 (76), 8-8 (26), 8-9 (33), 8-8 (78), 8-23 (93), 8-38 (150), 8-38 (151), 8-39 (155), 8-39 (156), 8-40 (157), 11-31 (44), 11-32 (45), 11-32 (45), 12-19 (167), 12-20 (173), 12-20 (174)</td>
</tr>
<tr>
<td>is-struct-al</td>
<td>6-6 (10), 12-12 (93), 12-12 (93)</td>
</tr>
<tr>
<td>is-struct-descr</td>
<td>12-7 (44), 12-8 (46), 12-19 (164)</td>
</tr>
<tr>
<td>is-struct-eda</td>
<td>12-18 (152), 12-18 (152), 12-19 (164)</td>
</tr>
<tr>
<td>is-subf-size-cond</td>
<td>8-53 (200), 8-53 (201), 8-62 (233), 8-62 (234), 8-62 (234)</td>
</tr>
<tr>
<td>is-succ-descr</td>
<td>12-8 (46), 12-8 (46), 12-8 (48)</td>
</tr>
</tbody>
</table>
is-succ-edata

is-td

is-te

is-title

is-title-descr

is-title-match

is-titled-dataset

is-tr

is-trail-sign

is-true-acc-num-comp

is-true-comp

is-true-num-comp

is-true-string-comp

is-ufl-cond

is-ua

is-unlock-arg

is-unlock-st

is-unrest-fu-file

is-usable

is-valid-da

is-valid-date

is-valid-single-char

is-value

is-var

is-var-descr

is-var-edescr

is-vr

is-wait-st
| is-while-group | 3-10(17), 12-11(77), 12-11(79) |
| is-write-st | 5-18(28), 10-13(33), 10-15(37), 10-17(40), 12-10(74), 12-15(129) |
| is-xdec-pic | 8-44 (167), 8-44(169), 8-48(192), 8-48(193), 8-55(206) |
| is-xnum-pic | 8-44(167) |
| is-xpic | 8-42(159) |
| is-xsterling-pic | 8-44(167), 8-44(169), 8-48(192), 8-48(193), 8-50(195), 8-50(195), 8-53(201), 8-55(206), 8-57(217), 8-64(280) |
| is-1-compl-pic | 8-47(187), 8-48(191), 8-54(205), 8-63(236) |
| is-2-compl-pic | 8-47(187), 8-48(191), 8-53(201), 8-54(205), 8-63(236) |
| isub-env | 7-25(60), 7-28(69) |
| isub-val | 7-18(43), 7-19(95) |
| iterate-assign | 7-4(3), 7-5(6), 7-5(6) |
| iterate-byname | 7-6(10), 7-6(11), 7-6(11) |
| iterate-do | 5-23(37), 5-24(39), 5-25(84) |
| iterate-dummy-assign | 5-45(81), 5-45(82), 5-45(82) |
| iterate-free | 4-11(27), 4-11(28), 4-11(28) |
| iterate-get | 10-68(190), 10-68(191), 10-68(191) |
| iterate-infix-op | 11-45(138), 11-45(139), 11-45(140), 11-45(141), 11-46(142), 11-46(142) |
| iterate-opl | 11-6(12), 11-6(13), 11-6(13) |
| iterate-put | 10-92(254), 10-92(255), 10-92(255) |
| iterate-spec-list | 5-23(36), 5-23(37), 5-23(37) |
| J-CHAR | 8-4(11) |
| K-CHAR | 8-4(11) |
| KEY | 2-31(101), 10-31(87), 10-39(104), 10-43(113), 10-49(129), 12-13(106) |
| key-buffer-keyto | 10-29(79), 10-45(118) |
| keyto-assign | 10-26(67), 10-26(68), 10-29(79) |
| L-CHAR | 8-4(11) |
I B fll LAB V I E N N A

r�erge-pref

•

m e rge-s u h f

•

TR 2 5 . 0 8 2

AT-46

5- 7 { 8) , 5- 7 ( 8) . �- 7 1.2t , 5-9 { 1 2 ) , 5- 1 8 ( 2 7)
•

8- 5 8 ( 2 1 9) , �- 5�111Ql
10-106 (289}

P! ES S - LE NG T R

1::11J !�l

min

1 1 -9 ( 1 9 ) , 1 1 - 1 0 ( 2 11)

MIN
m irl- f l t

•

•

8- 1 ( A ) , .!!:.!11§!1

s - 4 1 ( 6 9 ) , 5 - 4 1 c7 o ) , 8 - 4 ( 9 l , 8 - 1 2 c 4 7) , a- 2 a ( 1 1 o• , 8- 2 8 < 1 1 21 , 8- 2 q ( 1 1 3) •
8 -2 9 ( 1 1 7 ) , 8 - 3 2 ( 1 2 4 ) , 8- 4 5 ( 1 7 6 ) , 8 - 4 6 ( 1 8 0) , 8- 5 3 ( 2 0 3) , 8- 55 ( 2 1 0, ,
8 - 5 6 ( 2 1 6 ) , 8 -6 0 ( 2 2 8 ) , 8 - 6 0 ( 2 2 8) , 8 - 6 0 ( 2 2 9 ) , 1 0 - 5 6 ( 1 44) , 1 0 - 6 2 ( 1 78t .
1 0 - 6 2 ( 1 7 8 ) , 1 0 -9 9 ( 2 1 0 l , 1 0 - 1 0 3 ( 2 8 1 , , 1 0- 1 0 3 { 2 8 1) , 1 1 - 3 8 ( 8 3) , 1 2- 8 ( 51 } •
1 2- 1 7 ( 1 3 8l

IHN U S

6- 7 { 1 1 ) , 6-7 ( 1 1 ) , 6 - 7 { 1 1 ) , 6 - 7 ( 1 1 ) . �::11111

mix
a i x-spec

6- 3 ( 2)

•

; 6 - 7 ( 1 1 , , 6-7 { 1 1 ) • §:.111!l , 6 - 7 ( 1 1 }

8- 1 4 ( 6 5 } , A- 2 2 ( 9 1 ) , 8 - 2 2 ( 9 1 ) , 8 - 2 6 ( 1 0 5) , 8- 3 4 ( 1 29)

m lt - a r i t h m

•

8- 3 7 ( 1 4 3 ) , .!! :.1!H1!2t.
8 - 3 9 ( 1 53 ) , 8-4 1 ( A )
8 - 3 a c 1 4 9 ) , �:.§.H112l

m k - a r it h m - p i c

1 0 - 7 5 ( 2 1 0) , 1Q:.lli�!.!! l r 1 1 - 2 ( 3 ) , 1 1 - 2 ( 3 ) , 1 1 - 1 2 ( 2 7).
8- 3 7 ( 1 4 3) , 8 - 3 8 ( 1 50) , J!:.J2 1 1�Jl.

Il k -b i t -s t r in g
•

m k - c bar- stri n g

•

•

•

•

•

-

.;

1 1 - 2 ( 2 ) . !1:.1111

8- 2 2 ( 9 1 ) , 8- 2 2 { 9 1 ) , 8 - 3 7 ( 1 4 3 ) , 8 - 3 8 ( 1 50 )

•

•

•

-

•

•

•

•

•

.!!:.J .!!1 !.2ll

1 0 -8 2 ( 2 2 9) , 1Q:.!!21l!!1l

1 0 - 8 9 ( 2 4 9t , !Q:.2QJ22!L. 1 0- 9 0 ( 2 5 1 t

ll k - d a t a d i r - r e f - l i s t

1 0- 6 9 ( 1 9 3 ) , 1 0- 8 2 ( 2 2 9) , ! Q:.!!�i��2.l. 1 0- 8 2 ( 2 2 9 )
m k - e l i m - r e f- l i s t
r� k- f -f i e l d

•

•

•

•

•

•

•

•

•

•

•

1 0- 9 0 ( 2 5 1 ) , 1Q:.2. 111�1l

1 0 - 1 0 3 ( 2 8 1 ) , !Q:.!Q!1l.!!ll r 1 0- 1 0 4 ( 2 8J )

•

m k - g en

mk-id

•

•

•

3 -9 ( 1 2 ) , 5 - 1

1

( 1 l} , 5 - 1 1 ( 1 3 ) , 5 - 4 4 ( A 0) ,

5- 4 4 ( 80) , 6 - 4 { 4)

, 6- 4 ( 4) , .§:.21§1 ,
6 - 8 ( 1 3 ) , 6- 9 ( 1 5 )

5- 3 9 ( 6 5) , 5- 3 9 ( 6 5) , 7- 1 2 ( 2 7) , 7- 1 2 ( 2 7) , 7- 1 2 ( 271 , 7- 1 2 ( 2 7) , 7- 1 2 ( 2 7) ,
7- 1 2 ( 2 7) , 7- 1 2 ( 27) , 7- 1 2 ( 2 7) , 7- 1 2 ( 2 7) , 7 - 1 2 ( 2 7) " 7- 1 2 { 2 7) , 7 - 1 7. { 2 7 ) ,
7- 1 2 ( 2 9 ) , 7 - 1 2 ( 2 9 ) , 7- 1 2 ( 2 9l , 7 - 1 2 ( 2 9 ) , 7- 1 2 ( 2<J) , 7- 1 2 ( '). 9) , 1- t 2 ( 2 9) ,
7- 1 2 ( 2 9 ) , 1- 1 2 { 2 9 ) , 7- 1 2 ( 2 9 ) , 7- 1 2 ( 2 9 l , 9- 1 6 ( 3 H , 1 0- 8 ( A ) , 1 0- 9 ( A ) ,
1 o- 1 1 ( 2 41 , 1 o- 1 6 (3 AI , 1 o- 2 1 c 5 1 ) , 1 o- 2 1 c 5 1 , , 1 o-· 2 1 c 5 1 , , 1 o - 2 1 c 5 1 ) ,
1 0- 2 1 ( 7 0) , 1 0- 30 ( 8 61 , 1 o - 7 5 ( 2 1 21 , 1 o-76 ( 2 1 4 ) , 1 0 - 7 6 ! 2 1 4 ) , !Q.=I§HHl. ,
1 0-7 6 ( 2 1 4 ) , 1 0 - 7 6 ( 2 1 s ) , 1 0 - 9 6 ( 26 2) , 1 0 - 9 6 ( 2 6 2l , 1 0- 9 6 ( 2 6 2) , 1 1 - 4 ( 6) ,
1 1 - 4 ( 6 l , 1 1 - 9 { 1 9) , 1 1 - 9 ( 1 9 ) , 1 1 - 1 0 ( 2 4) , 1 1 - 1 0 ( 2 4 ) , 1 1 - 1 2 ( 2 5 I , 1 1 - 1 2 C 2 5 ) ,
1 1 - 1 2 ( 2 5, , 1 1 - , 2 ( 2 5) , 1 , - 1 2 ( 2 5) , 1 1 - , 2 ( 2 5 ) , 1 1 - 1 2 ( 2 5 ) , 1 1 - 1 2 { 2 5 ) ,
1 1 - 1 2 ( 2 5 1 , 1 1 - 1 2 ( 2 5) , 1 1 - 1 2 ( 2 5) , 1 1 - 1 2 < 2 5 ) , 1 1 - 1 2 ( 2 5 ) , 1 1 - 1 2 ( 2 5 ) ,
1 1 - , 2 (2 5) , , - 1 2 ( 2 5) • 1 1 - 1 2 { 2 5 ) , , 1 - 1 2 ( 2 5t , 1 ,- 1 2 ( 2 5 )
•


<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
<th>Entries</th>
</tr>
</thead>
<tbody>
<tr>
<td>ak-infix</td>
<td></td>
<td>7-7(13), 7-8(15)</td>
</tr>
<tr>
<td>ak-io-data-op</td>
<td></td>
<td>10-71(199), 10-72(204), 10-76(217), 10-77(219)</td>
</tr>
<tr>
<td>ak-list</td>
<td></td>
<td>1-21(221)</td>
</tr>
<tr>
<td>ak-list-1</td>
<td></td>
<td>1-21(221)</td>
</tr>
<tr>
<td>ak-m-field</td>
<td></td>
<td>10-103(281), 10-104(283)</td>
</tr>
<tr>
<td>ak-num-list</td>
<td></td>
<td>8-53(202), 8-55(206)</td>
</tr>
<tr>
<td>ak-num-list-1</td>
<td></td>
<td>8-55(206), 8-55(206), 8-55(206), 8-55(207), 8-55(207), 8-55(208)</td>
</tr>
<tr>
<td>ak-num-string</td>
<td></td>
<td>10-97(264), 10-97(265), 10-98(267)</td>
</tr>
<tr>
<td>ak-op</td>
<td></td>
<td>8-11(199), 8-30(119), 10-72(204), 10-72(204), 10-82(229), 10-85(241), 10-85(242), 10-89(248), 11-33(53), 11-38(80), 11-38(81), 11-38(82), 11-45(133), 11-47(151), 11-52(175)</td>
</tr>
<tr>
<td>ak-paren</td>
<td></td>
<td>7-7(13), 7-8(15)</td>
</tr>
<tr>
<td>ak-pc-op</td>
<td></td>
<td>10-82(229), 10-85(241)</td>
</tr>
<tr>
<td>ak-prefix</td>
<td></td>
<td>7-7(13), 7-8(15)</td>
</tr>
<tr>
<td>ak-pg-gen</td>
<td></td>
<td>7-12(27), 7-12(27), 7-12(28)</td>
</tr>
<tr>
<td>ak-ref-list</td>
<td></td>
<td>9-16(31), 9-17(33)</td>
</tr>
<tr>
<td>ak-ref-opl</td>
<td></td>
<td>10-75(210), 10-76(217), 10-76(217)</td>
</tr>
<tr>
<td>ak-r1</td>
<td></td>
<td>6-16(36), 7-25(59), 7-26(62), 7-26(62), 7-26(62), 10-92(256)</td>
</tr>
<tr>
<td>ak-r1-1</td>
<td></td>
<td>7-26(62), 7-26(63), 7-26(63), 7-26(63)</td>
</tr>
<tr>
<td>ak-sterl-subl</td>
<td></td>
<td>8-55(206), 8-55(206), 8-55(206)</td>
</tr>
<tr>
<td>ak-str-1</td>
<td></td>
<td>8-38(151), 8-39(153), 8-39(155)</td>
</tr>
<tr>
<td>ak-string</td>
<td></td>
<td>8-22(91), 8-22(91), 8-37(143), 8-38(159)</td>
</tr>
<tr>
<td>ak-te</td>
<td></td>
<td>4-3(11), 4-4(21)</td>
</tr>
<tr>
<td>ak-x-field</td>
<td></td>
<td>10-103(281), 10-105(283)</td>
</tr>
<tr>
<td>mod</td>
<td></td>
<td>5-45(82), 7-5(6), 7-5(6), 7-6(11), 7-6(11), 7-7(13), 7-7(13), 7-7(13), 7-7(13), 7-7(13), 7-7(13), 10-68(191), 10-92(255), 11-2(21), 11-6(13)</td>
</tr>
<tr>
<td>mod-builtins</td>
<td></td>
<td>7-7(13), 11-2(11)</td>
</tr>
<tr>
<td>mod-builtins-1</td>
<td></td>
<td>11-2(11), 11-2(11)</td>
</tr>
<tr>
<td>mod-op-string</td>
<td></td>
<td>10-97(264), 10-98(265), 10-98(268)</td>
</tr>
<tr>
<td>mod-param-builtin</td>
<td></td>
<td>11-2(11), 11-12(261)</td>
</tr>
<tr>
<td>mod-string</td>
<td></td>
<td>10-98(267), 10-98(269)</td>
</tr>
</tbody>
</table>
modulo
modulo-1
mpic
mpic-1
msubf
MULT
MULT
M-CHAR
n-of-d
n-of-d-subf
NAME
NE
new
new-position
new-position-delete
new-position-direct
new-position-segu
new-1
NO
no-area-cond
non-string-vr-set
NORMAL
NOT
note-buffer
note-delay
note-key
NPTR
11-41(118), 11-41(118), 11-41(119), 11-41(119), 11-41(119),
11-42(123), 11-42(123), 11-42(123), 11-43(124), 11-43(124),
11-43(125), 11-43(125), 11-43(127), 11-44(135), 11-44(135),
11-44(135), 11-49(158), 11-49(158), 11-49(159), 11-49(159),
11-51(172), 11-53(177), 11-53(178).

OPEN

open

open-associate

open-fd-1

open-fd-2

open-fu

open-intg-test

OPL

opt-list

OR

order

order-set

OUT

out-fix-pic

out-fit-pic

overpunch

OWN

P

P-CHAR

PACKED

PAGE

PAGE-DELIMITER

PARAM
PASS

PERC

permute

PIC

pic-length

pic-prec

pic-val

pic-val-1

PLUS

POINT

pos

ppic

prec

prec-fact

pref

pref-ar-da

pref-base

pref-mode

pref-prec

pref-scale

pref-scale-f

pref-str-base

pref-str-da
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>8-51(196), 8-51(197), 8-59(223), 8-59(226), 8-59(226)</td>
<td>8-51(196), 8-51(197), 8-59(223), 8-59(226), 8-59(226)</td>
<td>8-9(29), 8-9(29), 8-9(29), 8-9(31), 10-72(204), 10-72(204), 10-72(204), 10-82(229)</td>
<td>8-52(198), 8-52(198), 8-52(198), 8-52(199), 8-52(199), 8-52(199), 8-52(199), 8-52(200)</td>
<td>8-53(202), 8-56(216), 8-56(216), 8-56(214), 8-56(214), 8-56(214)</td>
<td>8-55(210), 8-56(212), 8-56(214), 8-56(214), 8-56(214)</td>
<td>8-55(210), 8-56(212), 8-56(214), 8-56(214)</td>
<td>8-53(200), 8-53(201), 8-62(233), 8-62(234), 8-62(234)</td>
<td>8-55(210), 8-56(212), 8-56(214), 8-56(214)</td>
<td>8-53(200), 8-53(201), 8-62(233), 8-62(234), 8-62(234)</td>
<td>8-14(66), 8-15(67), 8-26(105), 8-27(107)</td>
<td>8-14(66), 8-15(67), 8-22(88), 8-25(101), 8-25(101), 8-25(102)</td>
<td>8-25(101), 8-25(102)</td>
<td>8-25(101), 8-25(102)</td>
<td>7-25(60), 7-28(72)</td>
<td>8-9(33), 8-10(35)</td>
<td>7-20(49), 7-20(51), 7-24(58), 7-24(58)</td>
<td>2-33(108), 12-11(85)</td>
<td>5-53(97), 5-53(97), 5-53(97), 5-53(97), 5-53(98), 8-53(199)</td>
<td>7-20(49), 7-20(49), 7-20(50), 9-17(34)</td>
<td>2-33(108), 12-13(101)</td>
<td>9-7(12), 9-7(12), 9-7(12), 9-7(12), 9-7(12), 9-7(12), 9-7(12), 9-7(13), 9-7(14)</td>
<td>2-33(108), 12-16(129)</td>
</tr>
</tbody>
</table>
rt-builtin
rt-builtin-1
rt-fixed
rt-mod
rt-repeat
rt-round
rt-string
rt-trunc
rt-unspec
BWR-DIR
BWR-UNBUF
S
S-CHAR
S-INTG
search
search-1
sel
select-title
SEMIC
SEQ
segu-buffer
segu-into
segu-kerto
set-wait-state
set-wait-state-1
set-wait-state-2
set-wait-state-3
sf-set
<table>
<thead>
<tr>
<th>Raw Text</th>
<th>Natural Text</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>sign</code></td>
<td><code>8-45(176), 8-46(180), 8-46(184), 8-48(189), 8-54(204), 8-56(216), 8-60(229), 12-8(51), 12-9(55)</code></td>
</tr>
<tr>
<td><code>SIGN</code></td>
<td><code>8-32(123), 8-36(138), 8-39(139)</code></td>
</tr>
<tr>
<td><code>sign-and-left-adjust</code></td>
<td><code>8-32(123), 8-36(138), 8-39(139)</code></td>
</tr>
<tr>
<td><code>sign-char</code></td>
<td><code>8-33(126), 8-53(202), 8-53(203)</code></td>
</tr>
<tr>
<td><code>SIGNAL</code></td>
<td><code>2-33(108), 9-8(15), 9-8(15), 9-12(19), 12-13(102)</code></td>
</tr>
<tr>
<td><code>SIMP-REF</code></td>
<td><code>11-7(15), 11-19(38)</code></td>
</tr>
<tr>
<td><code>single-bit-char</code></td>
<td><code>8-36(138), 8-39(139)</code></td>
</tr>
<tr>
<td><code>single-bit-num</code></td>
<td><code>8-20(82), 8-20(83), 8-20(85), 8-35(136), 8-35(137)</code></td>
</tr>
<tr>
<td><code>single-char-bit</code></td>
<td><code>8-35(134), 8-35(135), 10-72(204), 10-81(3)</code></td>
</tr>
<tr>
<td><code>single-char-num</code></td>
<td><code>8-34(133), 10-62(178)</code></td>
</tr>
<tr>
<td><code>single-infix-bit</code></td>
<td><code>8-21(86), 8-24(87)</code></td>
</tr>
<tr>
<td><code>single-num-bit</code></td>
<td><code>8-34(131), 8-34(132)</code></td>
</tr>
<tr>
<td><code>single-num-char</code></td>
<td><code>8-33(127), 8-34(128), 8-34(133), 8-55(210), 8-56(213), 10-99(271)</code></td>
</tr>
<tr>
<td><code>single-prefix-not</code></td>
<td><code>8-29(114), 8-32(115)</code></td>
</tr>
<tr>
<td><code>SIZE</code></td>
<td><code>2-29(99), 2-31(103), 8-9(31), 8-53(200), 9-4(1), 9-11(18), 9-12(19), 10-105(285), 12-13(103)</code></td>
</tr>
<tr>
<td><code>size-1</code></td>
<td><code>2-22(62), 6-9(15), 6-11(20), 8-31(103), 8-31(103), 8-53(200), 9-4(1), 9-11(18), 9-12(19), 10-105(285), 12-13(103)</code></td>
</tr>
<tr>
<td><code>SKIP</code></td>
<td><code>10-69(195), 10-93(259), 10-93(259), 12-10(67)</code></td>
</tr>
<tr>
<td><code>SLASH</code></td>
<td><code>8-4(9), 8-46(141), 8-49(193), 9-49(193), 12-8(51)</code></td>
</tr>
<tr>
<td><code>snap-action</code></td>
<td><code>9-13(22), 9-13(22)</code></td>
</tr>
<tr>
<td><code>SPACE</code></td>
<td><code>10-69(195), 10-93(259), 12-10(67)</code></td>
</tr>
<tr>
<td><code>stack</code></td>
<td><code>5-3(1), 5-4(2), 5-50(89), 9-5(6), 9-13(24)</code></td>
</tr>
<tr>
<td><code>stand-check</code></td>
<td><code>9-5(6), 10-105(285)</code></td>
</tr>
<tr>
<td><code>STAND-PRINT</code></td>
<td><code>2-34(110), 10-8(4), 10-18(42), 10-22(55), 10-22(55), 11-51(170)</code></td>
</tr>
<tr>
<td><code>start-line</code></td>
<td><code>10-100(274), 10-101(275), 10-101(277)</code></td>
</tr>
<tr>
<td><code>STATIC</code></td>
<td><code>3-8(11), 5-9(12), 12-5(12)</code></td>
</tr>
<tr>
<td><code>static-initialize</code></td>
<td><code>3-7(10), 3-9(13)</code></td>
</tr>
</tbody>
</table>
target

target-1

target-2

TASK

...
tk-es ................................. 10-6(17), 10-7(18)
tk-es-transmit ........................ 10-6(16), 10-6(17), 10-20(50)
tk-onsource .............................. 10-70(197), 10-71(198), 10-78(222)
tk-stand-print-u ........................ 10-6(16), 10-20(286), 10-106(287)
tk-str-data ............................... 10-88(245), 10-89(249)
trace-wait-st ............................ 2-10(15), 2-12(23)
TRANSMIT ................................ 2-31(101), 10-30(84), 12-13(106)
transmit ................................. 10-6(12), 10-6(15), 10-6(17)
trunc ...................................... 1-11(161)
truth-to-bit .............................. 8-15(67), 8-15(67)
truth-val ................................... 5-20(31), 5-20(32), 5-24(40), 5-24(41)
U-CHAR .................................... 8-4(11)
ubd ........................................ 5-45(82), 6-11(21), 6-12(22), 6-17(39), 7-5(6), 7-6(11), 7-30(76), 7-30(77),
                                  10-68(191), 10-92(255), 12-20(173)
UPL ........................................ 2-29(99), 2-31(103), 8-17(73), 9-4(1), 9-11(18), 9-12(19), 9-12(19),
                                  12-13(103)
un-name ................................... 2-17(361), 3-7(8), 3-7(9), 4-4(2), 4-4(3), 4-15(31), 5-5(4), 5-9(12),
                                  5-41(71), 5-43(77), 5-46(85), 7-20(48), 7-20(49), 9-17(34), 10-17(39),
                                  10-32(88), 10-64(179), 10-88(245), 10-106(288)
UNBUF ..................................... 2-44(145), 10-2(11), 10-2(11), 10-2(11), 10-2(11), 10-2(11),
                                  10-2(11), 10-38(102), 10-40(108), 10-44(115), 10-44(115), 10-48(126),
                                  12-6(24)
UNDF ...................................... 2-31(101), 9-8(15), 9-8(15), 10-11(24), 10-17(40), 12-13(106)
UNLOCK .................................... 2-33(108), 12-16(132)
unlock ..................................... 10-35(291), 10-38(103), 10-41(109), 10-49(128)
unlock-taskend ............................ 4-9(18), 9-11(23)
unnest-cl ................................ 10-59(168), 10-59(168), 10-59(168), 10-60(171), 10-62(178),
                                  10-75(212)
UNSPEC .................................... 7-12(27), 7-12(29), 10-21(51)
 unspec-length ............................ 7-12(29), 7-17(42), 11-44(134)
unstack .................................. 5-4(3), 5-14(20), 9-5(6)
UPD ........................................ 2-44(145), 10-2(11), 10-2(11), 10-2(11), 10-2(11), 10-2(11), 10-2(11), 10-13(33),
                                  10-13(33), 10-13(33), 10-15(37), 10-15(37), 10-15(37), 12-6(24)
null
<p>| val-op-1   | 11-33 (53), 11-34 (54), 11-35 (58), 11-35 (60), 11-35 (61), 11-36 (68), 11-36 (69), 11-36 (70), 11-45 (143) |
| valid-init-state | 3-2 (2), 3-3 (3) |
| value      | 8-8 (26), 8-8 (26), 8-8 (26), 8-11 (40), 8-34 (130) |
| value-conv | 8-30 (119), 8-31 (120) |
| varying    | 11-20 (39), 11-30 (41), 11-42 (123), 11-43 (124) |
| verify-f-e | 10-82 (229), 10-83 (231), 10-83 (231) |
| verify-file-name | 10-11 (21), 10-12 (22), 10-13 (32), 10-21 (52) |
| verify-impl-open | 10-13 (32), 10-14 (34) |
| verify-pic  | 10-82 (229), 10-84 (239), 10-84 (239) |
| verify-remote-format | 10-53 (139), 10-54 (139) |
| verify-transmit | 10-71 (19), 10-17 (41), 10-22 (54), 10-68 (192), 10-92 (254) |
| verify-width | 10-102 (280), 10-103 (283) |
| vr-rf      | 10-102 (280), 10-103 (283) |
| vr-set     | 6-11 (29), 6-12 (23), 8-8 (23), 8-9 (24), 8-8 (25) |
| w-char     | 8-4 (11) |
| wait       | 2-33 (108), 12-12 (86) |
| wait-count | 4-15 (33), 4-16 (36) |
| wait-delay | 4-21 (50), 4-21 (52), 4-21 (52) |
| wait-for-unlock | 10-35 (96), 10-35 (96), 10-38 (103), 10-41 (109), 10-44 (116), 10-49 (129) |
| while-test | 5-21 (34), 5-22 (35) |
| wr-dir     | 10-33 (92), 10-38 (102) |
| wr-unbuf   | 10-33 (92), 10-38 (102) |
| write      | 2-33 (108), 12-15 (128) |
| write-direct | 10-33 (92), 10-38 (102) |
| write-direct-1 | 10-38 (103), 10-39 (104), 10-41 (109) |</p>
<table>
<thead>
<tr>
<th>Column</th>
<th>Characters</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>write-seq</strong></td>
<td>10-38(102), 10-39(105), 10-39(106), 10-40(108)</td>
</tr>
<tr>
<td><strong>write-unbuf</strong></td>
<td>10-33(92), 10-39(105), 10-41(110)</td>
</tr>
<tr>
<td><strong>X-CHAR</strong></td>
<td>8-4(11), 8-59(225), 12-9(57)</td>
</tr>
<tr>
<td><strong>xout-fix-pic</strong></td>
<td>9-60(227), 9-60(228)</td>
</tr>
<tr>
<td><strong>xpic</strong></td>
<td>8-44(165), 8-51(197), 8-61(231), 8-63(237), 8-64(239)</td>
</tr>
<tr>
<td><strong>Y-CHAR</strong></td>
<td>8-4(11), 8-43(164), 8-45(174), 8-49(194), 8-50(195), 8-50(195), 8-55(210), 12-8(51)</td>
</tr>
<tr>
<td><strong>Z-CHAR</strong></td>
<td>2-29(99), 2-31(103), 8-16(69), 9-4(1), 9-11(18), 9-12(19), 12-13(103)</td>
</tr>
<tr>
<td><strong>ZDIV</strong></td>
<td>3-3(3), 4-6(10), 8-5(14), 8-10(37), 8-18(77), 8-21(87), 8-28(115), 8-29(115), 8-34(132), 8-35(135), 8-35(137), 8-36(139), 8-54(204), 10-107(290), 11-45(137), 11-45(137), 11-45(137), 11-45(137), 11-45(137), 11-50(167)</td>
</tr>
<tr>
<td><strong>1-BIT</strong></td>
<td>1-12(30), 4-16(39), 4-20(49), 5-20(32), 7-9(20), 7-16(37), 8-5(14), 8-18(77), 8-21(87), 8-21(87), 8-21(87), 8-21(87), 8-21(87), 8-21(87), 8-29(115), 8-29(115), 8-34(132), 8-35(135), 8-35(137), 8-36(139), 8-54(204), 11-45(137), 11-45(137), 11-45(137), 11-45(137), 11-50(167)</td>
</tr>
<tr>
<td><strong>1-CHAR</strong></td>
<td>8-5(12), 8-33(128), 8-35(134), 8-35(134), 8-35(135), 8-36(139), 8-44(189), 8-48(189), 10-56(148), 12-9(55)</td>
</tr>
<tr>
<td><strong>2-CHAR</strong></td>
<td>8-5(12), 8-33(128), 8-48(190), 12-9(55)</td>
</tr>
<tr>
<td><strong>3-CHAR</strong></td>
<td>8-5(12), 8-33(128), 8-48(191), 12-9(55)</td>
</tr>
<tr>
<td><strong>4-CHAR</strong></td>
<td>8-5(12), 8-33(128)</td>
</tr>
<tr>
<td><strong>5-CHAR</strong></td>
<td>8-5(12), 8-33(128)</td>
</tr>
<tr>
<td><strong>6-CHAR</strong></td>
<td>8-5(12), 8-33(128), 8-45(173), 8-49(194), 8-55(208), 8-55(210), 8-56(215), 12-8(53)</td>
</tr>
<tr>
<td><strong>7-CHAR</strong></td>
<td>8-5(12), 8-33(128), 8-45(173), 8-49(194), 8-55(208), 8-55(210), 8-56(215), 12-8(53)</td>
</tr>
<tr>
<td><strong>8-CHAR</strong></td>
<td>8-5(12), 8-33(128), 8-45(173), 8-50(195), 8-59(225), 8-60(228), 8-60(229), 8-60(229), 12-8(51), 12-9(57)</td>
</tr>
<tr>
<td><strong>9-CHAR</strong></td>
<td>8-5(12), 8-33(128), 8-45(174), 8-50(195), 8-59(225), 8-60(228), 8-60(229), 12-8(51), 12-9(57)</td>
</tr>
<tr>
<td><strong>Γ</strong></td>
<td>10-4(5), 10-4(6), 10-4(7)</td>
</tr>
<tr>
<td><strong>Λ</strong></td>
<td>2-10(15), 2-11(22)</td>
</tr>
<tr>
<td><strong>Ψ</strong></td>
<td>2-10(15), 2-11(22)</td>
</tr>
</tbody>
</table>
APPENDIX II: LIST OF ELEMENTARY OBJECTS

This index lists all explicitly named elementary objects used in the document.

A-CHAR
ABNL
ADD
AGGR
ALIGNED
ALLOCATE
AND
APOSTR
AREA
ARR
ASSIGN
ASTER
ATAN
ATAND
AUTO
B-CHAR
BACK
BIN
BIT
BLANK
BLOCK
BREAK
BUF
BUILTIN
C-CHAR
C-EXP
CALL
CARR-RETURN
CAT
CHAR
CHECK
CLOSE
COLON
COLUMN
COMM-AT
COMMA
COMPLETION
COMPLEX
COND
CONST
CONV
CPLX
CTL
D-CHAR
DEC
DEF-ENVIRONMENT
DEF-LINESIZE
DEF-PAGESIZE
DEF-PREC-BIN
DEF-PREC-DEC
DEL-KEYED
DEL-UNBUF
DELAY
DELETE
DIR
DISPLAY
DIV

DOLLAR
DUMMY
E-CHAR
EDA
EMPTY-AREA
ENDFILE
ENDPAGE
ENTRY
EQ
ERROR
EVENT
EXCL
EXIT
EXP
EXP-SIZE
EXPND
EXT
P-CHAR
PILE
FILEMARK
FINISH
PTX
PLT
POPL
FREE
G-CHAR
GE
GEN
GET
GOTO
GT
H-CHAR
I-CHAR
IF
IGN
IMAG
IN
INH
INT
INTG
IRREL
J-CHAR
K-CHAR
KEY
KEYED
LABEL
LE
LEFT-PAR
LEPT-PAR
LIN
LIN-ARR
LINE
LINE-DELIMITER
LOCATE
LT
M-CHAR
MAX

MAX-LENGTH-ID
MESS-LENGTH
MTN
MINUS
MULT
N-CHAR
NAME
NE
NO
NORMAL
NOT
NULL
NUM
NUMBER-SIGN
O-CHAR
OFFSET
OP
ON
ONCHAR
ONKEY-LENGTH
ONSOURCE
OP
OPEN
OPL
OR
OUT
OWN
P-CHAR
PACKED
PAGE
PAGE-DELIMITER
PARAM
PERC
PIC
PLUS
POINT
PRI-PREC
PRINT
PRINT
PRIORITIES
PROC
PTR
POT
Q-CHAR
QUEST
R-CHAR
RD-DIR
RD-UNBUF-UNKEYED
READ
READ
REAL
RECORD
REMOTE
RETURN
REVERT
REWRITE
RIGHT-PAR
RWR-DTR
RWR-UNBUF
S-CHAR
S-INTG
SEMIC
SEQ
SIGN
SIGNAL
SIMP-REF
SIZE
SKIP
SLASH
SPACE
STAND-PRINT
STATIC
STATUS
STOP
STREAM
STRG
SUBRG
SUBSTR
SUBTR
SYSTEM
T-CHAR
TAB-SET
TABULATOR
TASK
TEXT
TRANSMIT
U-CHAR
UF
UNBUF
UNDP
UNLOCK
UNSPEC
UPD
V-CHAR
VARYING
W-CHAR
WAIT
WR-DIR
WR-UNBUF
WRITE
X-CHAR
Y-CHAR
Z-CHAR
ZDIV
0-BIT
1-BIT
1-CHAR
2-CHAR
3-CHAR
4-CHAR
5-CHAR
6-CHAR
7-CHAR
8-CHAR
9-CHAR