

IPL//

TECHNICAL REPORT

TR 25.097
30 June 1969

ULD

VERSION

III

**TRANSLATION OF PL/I
INTO ABSTRACT SYNTAX**

G. URSCHLER

IBM

LABORATORY VIENNA

N O T E

This document is not an official PL/I Language Specification. For information concerning the official interpretation the reader is referred to the PL/I Language Specifications, Form No. Y33-6003-1.

IBM LABORATORY VIENNA,
Austria

TRANSLATION OF PL/I
INTO ABSTRACT SYNTAX

by

G. URSCHLER

ABSTRACT

This document defines a function which maps a concrete PL/I program into an abstract program to be interpreted as described in the "Abstract Syntax and Interpretation of PL/I" (TR 25.098). Additionally, the concrete syntax, in an abstract representation, and the abstract syntax of PL/I are given.

Locator Terms for IBM
Subject Index

PL/I
Formal Definition
Syntax, concrete
Syntax, abstract
21 PROGRAMMING

TR 25.097

30 June 1969

30 June 1969

TRANSLATION OF PL/I INTO ABSTRACT SYNTAX

PREFACE

This document is an updated version of:

- /1/ ALBER, K., OLIVA, P.: Translation of PL/I into Abstract Text.
IBM Laboratory Vienna, Techn. Report TR 25.086, 28 June 1968.
- It is part of a series of documents (ULD Version III) presenting the formal definition of syntax and semantics of PL/I:
- /2/ FLECK, M.: Formal Definition of the PL/I Compile Time Facilities (ULD Version III).
IBM Laboratory Vienna, Techn. Report TR 25.095, 30 June 1969.
- /3/ URSCHLER, G.: Concrete Syntax of PL/I (ULD Version III).
IBM Laboratory Vienna, Techn. Report TR 25.096, 30 June 1969.
- /4/ URSCHLER, G.: Translation of PL/I into Abstract Text (ULD Version III).
IBM Laboratory Vienna, Techn. Report TR 25.097, 30 June 1969.
- /5/ WALK, K., ALBER, K., FLECK, M., GOLDMANN, H., LAUER, P., MOSER, E., OLIVA, P., STIGLEITNER, H., ZEISEL, G.: Abstract Syntax and Interpretation of PL/I (ULD Version III).
IBM Laboratory Vienna, Techn. Report TR 25.098, 30 April 1969.
- /6/ ALBER, K., GOLDMANN, H., LAUER, P., LUCAS, P., OLIVA, P., STIGLEITNER, H., WALK, K., ZEISEL, G.: Informal Introduction to the Abstract Syntax and Interpretation of PL/I (ULD Version III).
IBM Laboratory Vienna, Techn. Report TR 25.099, 30 June 1969.

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

- /7/ PL/I Definition Group of the Vienna Laboratory: Formal Definition of PL/I.
IBM Laboratory Vienna, Techn. Report TR 25.071, 30 December 1966.
- /8/ ALBER, K.: Syntactical Description of PL/I Text and its Translation into Abstract Normal Form.
IBM Laboratory Vienna, Techn. Report TR 25.074, 14 April 1967.

An outline of the method is given in:

- /9/ LUCAS, P., LAUER, P., STIGLEITNER, H.: Method and Notation for the Formal Definition of Programming Languages.
IBM Laboratory Vienna, Techn. Report TR 25.087, 28 June 1968.
- This document also contains the appropriate references to the relevant literature. The basic ideas and their application to PL/I have been made available through several workshops on the formal definition of PL/I, and presentations and publications inside and outside IBM. The method is demonstrated by application to an appropriately tailored subset of PL/I in:

- /10/ LUCAS, P., WALK, K.: On the Formal Description of PL/I.
To be published in Annual Review in Automatic Programming - Vol.6.
Pergamon Press, New York 1969.

The language defined in the present version is PL/I as specified in the PL/I Language Specifications, Form No. Y33-6003-1, with the addition of attention handling, input stream and string scanning, and file variables.

The present document will be made subject to validation by the PL/I Language Department, Hursley.

PRODUCTION

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

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

Coordination: P. Schwarzenberger, M. Stadler

Technical control: K.F. Koch, E. Moser, M. Stadler

Data transcription: Miss W. Schatzl, Mrs. H. Deim, and sub-contractors

System support: H. Chladek, G. Lehmann

30 June 1969

TRANSLATION OF PL/I INTO ABSTRACT SYNTAX

INTRODUCTION

This document describes the translation of a concrete PL/I program into an abstract program, i.e., into an object whose structure is described by the abstract syntax given in chapter 5 (which is identical with that given for proper programs in /5/) and which is to be interpreted as defined in /5/. In order to remain within the range of methods and concepts used throughout the formal definition (cf. chapter 1 of /5/), a concrete PL/I program is assumed as a list of character values, i.e., of abstract elementary objects representing uniquely the concrete PL/I characters.

The translation is performed in two steps by the two functions parse and translate: If txt is a concrete program, the corresponding abstract program progr is defined as

```
progr = translate•parse(txt).
```

The link between these two steps, namely the result of parse and the argument of translate, is a structured object t, called the abstract representation of the concrete program. Its structure is described by a set of predicate definitions, called the abstract representation of the concrete syntax. These predicate definitions, listed in chapter 3, are received by mechanical syntactic transcription of the production rules of the concrete syntax listed in /3/. Appendix I contains the transcription rule between the concrete syntax in /3/ and its abstract representation in chapter 3. In particular, appendix I contains the correspondence between the concrete PL/I characters and their abstract character values used in the formal definition.

An object t satisfying the abstract representation of the concrete syntax may be thought of as the parsing tree of a concrete program (for a more detailed description see the summary of chapter 2). Its elementary components are the character values which, listed in proper order, constitute the corresponding concrete programs. (Since there is freedom in the insertion of blanks and comments into a concrete program, one parsing tree corresponds to an infinite number of equivalent concrete programs). Chapter 2 defines a function, generate, which applied to t yields the set of concrete programs corresponding to t. This function may be considered as the formal definition of the directions for use of the concrete syntax to generate a concrete PL/I program, as given in /3/. The above mentioned function parse, defined in chapter 2 as the inverse of the function generate, transforms a character value list into its parsing tree t, provided it is a syntactically correct concrete program.

Chapter 4 defines the second step of the translation, namely the above mentioned function translate. It constructs an abstract program as described by the abstract syntax given in chapter 5. The main job is the recognition of all declarations in a concrete program and the testing, completing and structuring of their attributes. For the other components of a PL/I program, in particular the statements, the translation consists essentially of a one-to-one mapping from the parsing tree into the abstract program. This mapping constructs objects built up with mnemonically named selectors instead of selectors determined only by the ordering in the concrete program.

Chapter 1 gives some extensions to the notation and conventions defined in /5/. Appendix II is a detailed cross-reference list.

PL/I does not state whether a program, which contains "errors" in unexecuted parts, is erroneous and should be rejected by the formal definition. Obviously this question cannot be answered simply by "yes" or "no". On the one hand, the formal definition cannot translate any character value list successfully; at least a minimum of syntactic correctness is required. On the other hand, it seems impossible in many cases even to define what is an error in a part of a program without interpreting it. So, the formal definition has to draw a borderline between "statical" and "dynamical" errors. A program containing "statical errors" is rejected during translation independently from an interpretation. A program containing "dynamical errors" is rejected during interpretation only if the containing program part is really interpreted. Principally, this borderline has been drawn in the following way: The function parse rejects all concrete programs which contradict the concrete syntax. The function translate rejects all programs which would lead to abstract programs contradicting the abstract syntax, and additionally it tests that (redundant) information in a concrete program which is no more reflected in the abstract program. The recognition of all other "errors" is left to the interpreter, which consequently rejects them only if occurring in interpreted program parts.

30 June 1969

TRANSLATION OF PL/T INTO ABSTRACT SYNTAX

CONTENTS

1. NOTATION AND CONVENTIONS	1
1.1 Selector Relations	1
1.2 Some Generally Used Functions	1
1.3 Referencing and Abbreviations	2
2. GENERATION OF A CONCRETE PROGRAM	1
2.1 Special Syntactic Features of PL/T	4
2.1.1 Keyword abbreviations	4
2.1.2 Multiple closure of blocks and groups	5
2.1.3 Programs in the 48 character set	8
3. ABSTRACT REPRESENTATION OF CONCRETE SYNTAX	1
3.1 Declarations	2
3.1.1 Attributes	4
3.1.2 Formats	8
3.2 Statements	10
3.2.1 Block and groups	10
3.2.2 Flow of control statements	11
3.2.3 Storage manipulating statements	14
3.2.4 Condition and attention handling statements	15
3.2.5 Input and output statements	16
3.3 Expressions	20
3.4 Identifiers and Constants	22
3.5 Pictures	24
3.6 Implementation-defined Predicates	24
3.7 Predicates Defining Constant Character Lists	25
4. THE TRANSLATOR	1
4.1 Program, Procedure	2
4.1.1 Program	2
4.1.2 Auxiliary functions concerning block structure	3
4.1.3 Procedure body	4
4.2 Declarations	7
4.2.1 Construction of preliminary declarations	8
4.2.1.1 Explicit declaration contexts	11
4.2.1.2 Successor relation for structure declarations	11
4.2.1.3 Multiple declarations	15
4.2.1.4 Contextual declarations	16
4.2.1.5 Attributes specified for a declaration	17
4.2.1.6 Default attributes	18
4.2.2 Construction of declarations	22
4.2.2.1 Classification of declaration types	24
4.2.2.2 Scope, connectedness, density and storage class	26
4.2.2.3 The dimension attribute	28
4.2.2.4 Structure declarations	30
4.2.2.5 Classification of data attributes	31
4.2.2.6 Arithmetic data attributes	33
4.2.2.7 String data attributes	35
4.2.2.8 Picture data attributes	36
4.2.2.9 Area attribute	40
4.2.2.10 Label attribute	41
4.2.2.11 Entry declaration	41
4.2.2.12 File declaration	46
4.2.2.13 Defined and based declarations	47
4.2.2.14 Format label declaration	48
4.2.2.15 Generic declaration	50
4.2.2.16 Initial attribute and initial label	53
4.2.3 Parameter descriptors and allocate items	55

30 June 1969

TRANSLATION OF PL/I INTO ABSTRACT SYNTAX

4.3 Statements	59
4.3.1 Statement prefixes	62
4.3.2 Block and groups	64
4.3.3 Flow of control statements	65
4.3.4 Storage manipulating statements	68
4.3.5 Condition handling statements	72
4.3.6 Input and output statements	75
4.4 Expressions	81
4.4.1 References	82
4.4.2 Constants	87
4.5 Identifiers	89
5. ABSTRACT SYNTAX	1
5.1 Program, Procedure Body	1
5.2 Declarations	2
5.2.1 General	2
5.2.2 Aggregate attributes	3
5.2.2.1 Aggregates in variable declarations	3
5.2.2.2 Aggregates in parameter descriptors	5
5.2.2.3 Aggregates in generic parameter descriptors	6
5.2.2.4 Aggregates in allocate statements	7
5.2.3 Pictures	7
5.2.4 Formats	8
5.3 Statements	9
5.3.1 Block, group	10
5.3.2 Flow of control statements	10
5.3.3 Storage manipulating statements	11
5.3.4 Condition and attention handling statements	12
5.3.5 Input and output statements	14
5.4 Expressions	17
APPENDIX I: TRANSCRIPTION OF CONCRETE SYNTAX INTO ABSTRACT REPRESENTATION	1
1. The Meta Syntax	1
2. Transcription of Production Rules	1
2.1 Transcription of higher level production rules	2
2.2 Transcription of lower level production rules	3
APPENDIX II: CROSS-REFERENCE INDEX	1

30 June 1969

TRANSLATION OF PL/I INTO ABSTRACT SYNTAX

1. NOTATION AND CONVENTIONS

Throughout the present document the notation and conventions introduced in chapter 1 of /5/ are used without any special reference. (Of course, parts of it are irrelevant for the present document, in particular all those concerned with the concept of a sequential machine and instruction definitions). Any other function defined in /5/ and used in the present document is listed under an individual formula number with a reference to /5/.

The following additional notations are used in the present document.

1.1 SELECTOR RELATIONS

The following two relations between composite selectors are used.

$$(1) \quad p \Rightarrow q \equiv (\exists \sigma) (\sigma \neq I \& q = \sigma \circ p)$$

I.e., applied to any object x , $q(x)$ is a component of $p(x)$ ("q is a continuation of p").

$$(2) \quad p < q \equiv (\exists \sigma, n, m) (n < m \& ((s(n)) \circ \sigma = p \vee (s(n)) \circ \sigma \Rightarrow p) \& ((s(m)) \circ \sigma = q \vee (s(m)) \circ \sigma \Rightarrow q))$$

("p is left to q")

This relation introduces a partial ordering of selectors: The values of the function $s(n)$ and thereby all their continuations are ordered by the natural ordering of integer values.

1.2 SOME GENERALLY USED FUNCTIONS

By the values of the selector function $s(n)$ objects are formed which have a similar structure as the lists by the values of the selector function $\text{elem}(n)$. Analogously to the function length for lists, for these objects the function slength is defined:

$$(3) \quad \text{slength}(x) = \\ (\forall n) (\text{is-Q} \circ (s(n))(x)) \rightarrow 0 \\ T \rightarrow (\exists n) (\neg \text{is-Q} \circ (s(n))(x) \& (\forall m) (m > n \Rightarrow \text{is-Q} \circ (s(m))(x)))$$

Note: In principle, this function is defined for any object x : If there is no $s(n)$ -component it yields 0, else the maximum index of an existing $s(n)$ -component. Usually it is applied either to Q, yielding 0, or to objects of the form

$(\langle s(1) : \text{is-pred} \rangle, \dots)$

or $(\langle s-\text{del} : \text{is-pred}_0 \rangle, \langle s(1) : \text{is-pred} \rangle, \dots)$

where the class of is-pred does not include Q, i.e., to "gapless s-lists", possibly with one additional component $s\text{-del}$ ("list delimiter").

Similarly as for the function elem (cf. /5/) the following short notation is used for the function s :

(4) $s(n, x) =$
 $s(n)(x)$

Often it will be necessary to collect a set of selectors, defined implicitly by any property, into an ordered list according to the ordering relation defined in section 1.1. This is performed by the following function:

(5) $\text{collect}(\text{set}) =$
 $\text{is-}[](\text{set}) \rightarrow \langle \rangle$
 $T \rightarrow \langle \text{leftmost}(\text{set}) \rangle^* \text{collect}(\text{set} - \text{leftmost}(\text{set}))$

(6) $\text{leftmost}(\text{set}) =$
 $(\exists p) (p \in \text{set} \wedge \neg (\exists q) (q \in \text{set} \wedge q < p))$

1.3 REFERENCING AND ABBREVIATIONS

As in /5/, each definition formula is followed by a list of references of used functions with the same referencing notation, namely:

function-name chapter - page (formula-number).

Referenced are all those functions, which are not defined in the chapters on notation (chapters 1 of /5/ and of the present document) or the abstract representation of concrete syntax (chapter 3, predicate names starting with is-c-...). Appendix II lists all functions defined in chapters 2 through 5 with their definition and all occurrences.

The rules for specification of metavariables and of abbreviations are the same as in /5/.

In order to avoid accumulation of confusing parentheses in expressions like $(s(1)) \cdot (s(n)) \cdot p(t)$ the following abbreviations are used throughout the present document:

$s_n = s(n)$	for $\text{is-intg-val}(n)$
$\text{elem}_n = \text{elem}(n)$	for $\text{is-intg-val}(n)$.

Using this convention the above expression reads: $s_1 \cdot s_n \cdot p(t)$.

As a further convenience the abbreviation '?' is used with the following meaning:

If $g((\exists x) (f_1(x)), (\exists y) (f_2(y)), \dots)$ is a function call where all occurring \exists -expressions are specified, then a part of a conditional expression of the form

$(\exists x) (f_1(x)) \& (\exists y) (f_2(y)) \& \dots \rightarrow g((\exists x) (f_1(x)), (\exists y) (f_2(y)), \dots)$

may be abbreviated to

$\exists \rightarrow g((\exists x) (f_1(x)), (\exists y) (f_2(y)), \dots)$

In contrary to /5/ the ending 'T -- error' in conditional expressions is omitted throughout this document. Each conditional expression not ending with 'T -- ...' might be closed by 'T -- error'.

30 June 1969

TRANSLATION OF PL/I INTO ABSTRACT SYNTAX

2. GENERATION OF A CONCRETE PROGRAM

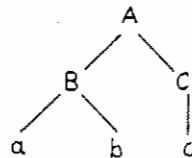
A concrete program text is a string of concrete PL/I characters. For the formal definition this string of concrete characters is represented by a list of abstract character values, i.e., of elementary objects satisfying the predicate is-char-val (cf. /5/). The correspondence between the individual concrete characters and these character values is given by the function ar-2 defined in appendix I.

To each concrete program corresponds an object t, called its abstract representation, which satisfies the predicate is-c-program. This predicate is-c-program is defined in chapter 3 by a set of predicate definitions, which is a one-to-one mapping of the set of production rules of the concrete syntax of PL/I as given in /3/. Thereby, the abstract representation t of a concrete program is an object, whose structuring represents the syntactical structuring of the concrete program and whose elementary components are the character values constituting the concrete program. One may think of t as the parsing tree of the concrete program, but with the following differences:

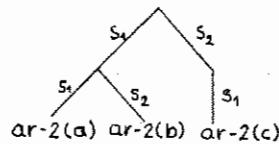
- (1) The terminal nodes are the abstract representations of the corresponding concrete character values.
- (2) Terminal nodes excepted, not the nodes but the edges leading to the nodes are named (by the selectors 's₁', 's₂', ... 'elem₁', 'elem₂', ... and 's-del'). The order of the edges has therefore no significance.

Example:

A parsing tree like



might get the form



The present chapter defines the correspondence between a concrete program (as a list of character values) and its abstract representation t . This is done by specifying a function parse , which maps a concrete program into its abstract representation t , and its inverse function generate .

The function generate maps t into a set of character lists in the following three steps:

- (1) The function generate-1 planes t into a list of those components which represent syntactical units that may not be interrupted by spaces (blanks or comments). This is done by planing the structure given by selectors of the form $s(n)$ or $s\text{-del}$, but not affecting the structure given by the selectors of the form $\text{elem}(n)$.
- (2) The function insert-space intersperses the list produced in the first step with spaces. It produces the (infinite) set of all lists resulting from this interspersing satisfying the condition that at least between all pairs of consecutive non-delimiters spaces are inserted.
- (3) The function generate-2 planes each individual element of the set produced in the second step into a final list of character values. This is done by planing also the structure given by selectors of the form $\text{elem}(n)$.

The abstract representation of the concrete syntax given in chapter 3 together with the function generate constitutes a formal definition of an algorithm to generate all concrete PL/I programs. So, they are equivalent to the production rules of the concrete syntax and the description how to use them, given in /3/. For completeness the transcription rule between both forms of the concrete syntax is given in appendix I of the present document.

The function parse is exactly defined as the unique inverse of the function generate . Special characteristics of parse are:

- (1) Metasyntactical components representing an unit different from not-var and not-const (cf. /3/ or appendix I) are not ignored in the parsing tree but get an own artificial node corresponding to a thought substitution rule. Omitted optional components (δ -components) get the terminal node Ω .
- (2) A production of the form $V ::= U^{***}$ (cf. /3/ or appendix I) is interpreted directly as a finite concatenation of U s without further V -structuring.
- (3) A text belonging to a production part of the form $\{ T \cdot U^{***} \}$ (cf. /3/ or appendix I) is parsed without the interposed T s according to a simple U^{***} and afterwards an edge named $s\text{-del}$ with the terminal node T is inserted into the parsing tree parallel to the edges corresponding to the U s.

The function parse produces that object t , which, by the function translate defined in chapter 4, is translated into an abstract program to be interpreted in /5/. So, the function parse is the first step of the formal definition of PL/I.

Metavariables

text	is-char-val-list	a concrete program
t	is-c-program	abstract representation of text
x		any component of t or a list of such components

```
(1) parse(text) =
    (bt) (text ∈ generate(t) & is-c-program(t))
```

30 June 1969.

TRANSLATION OF PL/I INTO ABSTRACT SYNTAX

Note: This function fails if the concrete syntax corresponding to the predicate is-c-program is ambiguous.

```
(2) generate(t) =
  (generate-2(x) | x ∈ insert-space•generate-1(t))

(3) generate-1(x) =
  is-Ω(x) → <>
  slength(x) = 1 → <x>
  T → generate-1•s1(x) ~ CONC (generate-1•s-del(x) ~ generate-1•sn(x))
    n=2
  slength(x)
  Note: The last case yields CONC generate-1•sn(x) for s-del(x) = Ω
        n=1
  generate-1 handles the objects corresponding to the higher level syntax.

(4) generate-2(x) =
  is-Ω(x) → <>
  is-char-val(x) → <x>
  T → CONC generate-2•elem(n,x)
    n=1
```

where:
 $lg_1 = (\bigwedge n) (\neg is-\Omega \cdot elem(n, x) \wedge (\forall m) (m > n \Rightarrow is-\Omega \cdot elem(m, x)))$

Note: This function is applicable for lists and "lists with gaps" as well. It handles the objects corresponding to the lower level syntax.

(5) is-char-val =

Note: cf. /5/.

(6) insert-space(x) =

insert-space-1(x) ∨ {<y>^z | is-c-space(y) & z ∈ insert-space-1(x)}

Note: This function inserts optional spaces.

(7) insert-space-1(x) =

is-<>(x) → {<>}

is-c-delimiter•head(x) ∨ is-c-delimiter•head•tail(x) →

{mk-list(head(x), y, z) | (is-c-space(y) ∨ is-Ω(y)) &
 z ∈ insert-space-1•tail(x)}

T → {mk-list(head(x), y, z) | is-c-space(y) & z ∈ insert-space-1•tail(x)}

Note: The last condition inserts mandatory spaces.

```
(8)    mk-list(a,b,list) =
      μ0 (<elem(1):a>,<elem(2):b>) ^ list
      for:-is-Ω(a),is-list(list),is-list•mk-list(a,b,list)

(9)    is-c-delimiter(x) =
      x ∈
      {EQ,PLUS,MINUS,ASTER,SLASH,LEFT-PAR,RIGHT-PAR,COMMA,POINT,SEMIC,Colon,
      AND,OR,NOT,GT,LT,<ASTER,ASTER>,<OR,OR>,<GT,EQ>,<LT,EQ>,<NOT,EQ>,<NOT,GT>,
      <NOT,LT>,<MINUS,GT>}

(10)   is-c-space =
      (<elem(1):is-BLANK ∨ is-c-comment>,...)

(11)   is-c-comment =
      (<elem(1):is-SLASH>,
       <elem(2):is-ASTER>,
       <elem(3):is-Ω ∨
       (<elem(1):is-SLASH ∨
        (<elem(1):is-Ω ∨
         (<elem(1):is-ASTER>,...)>,
         <elem(2):is-c-comment-symbol>)>,...)>,
       <elem(4):(<elem(1):is-ASTER>,...)>,
       <elem(5):is-SLASH>)

(12)   is-c-comment-symbol(ch) =
      is-char-val(ch) & ¬is-ASTER(ch) & ¬is-SLASH(ch)
```

2.1 SPECIAL SYNTACTIC FEATURES OF PL/I

This section describes the changes to be made in the general algorithm described above and in chapter 3 for the generation or parsing of a concrete program, if one wishes to include the following three special syntactic features of PL/I: Abbreviation of keywords, multiple closure of blocks and groups, and writing of programs in a restricted 48 character set.

2.1.1 KEYWORD ABBREVIATIONS

The following table lists all keywords of the concrete syntax, which may be abbreviated, and their abbreviations. To include this facility, one has to replace the predicates is-c-[name], where [name] is one of the concrete keywords listed in the table, throughout the formal definition by the corresponding predicates is-c-abbr-[name] defined by:

```
(13)   is-c-abbr-[name] =
      is-c-[name] ∨ is-c-[abbr-name]
```

30 June 1969

TRANSLATION OF PL/I INTO ABSTRACT SYNTAX

where: [name] and [abbr-name] are pairs of names in capital letters given by the following table.

Note: As an example, this scheme defines the predicate `is-c-abbr-ATTENTION` as:

```
is-c-abbr-ATTENTION =
    is-c-ATTENTION v is-c-ATTN.
```

[name]		[abbr-name]
ATTENTION		ATTN
AUTOMATIC		AUTO
BCOLUMN		BCOL
BEGINVOLUME		BOV
BINARY		BIN
BUFFERED		BUF
CHARACTER		CHAR
COLUMN		COL
COMPLEX		CPLX
CONNECTED		CONN
CONTROLLED		CTL
CONVERSION		CONV
DECIMAL		DEC
DECLARE		DCL
DEFINED		DEF
ENVIRONMENT		ENV
EXCLUSIVE		EXCL
EXTERNAL		EXT
FIXEDOVERFLOW		FOFL
INITIAL		INIT
INTERNAL		INT
IRREDUCIBLE		IRRED
NOCONVERSION		NOCONV
NOMIXEDOVERFLOW		NOFOFL
NOOVERFLOW		NOOFL
NOSTRINGRANGE		NOSTRG
NOSUBSCRIPTRANGE		NOSUBRG
NOUNDERFLOW		NOUFL
NOZERODIVIDE		NOZDIV
OVERFLOW		OFL
PICTURE		PIC
POINTER		PTR
POSITION		POS
PROCEDURE		PROC
REDUCIBLE		RED
SEQUENTIAL		SEQL
STRINGRANGE		STRG
STRINGSIZE		STRZ
SUBSCRIPTRANGE		SUBRG
UNALIGNED		UNAL
SUBSCRIPTRANGE		SUBRG
UNBUFFERED		UNBUF
UNDEFINEDFILE		UNDF
UNDERFLOW		UFL
VARYING		VAR
ZERODIVIDE		ZDIV

2.1.2 MULTIPLE CLOSURE OF BLOCKS AND GROUPS

It is possible to close more than one begin block, procedure or group, shortly called "compound" here, by one common end clause. To be syntactically unique, this end clause has to contain one of the identifiers occurring as labels

TRANSLATION OF PL/I INTO ABSTRACT SYNTAX

30 June 1969

of the outermost compound to be closed. In this case it is not allowed, that one of the inner compounds to be closed by the common end clause has the same identifier within the labellist preceding the keywords 'PROCEDURE', 'BEGIN' or 'DO'. If the common end clause has prefixes or labels, they are handled as prefixes or labels of the end clause in the outermost compound.

This syntactic facility is described in the following by giving an equivalence relation between an abstract representation t of a concrete program, i.e., an object t satisfying the predicate is-c-program , and an object u , which is received from t by performing suitable changes: deleting simple end clauses (i.e., such without prefixes and labels) and inserting identifiers into end clauses.

To include this facility, one has to replace the function generate throughout the formal definition by the following function generate-equiv:

(14) generate-equiv(t) =
$$\text{generate}(t) \cup \bigcup_{u \in \text{set}_1} \text{generate}(u)$$

where:

$$\text{set}_1 = \{x \mid \text{is-equiv-c-program}(t, x)\}$$
(15) is-equiv-c-program(t, u) =
$$\text{is-c-program}(t) \wedge (\exists t-1, p-1, c-id, p) ((t = t-1 \vee \text{is-equiv-c-program}(t, t-1)) \wedge \text{is-equiv-compound}(p-1(t-1), p-1(u), c-id, p) \wedge \delta(t-1; p-1) = \delta(u; p-1))$$

Note: This equivalence relation expresses that u is received from the "complete" t , satisfying $\text{is-c-program}(t)$, by successively replacing "complete" compounds by equivalent "modified" ones. If $t-1$ has already been received this way, u is received from $t-1$ by only replacing the complete compound $p-1(t-1)$ by the equivalent modified $p-1(u)$.

(16) is-equiv-compound($x, y, c-id, p$) =
$$p = I \rightarrow$$

$$\text{is-c-compound}(x) \wedge \text{is-labeled}(x, c-id) \wedge y = \mu(x; \langle s_4, \cdot(\text{end-cl-p}(x)):c-id \rangle, \langle s_5, \cdot(\text{end-cl-p}(x)):SEMIC \rangle)$$

$$T \rightarrow$$

$$(\exists y-1, p-1) (\text{is-equiv-compound}(x, y-1, c-id, p-1) \wedge p = (\text{last-sent-o} \cdot p-1(x)) \cdot p-1 \wedge \text{is-c-compound-p}(x) \wedge \text{is-simple-end}(\text{end-cl-p} \cdot p(x)) \cdot p(x) \wedge \neg \text{is-labeled}(p(x), c-id) \wedge y = \delta(y-1; (\text{end-cl-p} \cdot p(x)) \cdot p))$$

Note: This relation defines the equivalence between a complete compound x , satisfying $\text{is-c-compound}(x)$, and the modified y . Here, $c-id$ is the concrete identifier inserted into the end clause of x and p is the selector which, applied to x , points to the innermost compound to be closed by the common end clause. The modified y is received from x successively: the first step ($p=I$) inserts only $c-id$ into the end clause of x ; each of the following steps deletes one simple end clause if allowed, going from the outer to the inner compound.

(17) is-c-compound(x) =
$$\text{is-c-begin-block-s}_3(x) \vee \text{is-c-procedure}(x) \vee \text{is-c-group-s}_3(x)$$

30 June 1969.

TRANSLATION OF PL/I INTO ABSTRACT SYNTAX

(18) is-labeled(x,c-id) =
 is-c-identifier(c-id) & ($\exists n$) (c-id = $s_1 \cdot s_n \cdot s_2(x)$)

for:is-c-compound(x),is-c-identifier(c-id)

(19) end-cl-p(x) =
 $s_2 \cdot (\text{sent-list-p}(x))$

for:is-c-compound(x)

(20) last-sent-p(x) =
 $(\text{last-sent-p-1} \cdot p_1(x)) \cdot p_1$

where:

$p_1 = (s(lg_1)) \cdot s_1 \cdot (\text{sent-list-p}(x))$
 $lg_1 = \text{slength} \cdot s_1 \cdot (\text{sent-list-p}(x))(x)$

for:is-c-compound(x)

(21) last-sent-p-1(x) =
 $\neg \text{is-c-if-statement}(x) \rightarrow I$
 $T \rightarrow (\text{last-sent-p-1} \cdot p_1(x)) \cdot p_1$

where:

$p_1 = (\text{is-Q} \cdot s_4 \cdot s_3(x) \rightarrow s_2 \cdot s_3,$
 $T \rightarrow s_4 \cdot s_3)$

for:is-c-sentence(x)

(22) sent-list-p(x) =
 $\text{is-c-begin-block} \cdot s_3(x) \rightarrow s_4 \cdot s_3$
 $\text{is-c-procedure}(x) \rightarrow s_7$
 $\text{is-c-simple-group} \cdot s_3(x) \rightarrow s_3 \cdot s_3$
 $\text{is-c-iterated-group} \cdot s_3(x) \rightarrow s_4 \cdot s_3$

for:is-c-compound(x)

(23) is-simple-end =
 $(\langle s_3 : \text{is-c-END} \rangle,$
 $\langle s_4 : \text{is-SEMIC} \rangle)$

2.1.3 PROGRAMS IN THE 48 CHARACTER SET

It is possible to write PL/I programs in a restricted 48 character set and to replace the characters not occurring in this character set by corresponding sequences of other characters. In this case the language has 12 "reserved words" which may not be used as identifiers. Additionally, for syntactic reasons of unambiguity a point and a following colon (replaced by two points) has to be separated by a space.

If a program which is written in this 48 character set shall be handled by the formal definition, the function generate has to be replaced throughout the formal definition by the following function generate-48.

```
(24) generate-48(t) =
      ~ (Ep) (is-c-identifier•o(t) &
              (is-c-NOT•generate-2•o(t) v is-c-AND•generate-2•o(t) v
               is-c-OR•generate-2•o(t) v is-c-GT•generate-2•o(t) v
               is-c-LT•generate-2•o(t) v is-c-GE•generate-2•o(t) v
               is-c-ID•generate-2•o(t) v is-c-NG•generate-2•o(t) v
               is-c-NL•generate-2•o(t) v is-c-NE•generate-2•o(t) v
               is-c-CAT•generate-2•o(t) v is-c-PT•generate-2•o(t))) ++
      {generate-2(x) | x ε insert-space-48•replace-48•generate-1(t) &
      ~ (Em) (elem(n,generate-2(x)) ε
              {COMM-AT,NUMBER-SIGN,BREAK,SEMIC,COLON,AND,OR,NOT,GT,LT,QUEST,PERC}))}

(25) replace-48(x) =
      length(x)
      LIST  replace-48-1•elem(n,x)
      n=1

(26) replace-48-1(x) =
      is-SEMIC(x) --> <COMMA,POINT>
      is-COLON(x) --> <POINT,POINT>
      is-NOT(x) --> <N-CHAR,O-CHAR,T-CHAR>
      is-AND(x) --> <A-CHAR,N-CHAR,D-CHAR>
      is-OR(x) --> <O-CHAR,R-CHAR>
      is-GT(x) --> <G-CHAR,T-CHAR>
      is-LT(x) --> <L-CHAR,T-CHAR>
      x = <GT,EQ> --> <G-CHAR,E-CHAR>
      x = <LT,EQ> --> <L-CHAR,E-CHAR>
      x = <NOT,GT> --> <N-CHAR,G-CHAR>
      x = <NOT,LT> --> <N-CHAR,L-CHAR>
      x = <NOT,EQ> --> <N-CHAR,E-CHAR>
      x = <OR,OR> --> <C-CHAR,A-CHAR,T-CHAR>
      x = <MTNUS,GT> --> <P-CHAR,T-CHAR>
      T --> x
```

30 June 1969

TRANSLATION OF PL/I INTO ABSTRACT SYNTAX

```
(27)    insert-space-48(x) =
        insert-space-48-1(x) u [y]^z | is-c-space(y) & z ∈ insert-space-48-1(x)

(28)    insert-space-48-1(x) =
        is-<>(x) -- [<>]
        is-POINT•last•generate-2•head(x) & head•tail(x) = <POINT,POINT> --
        {mk-list(head(x),y,z) | is-c-space(y) & z ∈ insert-space-48-1•tail(x)}
        is-c-delimiter-48•head(x) v is-c-delimiter-48•head•tail(x) --
        {mk-list(head(x),y,z) | (is-Ω(y) v is-c-space(y)) &
         z ∈ insert-space-48-1•tail(x)}
        T -- {mk-list(head(x),y,z) | is-c-space(y) & z ∈ insert-space-48-1•tail(x)}

(29)    is-c-delimiter-48(x) =
        is-c-delimiter(x) v x = <POINT,POINT> v x = <COMMA,POINT>
```


30 June 1969

TRANSLATION OF PL/I INTO ABSTRACT SYNTAX

3. ABSTRACT REPRESENTATION OF CONCRETE SYNTAX

The abstract representation of the concrete syntax is a set of predicate definitions which together define the predicate is-c-program. Objects satisfying this predicate are valid arguments for the function translate defined in chapter 4, and for the function generate defined in chapter 2.

The abstract representation of the concrete syntax is closely related with the production rules of the concrete syntax as defined in chapter 3 of /3/. See the summary of chapter 2 and appendix I.

- (1) is-c-program =

$$(<s(1):is-c-procedure>, \dots)$$
- (2) is-c-procedure =

$$(<s(1):is-0 \vee is-c-prefixlist>,
 <s(2):is-c-labellist>,
 <s(3):is-c-PROCEDURE>,
 <s(4):is-0 \vee is-c-parameterlist>,
 <s(5):is-0 \vee is-c-procedure-optionslist>,
 <s(6):is-SEMIC>,
 <s(7):is-c-sentencelist>)$$
- (3) is-c-parameterlist =

$$(<s(1):is-LEFT-PAR>,
 <s(2):(<s-del:is-COMMA>,
 <s(1):is-c-identifier>, \dots)>,
 <s(3):is-RIGHT-PAR>)$$
- (4) is-c-procedure-optionslist =

$$(<s(1):is-c-options-attribute \vee is-c-returns-attribute \vee is-c-ORDER \vee
 is-c-REORDER \vee is-c-RECURSIVE>, \dots)$$
- (5) is-c-sentencelist =

$$(<s(1):is-0 \vee
 (<s(1):is-c-sentence>, \dots)>,
 <s(2):is-c-end-clause>)$$
- (6) is-c-end-clause =

$$(<s(1):is-0 \vee is-c-prefixlist>,
 <s(2):is-0 \vee is-c-labellist>,
 <s(3):is-c-END>,
 <s(4):is-SEMIC>)$$
- (7) is-c-sentence =

$$\text{is-c-procedure} \vee \text{is-c-entry} \vee \text{is-c-declaration-sentence} \vee
 \text{is-c-format-sentence} \vee \text{is-c-statement}$$

(8) is-c-entry =
 (<s(1):is-c-labellist>,
 <s(2):is-c-ENTRY>,
 <s(3):is-Ω v is-c-parameterlist>,
 <s(4):is-Ω v is-c-returns-attribute>,
 <s(5):is-SEMIC>)

3.1 DECLARATIONS

(9) is-c-declaration-sentence =
 (<s(1):is-Ω v is-c-labellist>,
 <s(2):is-c-declare-sentence v is-c-default-sentence>)

(10) is-c-declare-sentence =
 (<s(1):is-c-DECLARE>,
 <s(2):is-c-declarationlist>,
 <s(3):is-SEMIC>)

(11) is-c-declarationlist =
 (<s-del:is-COMMA>,
 <s(1):is-c-declaration>,...)

(12) is-c-declaration =
 (<s(1):is-Ω v is-c-integer>,
 <s(2):is-c-identifier v
 (<s(1):is-LEFT-PAR>,
 <s(2):is-c-declarationlist>,
 <s(3):is-RIGHT-PAR>),
 <s(3):is-Ω v is-c-dimension-attribute>,
 <s(4):is-Ω v
 (<s(1):is-c-attribute>,...)>)

(13) is-c-default-sentence =
 is-c-default-option-1 v is-c-default-option-2

(14) is-c-default-option-1 =
 (<s(1):is-c-DEFAULT>,
 <s(2):is-c-ALL>,
 <s(3):is-Ω v is-c-attribute-spec>,
 <s(4):is-SEMIC>)

(15) is-c-default-option-2 =
 (<s(1):is-c-DEFAULT>,
 <s(2):(s-del:is-COMMA>,
 <s(1):is-c-default-spec>,...)>,
 <s(3):is-SEMIC>)

30 June 1969

TRANSLATION OF PL/I INTO ABSTRACT SYNTAX

- (16) is-c-default-spec =
 is-c-simple-default-spec \vee is-c-factored-default-spec
- (17) is-c-simple-default-spec =
 ($s(1)$:is-c-range-spec,
 $s(2)$:is- \emptyset \vee is-c-attribute-spec>)
- (18) is-c-range-spec =
 is-c-identifier-range-spec \vee is-c-DESCRIPTORS
- (19) is-c-identifier-range-spec =
 ($s(1)$:is-c-RANGE,
 $s(2)$:is-LEFT-PAR,
 $s(3)$:(s -del:is-COMMA>,
 $s(1)$:is-c-identifier \vee
 ($s(1)$:is-c-letter,
 $s(2)$:is-COLON,
 $s(3)$:is-c-letter>)>,...) \vee is-ASTER,
 $s(4)$:is-RIGHT-PAR>)
- (20) is-c-attribute-spec =
 ($s(1)$:is- \emptyset \vee is-c-dimension-attribute),
 $s(2)$:($s(1)$:is-c-attribute \vee is-c-value-clause>,...)>) \vee is-c-SYSTEM
- (21) is-c-value-clause =
 ($s(1)$:is-c-VALUE,
 $s(2)$:is-LEFT-PAR,
 $s(3)$:(s -del:is-COMMA>,
 $s(1)$:is-c-value-spec>,...)>,
 $s(4)$:is-RIGHT-PAR)
- (22) is-c-factored-default-spec =
 ($s(1)$:is-LEFT-PAR,
 $s(2)$:(s -del:is-COMMA>,
 $s(1)$:is-c-default-spec>,...)>,
 $s(3)$:is-RIGHT-PAR,
 $s(4)$:is- \emptyset \vee is-c-attribute-spec)
- (23) is-c-value-spec =
 is-c-precision-spec \vee is-c-string-attribute \vee is-c-area-attribute
- (24) is-c-precision-spec =
 ($s(1)$:is-c-arithmetic-attribute>,...) \vee
 ($s(1)$:is-LEFT-PAR,
 $s(2)$:(s -del:is-COMMA>,
 $s(1)$:is-c-arithmetic-attribute>,...)>,
 $s(3)$:is-RIGHT-PAR,
 $s(4)$:is-c-arithmetic-attribute)

3.1.1 ATTRIBUTES

```

(25)  is-c-options-attribute =
      (<s(1):is-c-OPTIONS>,
       <s(2):is-LEFT-PAR>,
       <s(3):(<s-del:is-COMMA>,
              <s(1):is-c-external-option>, ...)>,
       <s(4):is-RIGHT-PAR>)

(26)  is-c-returns-attribute =
      (<s(1):is-c-RETURNS>,
       <s(2):is-LEFT-PAR>,
       <s(3):(<s(1):is-c-data-attribute ∨ is-c-entry-name-attribute ∨
              is-c-FILE>, ...)>,
       <s(4):is-RIGHT-PAR>)

(27)  is-c-dimension-attribute =
      (<s(1):is-LEFT-PAR>,
       <s(2):(<s-del:is-COMMA>,
              <s(1):is-c-bound-pair>, ...)>,
       <s(3):is-RIGHT-PAR>)

(28)  is-c-bound-pair =
      (<s(1):is-Ω ∨
       <s(1):is-c-refer-expression>,
       <s(2):is-COLON>),
       <s(2):is-c-refer-expression>) ∨ is-ASTER

(29)  is-c-refer-expression =
      (<s(1):is-c-expression>,
       <s(2):is-Ω ∨
       (<s(1):is-c-REFER>,
        <s(2):is-LEFT-PAR>,
        <s(3):is-c-unsubscripted-reference>,
        <s(4):is-RIGHT-PAR>))

(30)  is-c-attribute =
      is-c-data-attribute ∨ is-c-non-data-attribute ∨ is-c-entry-name-attribute ∨
      is-c-file-name-attribute ∨ is-c-scope-attribute ∨ is-c-like-attribute

(31)  is-c-data-attribute =
      is-c-arithmetic-attribute ∨ is-c-string-attribute ∨ is-c-VARYING ∨
      is-c-picture-attribute ∨ is-c-area-attribute ∨ is-c-label-attribute ∨
      is-c-POINTER ∨ is-c-offset-attribute ∨ is-c-TASK ∨ is-c-EVENT ∨
      is-c-storage-class-attribute ∨ is-c-defined-attribute ∨ is-c-based-attribute ∨
      is-c-UNALIGNED ∨ is-c-ALIGNED ∨ is-c-SECONDARY ∨ is-c-CONNECTED ∨
      is-c-VARIABLE ∨ is-c-initial-attribute

```

30 June 1969

TRANSLATION OF PL/I INTO ABSTRACT SYNTAX

- (32) is-c-arithmetic-attribute =
 (<s(1):is-c-REAL v is-c-COMPLEX v is-c-DECIMAL v is-c-BINARY v is-c-FLOAT v
 is-c-FIXED>,
 <s(2):is-Ω v
 (<s(1):is-LEFT-PAR>,
 <s(2):is-c-integer>,
 <s(3):is-Ω v
 (<s(1):is-COMMA>,
 <s(2):is-c-signed-integer>),
 <s(4):is-RIGHT-PAR>))
- (33) is-c-signed-integer =
 (<s(1):is-Ω v is-PLUS v is-MINUS>,
 <s(2):is-c-integer>)
- (34) is-c-string-attribute =
 (<s(1):is-c-BIT v is-c-CHARACTER>,
 <s(2):is-Ω v
 (<s(1):is-LEFT-PAR>,
 <s(2):is-c-refer-expression v is-ASTER>,
 <s(3):is-RIGHT-PAR>))
- (35) is-c-picture-attribute =
 (<s(1):is-c-PICTURE>,
 <s(2):is-Ω v is-c-picture-specification>)
- (36) is-c-area-attribute =
 (<s(1):is-c-AREA>,
 <s(2):is-Ω v
 (<s(1):is-LEFT-PAR>,
 <s(2):is-c-refer-expression v is-ASTER>,
 <s(3):is-RIGHT-PAR>))
- (37) is-c-label-attribute =
 (<s(1):is-c-LABEL>,
 <s(2):is-Ω v
 (<s(1):is-LEFT-PAR>,
 <s(2):(<s-del:is-COMMA>,
 <s(1):is-c-identifier>,...)>,
 <s(3):is-RIGHT-PAR>))
- (38) is-c-offset-attribute =
 (<s(1):is-c-OFFSET>,
 <s(2):is-Ω v
 (<s(1):is-LEFT-PAR>,
 <s(2):is-c-reference>,
 <s(3):is-RIGHT-PAR>))
- (39) is-c-storage-class-attribute =
 is-c-AUTOMATIC v is-c-STATIC v is-c-CONTROLLED

```

(40)  is-c-defined-attribute =
      (<s(1):is-c-DEFINED>,
       <s(2):is-c-basic-reference>) ∨
      (<s(1):is-c-POSITION>,
       <s(2):is-LEFT-PAR>,
       <s(3):is-c-expression>,
       <s(4):is-RIGHT-PAR>)

(41)  is-c-based-attribute =
      (<s(1):is-c-BASED>,
       <s(2):is-Ω ∨
       (<s(1):is-LEFT-PAR>,
        <s(2):is-c-reference>,
        <s(3):is-RIGHT-PAR>))

(42)  is-c-initial-attribute =
      (<s(1):is-c-INITIAL>,
       <s(2):is-c-initial-call ∨ is-c-initial-itemlist>)

(43)  is-c-initial-call =
      (<s(1):is-c-CALL>,
       <s(2):is-c-reference>)

(44)  is-c-initial-itemlist =
      (<s(1):is-LEFT-PAR>,
       <s(2):(<s-del:is-COMMA>,
              <s(1):is-c-initial-item>, ...),
       <s(3):is-RIGHT-PAR>)

(45)  is-c-initial-item =
      is-c-initial-iteration ∨ is-c-initial-constant ∨ is-c-simple-string-constant ∨
      is-c-reference ∨
      (<s(1):is-LEFT-PAR>,
       <s(2):is-c-expression>,
       <s(3):is-RIGHT-PAR>) ∨ is-ASTER

(46)  is-c-initial-iteration =
      (<s(1):is-LEFT-PAR>,
       <s(2):is-c-expression>,
       <s(3):is-RIGHT-PAR>,
       <s(4):is-c-initial-constant ∨ is-c-initial-itemlist ∨ is-c-reference>)

(47)  is-c-initial-constant =
      is-c-replicated-string-constant ∨ is-c-arithmetic-init-constant ∨
      is-c-sterling-constant

```

30 June 1969

TRANSLATION OF PL/I INTO ABSTRACT SYNTAX

```

(48)  is-c-arithmetic-init-constant =
      (<s(1):is-0 v is-PLUS v is-MINUS>,
       <s(2):is-c-real-constant>,
       <s(3):is-0 v
         (<s(1):is-PLUS v is-MINUS>,
          <s(2):is-c-imaginary-constant>) ) v
      (<s(1):is-0 v is-PLUS v is-MINUS>,
       <s(2):is-c-imaginary-constant>)

(49)  is-c-non-data-attribute =
      is-c-BUILTIN v is-c-generic-attribute v is-c-attention-attribute

(50)  is-c-entry-name-attribute =
      (<s(1):is-c-ENTRY>,
       <s(2):is-0 v
         (<s(1):is-LEFT-PAR>,
          <s(2):is-c-descriptorlist>,
          <s(3):is-RIGHT-PAR>) ) v is-c-returns-attribute v is-c-REDUCIBLE v
      is-c-IRREDUCIBLE

(51)  is-c-descriptorlist =
      (<s(1):is-c-descriptor>,
       <s(2):is-0 v
         (<s(1):is-COMMA>,
          <s(2):is-c-descriptorlist>) )

(52)  is-c-descriptor =
      (<s(1):is-0 v is-c-integer>,
       <s(2):is-0 v is-c-dimension-attribute>,
       <s(3):is-0 v
         (<s(1):is-c-attribute>,...)> ) v is-ASTER

(53)  is-c-file-name-attribute =
      is-c-FILE v is-c-file-attribute v
      (<s(1):is-c-ENVIRONMENT>,
       <s(2):is-LEFT-PAR>,
       <s(3):is-c-env-option>,
       <s(4):is-RIGHT-PAR>)

(54)  is-c-file-attribute =
      is-c-BITSTREAM v is-c-STREAM v is-c-RECORD v is-c-INPUT v is-c-OUTPUT v
      is-c-UPDATE v is-c-SEQUENTIAL v is-c-DIRECT v is-c-BUFFERED v is-c-UNBUFFERED v
      is-c-KEYED v is-c-PRINT v is-c-BACKWARDS v is-c-EXCLUSIVE v is-c-TRANSIENT

(55)  is-c-generic-attribute =
      (<s(1):is-c-GENERIC>,
       <s(2):is-LEFT-PAR>,
       <s(3):(<s-del:is-COMMA>,
              <s(1):is-c-generic-element>,...)>,
       <s(4):is-RIGHT-PAR>)

```

```
(56)  is-c-generic-element =
      (<s(1):is-c-reference>,
       <s(2):is-c-WHEN>,
       <s(3):is-LEFT-PAR>,
       <s(4):is-c-descriptorlist>,
       <s(5):is-RIGHT-PAR>)

(57)  is-c-scope-attribute =
      is-c-INTERNAL v is-c-EXTERNAL

(58)  is-c-like-attribute =
      (<s(1):is-c-LIKE>,
       <s(2):is-c-unsubscripted-reference>)

(59)  is-c-attention-attribute =
      (<s(1):is-c-ATTENTION>,
       <s(2):is-c-ENVIRONMENT>,
       <s(3):is-LEFT-PAR>,
       <s(4):is-c-env-option>,
       <s(5):is-RIGHT-PAR>)
```

3.1.2 FORMATS

```
(60)  is-c-format-sentence =
      (<s(1):is-@ v is-c-prefixlist>,
       <s(2):is-c-labellist>,
       <s(3):is-c-FORMAT>,
       <s(4):is-c-formatlist>,
       <s(5):is-SEMIC>)

(61)  is-c-formatlist =
      (<s(1):is-LEFT-PAR>,
       <s(2):(<s-del:is-COMMA>,
              <s(1):is-c-format>, ...),
       <s(3):is-RIGHT-PAR>)

(62)  is-c-format =
      is-c-format-iteration v is-c-format-item

(63)  is-c-format-iteration =
      (<s(1):is-c-integer v
       (<s(1):is-LEFT-PAR>,
        <s(2):is-c-expression>,
        <s(3):is-RIGHT-PAR>),
       <s(2):is-c-format-item v is-c-formatlist>)

(64)  is-c-format-item =
      is-c-data-format v is-c-control-format v is-c-remote-format
```

30 June 1969.

TRANSLATION OF PL/I INTO ABSTRACT SYNTAX

```

(65)  is-c-data-format =
      is-c-real-format v is-c-complex-format v is-c-string-format v
      is-c-picture-format

(66)  is-c-real-format =
      (<s(1):is-c-E v is-c-F>,
       <s(2):is-LEFT-PAR>,
       <s(3):is-c-expression>,
       <s(4):is-Q v
         (<s(1):is-COMMA>,
          <s(2):is-c-expression>,
          <s(3):is-Q v
            (<s(1):is-COMMA>,
             <s(2):is-c-expression>) >),
       <s(5):is-RIGHT-PAR>)

(67)  is-c-complex-format =
      (<s(1):is-c-C>,
       <s(2):is-LEFT-PAR>,
       <s(3):is-c-real-format v is-c-picture-format>,
       <s(4):is-Q v
         (<s(1):is-COMMA>,
          <s(2):is-c-real-format>) v
         (<s(1):is-COMMA>,
          <s(2):is-c-picture-format>),
       <s(5):is-RIGHT-PAR>)

(68)  is-c-string-format =
      (<s(1):is-c-A v is-c-B v is-c-BB>,
       <s(2):is-Q v
         (<s(1):is-LEFT-PAR>,
          <s(2):is-c-expression>,
          <s(3):is-RIGHT-PAR>) >)

(69)  is-c-picture-format =
      (<s(1):is-c-BP v is-c-P>,
       <s(2):is-c-picture-specification>)

(70)  is-c-control-format =
      (<s(1):is-c-BCOLUMN v is-c-BX v is-c-COLUMN v is-c-LINE v is-c-PAGE v
       is-c-SKIP v is-c-X>,
       <s(2):is-Q v
         (<s(1):is-LEFT-PAR>,
          <s(2):is-c-expression>,
          <s(3):is-RIGHT-PAR>) >)

(71)  is-c-remote-format =
      (<s(1):is-c-R>,
       <s(2):is-LEFT-PAR>,
       <s(3):is-c-reference>,
       <s(4):is-RIGHT-PAR>)

```

3.2 STATEMENTS

```
(72)  is-c-statement =
      (<s(1):is-0 v is-c-prefixlist>,
       <s(2):is-0 v is-c-labellist>,
       <s(3):is-c-if-statement v is-c-unconditional-statement>)

(73)  is-c-prefixlist =
      (<s(1):(s(1):is-LEFT-PAR>,
              <s(2):(<s-del:is-COMMA>,
                     <s(1):is-c-prefix-element>,...)>,
              <s(3):is-RIGHT-PAR>,
              <s(4):is-COLON>),...)

(74)  is-c-prefix-element =
      is-c-prefix v is-c-no-prefix v is-c-check-condition v is-c-no-check-condition

(75)  is-c-prefix =
      is-c-CONVERSION v is-c-FIXEDOVERFLOW v is-c-OVERFLOW v is-c-STRINGRANGE v
      is-c-STRINGSIZE v is-c-SUBSCRIPTRANGE v is-c-UNDERFLOW v is-c-ZERODIVIDE

(76)  is-c-no-prefix =
      is-c-NOCONVERSION v is-c-NOPROCESSOR v is-c-NOOVERFLOW v is-c-NOSIZE v
      is-c-NOSTRINGSIZE v is-c-NOSTRINGRANGE v is-c-NOSUBSCRIPTRANGE v
      is-c-NOUNDERFLOW v is-c-NOZERODIVIDE

(77)  is-c-labellist =
      (<s(1):(<s(1):is-c-basic-reference>,
              <s(2):is-COLON>),...)

(78)  is-c-unconditional-statement =
      is-c-begin-block v is-c-group v is-c-goto-statement v is-c-call-statement v
      is-c-incorporate-statement v is-c-fetch-statement v is-c-release-statement v
      is-c-return-statement v is-c-wait-statement v is-c-delay-statement v
      is-c-exit-statement v is-c-stop-statement v is-c-assignment-statement v
      is-c-allocate-statement v is-c-free-statement v is-c-on-statement v
      is-c-revert-statement v is-c-signal-statement v is-c-enable-statement v
      is-c-disable-statement v is-c-access-statement v is-c-open-statement v
      is-c-close-statement v is-c-stream-io-statement v is-c-record-io-statement v
      is-c-display-statement v is-c-null-statement

(79)  is-c-null-statement =
      is-SEMIC
```

3.2.1 BLOCK AND GROUPS

30 June 1969

TRANSLATION OF PL/I INTO ABSTRACT SYNTAX

```

(80)  is-c-begin-block =
      (<s(1):is-c-BEGIN>,
       <s(2):is-0 v
         (<s(1):is-c-options-attribute v is-c-ORDER v is-c-REORDER>,...)>,
       <s(3):is-SEMIC>,
       <s(4):is-c-sentencelist>)

(81)  is-c-group =
      is-c-simple-group v is-c-iterated-group

(82)  is-c-simple-group =
      (<s(1):is-c-DO>,
       <s(2):is-SEMIC>,
       <s(3):is-c-sentencelist>)

(83)  is-c-iterated-group =
      (<s(1):is-c-DO>,
       <s(2):is-c-do-specification v
         (<s(1):is-c-WHILE>,
          <s(2):is-LEFT-PAR>,
          <s(3):is-c-expression>,
          <s(4):is-RIGHT-PAR>),
       <s(3):is-SEMIC>,
       <s(4):is-c-sentencelist>)

(84)  is-c-do-specification =
      (<s(1):is-c-reference>,
       <s(2):is-EQ>,
       <s(3):(<s-del:is-COMMA>,
              <s(1):is-c-specification>,...)>)

(85)  is-c-specification =
      (<s(1):is-c-expression>,
       <s(2):is-0 v
         (<s(1):is-c-BY>,
          <s(2):is-c-expression>,
          <s(3):is-0 v
            (<s(1):is-c-TO>,
             <s(2):is-c-expression>)>) v
         (<s(1):is-c-TO>,
          <s(2):is-c-expression>,
          <s(3):is-0 v
            (<s(1):is-c-BY>,
             <s(2):is-c-expression>)>),
       <s(3):is-0 v
         (<s(1):is-c-WHILE>,
          <s(2):is-LEFT-PAR>,
          <s(3):is-c-expression>,
          <s(4):is-RIGHT-PAR>))

```

3.2.2 FLOW OF CONTROL STATEMENTS

TRANSLATION OF PL/I INTO ABSTRACT SYNTAX

30 June 1969

```

(86)    is-c-if-statement =
        (<s(1):is-c-if-clause>,
         <s(2):is-c-statement>) ∨
        (<s(1):is-c-if-clause>,
         <s(2):is-c-balanced-statement>,
         <s(3):is-c-ELSE>,
         <s(4):is-c-statement>)

(87)    is-c-if-clause =
        (<s(1):is-c-IF>,
         <s(2):is-c-expression>,
         <s(3):is-c-THEN>)

(88)    is-c-balanced-statement =
        (<s(1):is-Ω ∨ is-c-prefixlist>,
         <s(2):is-Ω ∨ is-c-labellist>,
         <s(3):(<s(1):is-c-if-clause>,
                 <s(2):is-c-balanced-statement>,
                 <s(3):is-c-ELSE>,
                 <s(4):is-c-balanced-statement>) ∨ is-c-unconditional-statement>)

(89)    is-c-goto-statement =
        (<s(1):is-c-GOTO ∨
         (<s(1):is-c-GO>,
          <s(2):is-c-TO>)>,
         <s(2):is-c-reference>,
         <s(3):is-SEMIC>)

(90)    is-c-call-statement =
        (<s(1):is-c-CALL>,
         <s(2):is-c-reference>,
         <s(3):is-Ω ∨ is-c-call-optionslist>,
         <s(4):is-SEMIC>)

(91)    is-c-call-optionslist =
        (<s(1):(<s(1):is-c-TASK>,
                 <s(2):is-Ω ∨
                 (<s(1):is-LEFT-PAR>,
                  <s(2):is-c-reference>,
                  <s(3):is-RIGHT-PAR>)) ∨
                 (<s(1):is-c-PRIORITY>,
                  <s(2):is-LEFT-PAR>,
                  <s(3):is-c-expression>,
                  <s(4):is-RIGHT-PAR>) ∨
                 (<s(1):is-c-EVENT>,
                  <s(2):is-LEFT-PAR>,
                  <s(3):is-c-reference>,
                  <s(4):is-RIGHT-PAR>),...)

```

30 June 1969

TRANSLATION OF PL/I INTO ABSTRACT SYNTAX

```

(92)  is-c-return-statement =
      (<s(1):is-c-RETURN>,
       <s(2):is-0 v
         (<s(1):is-LEFT-PAR>,
          <s(2):is-c-expression>,
          <s(3):is-RIGHT-PAR>),
       <s(3):is-SEMIC>)

(93)  is-c-incorporate-statement =
      (<s(1):is-c-INCORPORATE>,
       <s(2):is-LEFT-PAR>,
       <s(3):is-c-incorporate-specification>,
       <s(4):is-RIGHT-PAR>)

(94)  is-c-fetch-statement =
      (<s(1):is-c-FETCH>,
       <s(2):(<s-del:is-COMMA>,
              <s(1):is-c-reference>,...)>)

(95)  is-c-release-statement =
      (<s(1):is-c-RELEASE>,
       <s(2):(<s-del:is-COMMA>,
              <s(1):is-c-reference>,...)>)

(96)  is-c-wait-statement =
      (<s(1):is-c-WAIT>,
       <s(2):is-LEFT-PAR>,
       <s(3):(<s-del:is-COMMA>,
              <s(1):is-c-reference>,...)>,
       <s(4):is-RIGHT-PAR>,
       <s(5):is-0 v
         (<s(1):is-LEFT-PAR>,
          <s(2):is-c-expression>,
          <s(3):is-RIGHT-PAR>),
       <s(6):is-SEMIC>)

(97)  is-c-delay-statement =
      (<s(1):is-c-DELAY>,
       <s(2):is-LEFT-PAR>,
       <s(3):is-c-expression>,
       <s(4):is-RIGHT-PAR>,
       <s(5):is-SEMIC>)

(98)  is-c-exit-statement =
      (<s(1):is-c-EXIT>,
       <s(2):is-SEMIC>)

(99)  is-c-stop-statement =
      (<s(1):is-c-STOP>,
       <s(2):is-SEMIC>)

```

3.2.3 STORAGE MANIPULATING STATEMENTS

```

(100) is-c-assignment-statement =
      (<s(1):(<s-del:is-COMMA>,
                <s(1):is-c-reference>,...)>,
       <s(2):is-EQ>,
       <s(3):is-c-expression>,
       <s(4):is-0 v
         (<s(1):is-COMMA>,
          <s(2):is-c-BY>,
          <s(3):is-c-NAME>),
       <s(5):is-SEMIC>)

(101) is-c-allocate-statement =
      (<s(1):is-c-ALLOCATE>,
       <s(2):(<s-del:is-COMMA>,
              <s(1):is-c-based-allocate-item v is-c-controlled-allocate-item>,...)>,
       <s(3):is-SEMIC>)

(102) is-c-based-allocate-item =
      (<s(1):is-c-identifier>,
       <s(2):(<s(1):is-c-SET>,
              <s(2):is-LEFT-PAR>,
              <s(3):is-c-reference>,
              <s(4):is-RIGHT-PAR>,
              <s(5):is-0 v
                (<s(1):is-c-IN>,
                 <s(2):is-LEFT-PAR>,
                 <s(3):is-c-reference>,
                 <s(4):is-RIGHT-PAR>)) v
       (<s(1):is-c-IN>,
        <s(2):is-LEFT-PAR>,
        <s(3):is-c-reference>,
        <s(4):is-RIGHT-PAR>,
        <s(5):is-0 v
          (<s(1):is-c-SET>,
           <s(2):is-LEFT-PAR>,
           <s(3):is-c-reference>,
           <s(4):is-RIGHT-PAR>))>)

(103) is-c-controlled-allocate-item =
      (<s(1):is-0 v is-c-integer>,
       <s(2):is-c-identifier>,
       <s(3):is-0 v is-c-dimension-attribute>,
       <s(4):is-0 v
         (<s(1):is-c-string-attribute v is-c-area-attribute v
          is-c-initial-attribute>,...)>)

```

30 June 1969

TRANSLATION OF PL/I INTO ABSTRACT SYNTAX

(104) is-c-free-statement =
 (<s(1):is-c-FREE>,
 <s(2):(<s-del:is-COMMA>,
 <s(1):(<s(1):is-c-reference>,
 <s(2):is-Ω ∨
 (<s(1):is-c-IN>,
 <s(2):is-LEFT-PAR>,
 <s(3):is-c-reference>,
 <s(4):is-RIGHT-PAR>)>,...)>,
 <s(3):is-SEMIC>)

3.2.4 CONDITION AND ATTENTION HANDLING STATEMENTS

(105) is-c-on-statement =
 (<s(1):is-c-ON>,
 <s(2):is-c-condition>,
 <s(3):is-Ω ∨ is-c-SNAP>,
 <s(4):is-c-unconditional-statement ∨
 (<s(1):is-c-SYSTEM>,
 <s(2):is-SEMIC>))

(106) is-c-revert-statement =
 (<s(1):is-c-REVERT>,
 <s(2):is-c-condition>,
 <s(3):is-SEMIC>)

(107) is-c-signal-statement =
 (<s(1):is-c-SIGNAL>,
 <s(2):is-c-condition>,
 <s(3):is-SEMIC>)

(108) is-c-condition =
 is-c-prefix ∨ is-c-check-condition ∨ is-c-AREA ∨ is-c-named-io-condition ∨
 is-c-ERROR ∨ is-c-FINISH ∨ is-c-programmer-named-condition ∨
 is-c-attention-condition

(109) is-c-check-condition =
 (<s(1):is-c-CHECK>,
 <s(2):is-LEFT-PAR>,
 <s(3):(<s-del:is-COMMA>,
 <s(1):is-c-unsubscripted-reference>,...)>,
 <s(4):is-RIGHT-PAR>)

(110) is-c-no-check-condition =
 (<s(1):is-c-NOCHECK>,
 <s(2):is-LEFT-PAR>,
 <s(3):(<s-del:is-COMMA>,
 <s(1):is-c-unsubscripted-reference>,...)>,
 <s(4):is-RIGHT-PAR>)

TRANSLATION OF PL/I INTO ABSTRACT SYNTAX

30 June 1969

```

(111) is-c-named-io-condition =
      (<s(1):is-c-io-condition>,
       <s(2):is-LEFT-PAR>,
       <s(3):is-c-reference>,
       <s(4):is-RIGHT-PAR>)

(112) is-c-io-condition =
      is-c-BEGINVOLUME v is-c-ENDFILE v is-c-ENDPAGE v is-c-ENDVOLUME v is-c-KEY v
      is-c-NAME v is-c-PENDING v is-c-RECORD v is-c-TRANSMIT v is-c-UNDEFINEDFILE

(113) is-c-programmer-named-condition =
      (<s(1):is-c-CONDITION>,
       <s(2):is-LEFT-PAR>,
       <s(3):is-c-identifier>,
       <s(4):is-RIGHT-PAR>)

(114) is-c-attention-condition =
      (<s(1):is-c-ATTENTION>,
       <s(2):is-LEFT-PAR>,
       <s(3):(<s-del:is-COMMA>,
              <s(1):is-c-identifier>, ...),
       <s(4):is-RIGHT-PAR>)

(115) is-c-access-statement =
      (<s(1):is-c-ACCESS>,
       <s(2):is-c-ATTENTION>,
       <s(3):is-0 v
          (<s(1):is-LEFT-PAR>,
           <s(2):(<s-del:is-COMMA>,
                  <s(1):is-c-identifier>, ...),
           <s(3):is-RIGHT-PAR>),
       <s(4):(<s(1):is-c-ELSE>,
              <s(2):is-c-statement>) v is-SEMIC>)

(116) is-c-enable-statement =
      (<s(1):is-c-ENABLED>,
       <s(2):(<s-del:is-COMMA>,
              <s(1):(<s(1):is-c-attention-condition>,
                     <s(2):is-0 v
                        (<s(1):is-c-ACCESS v is-c-ASYNC v
                         (<s(1):is-c-EVENT>,
                          <s(2):is-LEFT-PAR>,
                          <s(3):is-c-reference>,
                          <s(4):is-RIGHT-PAR>)>, ...)>,...)>))

(117) is-c-disable-statement =
      (<s(1):is-c-DISABLE>,
       <s(2):is-c-attention-condition>)

```

3.2.5 INPUT AND OUTPUT STATEMENTS

30 June 1969

TRANSLATION OF PL/I INTO ABSTRACT SYNTAX

```

(118) is-c-open-statement =
      (<s(1):is-c-OPEN>,
       <s(2):(<s-del:is-COMMA>,
              <s(1):is-c-open-optionslist>, ...)>,
       <s(3):is-SEMIC>)

(119) is-c-open-optionslist =
      (<s(1):is-c-file-attribute v
       (<s(1):is-c-FILE>,
        <s(2):is-LEFT-PAR>,
        <s(3):is-c-reference>,
        <s(4):is-RIGHT-PAR>) v
       (<s(1):is-c-BLINESIZE>,
        <s(2):is-LEFT-PAR>,
        <s(3):is-c-expression>,
        <s(4):is-RIGHT-PAR>) v
       (<s(1):is-c-LINESIZE>,
        <s(2):is-LEFT-PAR>,
        <s(3):is-c-expression>,
        <s(4):is-RIGHT-PAR>) v
       (<s(1):is-c-PAGESIZE>,
        <s(2):is-LEFT-PAR>,
        <s(3):is-c-expression>,
        <s(4):is-RIGHT-PAR>) v
       (<s(1):is-c-TITLE>,
        <s(2):is-LEFT-PAR>,
        <s(3):is-c-expression>,
        <s(4):is-RIGHT-PAR>) v
       (<s(1):is-c-ENVIRONMENT>,
        <s(2):is-LEFT-PAR>,
        <s(3):is-c-env-option>,
        <s(4):is-RIGHT-PAR>) v is-c-VOLUME>, ...)

(120) is-c-close-statement =
      (<s(1):is-c-CLOSE>,
       <s(2):(<s-del:is-COMMA>,
              <s(1):is-c-close-optionslist>, ...)>,
       <s(3):is-SEMIC>)

(121) is-c-close-optionslist =
      (<s(1):(<s(1):is-c-FILE>,
              <s(2):is-LEFT-PAR>,
              <s(3):is-c-reference>,
              <s(4):is-RIGHT-PAR>) v
       (<s(1):is-c-ENVIRONMENT>,
        <s(2):is-LEFT-PAR>,
        <s(3):is-c-env-option>,
        <s(4):is-RIGHT-PAR>) v is-c-VOLUME>, ...)

(122) is-c-stream-io-statement =
      (<s(1):is-c-GET v is-c-PUT>,
       <s(2):is-c-stream-optionslist>,
       <s(3):is-SEMIC>)

```

TRANSLATION OF PL/T INTO ABSTRACT SYNTAX

30 JUNE 1969

(123) is-c-stream-optionslist =

```
(<s(1):(<s(1):is-c-FILE>,
         <s(2):is-LEFT-PAR>,
         <s(3):is-c-reference>,
         <s(4):is-RIGHT-PAR>) ∨
      (<s(1):is-c-BITSTRING>,
       <s(2):is-LEFT-PAR>,
       <s(3):is-c-expression>,
       <s(4):is-RIGHT-PAR>) ∨
      (<s(1):is-c-STRING>,
       <s(2):is-LEFT-PAR>,
       <s(3):is-c-expression>,
       <s(4):is-RIGHT-PAR>) ∨ is-c-data-specification ∨ is-c-COPY ∨
      (<s(1):is-c-SKIP>,
       <s(2):is-Ω ∨
          (<s(1):is-LEFT-PAR>,
           <s(2):is-c-expression>,
           <s(3):is-RIGHT-PAR>) ∨ is-c-PAGE ∨
          (<s(1):is-c-LINE>,
           <s(2):is-LEFT-PAR>,
           <s(3):is-c-expression>,
           <s(4):is-RIGHT-PAR>),...)
```

(124) is-c-data-specification =

is-c-data-directed ∨ is-c-edit-directed ∨ is-c-list-directed

(125) is-c-data-directed =

```
(<s(1):is-c-DATA>,
 <s(2):is-Ω ∨
    (<s(1):is-LEFT-PAR>,
     <s(2):is-c-datalist>,
     <s(3):is-RIGHT-PAR>))
```

(126) is-c-edit-directed =

```
(<s(1):is-c-EDIT>,
 <s(2):(<s(1):(<s(1):is-LEFT-PAR>,
                <s(2):is-c-datalist>,
                <s(3):is-RIGHT-PAR>,
                <s(4):is-c-formatlist>),...)>,...)>
```

(127) is-c-list-directed =

```
(<s(1):is-c-LIST>,
 <s(2):is-LEFT-PAR>,
 <s(3):is-c-datalist>,
 <s(4):is-RIGHT-PAR>)
```

(128) is-c-datalist =

```
(<s-del:is-COMMA>,
 <s(1):is-c-datalist-element>,...)
```

30 June 1969

TRANSLATION OF PL/I INTO ABSTRACT SYNTAX

```

(129)  is-c-datalist-element =
        (<s(1):is-LEFT-PAR>,
         <s(2):is-c-datalist>,
         <s(3):is-c-DO>,
         <s(4):is-c-do-specification>,
         <s(5):is-RIGHT-PAR>) ∨ is-c-expression

(130)  is-c-record-io-statement =
        (<s(1):is-c-READ ∨ is-c-WRITE ∨ is-c-REWRITE ∨
         (<s(1):is-c-LOCATE>,
          <s(2):is-c-identifier>) ∨ is-c-DELETE ∨ is-c-UNLOCK>,
         <s(2):is-c-record-optionslist>,
         <s(3):is-SEMIC>)

(131)  is-c-record-optionslist =
        (<s(1):(<s(1):is-c-FILE>,
                <s(2):is-LEFT-PAR>,
                <s(3):is-c-reference>,
                <s(4):is-RIGHT-PAR>) ∨
         (<s(1):is-c-EVENT>,
          <s(2):is-LEFT-PAR>,
          <s(3):is-c-reference>,
          <s(4):is-RIGHT-PAR>) ∨
         (<s(1):is-c-FROM>,
          <s(2):is-LEFT-PAR>,
          <s(3):is-c-reference>,
          <s(4):is-RIGHT-PAR>) ∨
         (<s(1):is-c-IGNORE>,
          <s(2):is-LEFT-PAR>,
          <s(3):is-c-expression>,
          <s(4):is-RIGHT-PAR>) ∨
         (<s(1):is-c-INTO>,
          <s(2):is-LEFT-PAR>,
          <s(3):is-c-reference>,
          <s(4):is-RIGHT-PAR>) ∨
         (<s(1):is-c-KEY>,
          <s(2):is-LEFT-PAR>,
          <s(3):is-c-expression>,
          <s(4):is-RIGHT-PAR>) ∨
         (<s(1):is-c-KEYTO>,
          <s(2):is-LEFT-PAR>,
          <s(3):is-c-reference>,
          <s(4):is-RIGHT-PAR>) ∨
         (<s(1):is-c-KEYFROM>,
          <s(2):is-LEFT-PAR>,
          <s(3):is-c-expression>,
          <s(4):is-RIGHT-PAR>) ∨
         (<s(1):is-c-SET>,
          <s(2):is-LEFT-PAR>,
          <s(3):is-c-reference>,
          <s(4):is-RIGHT-PAR>) ∨ is-c-NOLOCK>, ...)
```

(132) is-c-display-statement =

```

(<s(1):is-c-DISPLAY>,
 <s(2):is-LEFT-PAR>,
 <s(3):is-c-expression>,
 <s(4):is-RIGHT-PAR>,
 <s(5):is-Ω ∨
   (<s(1):is-c-REPLY>,
    <s(2):is-LEFT-PAR>,
    <s(3):is-c-reference>,
    <s(4):is-RIGHT-PAR>,
    <s(5):is-Ω ∨
      (<s(1):is-c-EVENT>,
       <s(2):is-LEFT-PAR>,
       <s(3):is-c-reference>,
       <s(4):is-RIGHT-PAR>) >)
    (<s(1):is-c-EVENT>,
     <s(2):is-LEFT-PAR>,
     <s(3):is-c-reference>,
     <s(4):is-RIGHT-PAR>,
     <s(5):is-c-REPLY>,
     <s(6):is-LEFT-PAR>,
     <s(7):is-c-reference>,
     <s(8):is-RIGHT-PAR>) >,
   <s(6):is-SEMIC>)

```

3.3 EXPRESSIONS

(133) is-c-expression =

```

is-c-expression-six ∨
(<s(1):is-c-expression>,
 <s(2):is-Ω>,
 <s(3):is-c-expression-six>)

```

(134) is-c-expression-six =

```

is-c-expression-five ∨
(<s(1):is-c-expression-six>,
 <s(2):is-AND>,
 <s(3):is-c-expression-five>)

```

(135) is-c-expression-five =

```

is-c-expression-four ∨
(<s(1):is-c-expression-five>,
 <s(2):is-c-comparison-operator>,
 <s(3):is-c-expression-four>)

```

30 June 1969

TRANSLATION OF PL/T INTO ABSTRACT SYNTAX

(136) is-c-comparison-operator =

```

is-GT v
(<elem(1):is-GT>,
 <elem(2):is-EQ>) v is-EQ v is-LT v
(<elem(1):is-LT>,
 <elem(2):is-EQ>) v
(<elem(1):is-NOT>,
 <elem(2):is-GT>) v
(<elem(1):is-NOT>,
 <elem(2):is-EQ>) v
(<elem(1):is-NOT>,
 <elem(2):is-LT>)

```

(137) is-c-expression-four =

```

is-c-expression-three v
(<s(1):is-c-expression-four>,
 <s(2):(<elem(1):is-OR>,
        <elem(2):is-OR>)>,
 <s(3):is-c-expression-three>)

```

(138) is-c-expression-three =

```

is-c-expression-two v
(<s(1):is-c-expression-three>,
 <s(2):is-PLUS v is-MINUS>,
 <s(3):is-c-expression-two>)

```

(139) is-c-expression-two =

```

is-c-expression-one v
(<s(1):is-c-expression-two>,
 <s(2):is-ASTER v is-SLASH>,
 <s(3):is-c-expression-one>)

```

(140) is-c-expression-one =

```

is-c-primitive-expression v
(<s(1):is-PLUS v is-MINUS v is-NOT>,
 <s(2):is-c-expression-one>) v
(<s(1):is-c-primitive-expression>,
 <s(2):(<elem(1):is-ASTER>,
        <elem(2):is-ASTER>)>,
 <s(3):is-c-expression-one>)

```

(141) is-c-primitive-expression =

```

(<s(1):is-LEFT-PAR>,
 <s(2):is-c-expression>,
 <s(3):is-RIGHT-PAR>) v is-c-reference v is-c-constant v is-c-isub

```

(142) is-c-reference =

```

(<s(1):is-Q v
   (<s(1):is-c-reference>,
    <s(2):(<elem(1):is-MINUS>,
           <elem(2):is-GT>)>),
   <s(2):is-c-basic-reference>)

```

```
(143) is-c-basic-reference =
      (<s(1):is-@ v
       (<s(1):(<s(1):is-c-identifier,
              <s(2):is-@ v is-c-subscriptlist>,
              <s(3):is-POINT>),...),
       <s(2):is-c-identifier>,
       <s(3):is-@ v
              (<s(1):is-c-subscriptlist>, ...)>)

(144) is-c-subscriptlist =
      (<s(1):is-LEFT-PAR>,
       <s(2):(<s-del:is-COMMA>,
              <s(1):is-c-expression v is-ASTER>,...)>,
       <s(3):is-RIGHT-PAR>)

(145) is-c-unsubscripted-reference =
      (<s-del:is-POINT>,
       <s(1):is-c-identifier>,...)

(146) is-c-constant =
      is-c-real-constant v is-c-imaginary-constant v is-c-sterling-constant v
      is-c-simple-string-constant v is-c-replicated-string-constant

(147) is-c-replicated-string-constant =
      (<s(1):is-LEFT-PAR>,
       <s(2):is-c-integer>,
       <s(3):is-RIGHT-PAR>,
       <s(4):is-c-simple-string-constant>)
```

3.4 IDENTIFIERS AND CONSTANTS

```
(148) is-c-identifier =
      (<elem(1):is-c-letter>,
       <elem(2):is-@ v
              (<elem(1):is-c-alphabetic-character>, ...)>)

(149) is-c-letter =
      is-c-A v is-c-B v is-c-C v is-c-D v is-c-E v is-c-F v is-c-G v is-c-H v
      is-c-I v is-c-J v is-c-K v is-c-L v is-c-M v is-c-N v is-c-O v is-c-P v
      is-c-Q v is-c-R v is-c-S v is-c-T v is-c-U v is-c-V v is-c-W v is-c-X v
      is-c-Y v is-c-Z v is-DOLLAR v is-COMM-AT v is-NUMBER-SIGN

(150) is-c-alphabetic-character =
      is-c-letter v is-c-digit v is-BREAK

(151) is-c-digit =
      is-0-CHAR v is-1-CHAR v is-2-CHAR v is-3-CHAR v is-4-CHAR v is-5-CHAR v
      is-6-CHAR v is-7-CHAR v is-8-CHAR v is-9-CHAR
```

30 June 1969

TRANSLATION OF PL/I INTO ABSTRACT SYNTAX

```

(152)  is-c-isub =
        (<elem(1):is-c-integer>,
         <elem(2):is-c-SUB>)

(153)  is-c-integer =
        (<elem(1):is-c-digit>, ...)

(154)  is-c-real-constant =
        (<elem(1):is-c-fixed-constant v is-c-float-constant>,
         <elem(2):is-Ω v is-c-B>)

(155)  is-c-fixed-constant =
        (<elem(1):is-c-integer>,
         <elem(2):is-Ω v is-POINT>) v
        (<elem(1):is-Ω v is-c-integer>,
         <elem(2):is-POINT>,
         <elem(3):is-c-integer>)

(156)  is-c-float-constant =
        (<elem(1):is-c-fixed-constant>,
         <elem(2):is-c-E>,
         <elem(3):is-Ω v is-PLUS v is-MINUS>,
         <elem(4):is-c-integer>)

(157)  is-c-imaginary-constant =
        (<elem(1):is-c-real-constant>,
         <elem(2):is-c-I>)

(158)  is-c-simple-string-constant =
        is-c-bit-string v is-c-character-string

(159)  is-c-bit-string =
        (<elem(1):is-APOSTR>,
         <elem(2):is-Ω v
          (<elem(1):is-c-bit>, ...)>,
         <elem(3):is-APOSTR>,
         <elem(4):is-c-B>)

(160)  is-c-bit =
        is-0-CHAR v is-1-CHAR

(161)  is-c-character-string =
        (<elem(1):is-APOSTR>,
         <elem(2):is-Ω v
          (<elem(1):is-c-string-character>, ...)>,
         <elem(3):is-APOSTR>)

```

(162) is-c-string-character =
 is-c-alphabetic-character ∨ is-BLANK ∨
 (<elem(1):is-APOSTR>,
 <elem(2):is-APOSTR>) ∨ is-EQ ∨ is-PLUS ∨ is-MINUS ∨ is-ASTER ∨ is-SLASH ∨
 is-LEFT-PAR ∨ is-RIGHT-PAR ∨ is-COMMA ∨ is-POINT ∨ is-SEMIC ∨ is-COLON ∨
 is-AND ∨ is-OR ∨ is-NOT ∨ is-GT ∨ is-LT ∨ is-QUEST ∨ is-PERC ∨
 is-c-extralingual-character

(163) is-c-sterling-constant =
 (<elem(1):is-c-integer>,
 <elem(2):is-POINT>,
 <elem(3):is-c-integer>,
 <elem(4):is-POINT>,
 <elem(5):is-c-fixed-constant>,
 <elem(6):is-c-L>)

3.5 PICTURES

(164) is-c-picture-specification =
 (<elem(1):is-APOSTR>,
 <elem(2):is-c-picture-string>,
 <elem(3):is-Ω ∨
 (<elem(1):is-c-F>,
 <elem(2):is-LEFT-PAR>,
 <elem(3):is-Ω ∨ is-PLUS ∨ is-MINUS>,
 <elem(4):is-c-integer>,
 <elem(5):is-RIGHT-PAR>),
 <elem(4):is-APOSTR>)

(165) is-c-picture-string =
 (<elem(1):is-Ω ∨
 (<elem(1):is-LEFT-PAR>,
 <elem(2):is-c-integer>,
 <elem(3):is-RIGHT-PAR>),
 <elem(2):is-c-picture-character>,
 <elem(3):is-Ω ∨ is-c-picture-string>)

(166) is-c-picture-character =
 is-c-A ∨ is-c-B ∨ is-c-C ∨ is-c-D ∨ is-c-E ∨ is-c-G ∨ is-c-H ∨ is-c-I ∨
 is-c-K ∨ is-c-M ∨ is-c-P ∨ is-c-R ∨ is-c-S ∨ is-c-T ∨ is-c-V ∨ is-c-X ∨
 is-c-Y ∨ is-c-Z ∨ is-DOLLAR ∨ is-1-CHAR ∨ is-2-CHAR ∨ is-3-CHAR ∨ is-4-CHAR ∨
 is-5-CHAR ∨ is-6-CHAR ∨ is-7-CHAR ∨ is-8-CHAR ∨ is-9-CHAR ∨ is-PLUS ∨
 is-MINUS ∨ is-ASTER ∨ is-SLASH ∨ is-COMMA ∨ is-POINT

3.6 IMPLEMENTATION-DEFINED PREDICATES

The following predicates are defined individually by any implementation and for this reason they do not have a corresponding rule in the concrete syntax.

is-c-external-option

30 June 1969

TRANSLATION OF PL/I INTO ABSTRACT SYNTAX

```
is-c-env-option
```

```
is-c-incorporate-specification
```

```
is-c-extralingual-character
```

3.7 PREDICATES DEFINING CONSTANT CHARACTER LISTS

Predicates of the form is-c-[char] and is-c-[name] occurring in the preceding sections or elsewhere in this document are defined by the following schemata. In the above predicates, [char] stands for a single capital letter, and [name] is any string of capital letters longer than one.

```
is-c-[char] =
```

```
  is-[char]-CHAR
```

where: [char] should be replaced by A, or B,..., or Z.

Example:

```
is-c-A =
```

```
  is-A-CHAR
```

```
is-c-[name] =
```

```
  (<elem(1) : is-char1

```

```
  ...
```

```
  <elem(n) : is-charn>)
```

where: char₁, char₂,... characterize uniquely the character values corresponding to the members of the string forming [name], and n (n>1) is the length of the string. This correspondence maps the letter A into A-CHAR, ..., Z into Z-CHAR.

Note: This scheme of predicates is introduced merely as a writing convenience.

Example:

```
is-c-END =
```

```
  (<elem(1) : is-E-CHAR>,
```

```
  <elem(2) : is-N-CHAR>,
```

```
  <elem(3) : is-D-CHAR>)
```


30 June 1969

TRANSLATION OF PL/I INTO ABSTRACT SYNTAX

4. THE TRANSLATOR

This chapter describes the translation from the abstract representation of a concrete program into an abstract program. This translation is performed by the function

```
translate(t)
```

which maps an object t satisfying the predicate is-c-program, described by the abstract representation of the concrete syntax given in chapter 3, into an object satisfying the predicate is-proper-program, described by the abstract syntax given in chapter 5.

Similarly as for the interpretation of an abstract program (cf./5/) the definition of the translation is reduced step by step to the translation of the components of the program text t . But there are two essential differences between the concepts of the interpreter and the translator:

- (1) The translator is specified by a function mapping a concrete program, in its abstract representation, into the corresponding abstract program, while the interpreter is specified by instructions mapping machine states into successor machine states.
- (2) The translation of a part of a program generally depends not only on this program part itself but also on the context in which it occurs within the complete program text, while the interpretation of a part of a program generally depends only on this program part itself (and the current machine state ξ , which may reflect contextual information if necessary).

To cover these two differences, the following concepts are applied in defining the translator:

- (1) Instead of the current machine state ξ , which is a hidden argument of each instruction of the interpreter, the complete program text t to be translated is a hidden argument of each function of the translator, which is not specified explicitly for each function. Throughout this chapter the letter t denotes this hidden argument, called the "text", which is the same object satisfying is-c-program for all functions.
- (2) Instead of objects to be translated, which are components of t , generally the selectors selecting them from t are specified as arguments of the functions. These selectors, called "text pointers" or shortly "pointers", are composed of selectors of the form $\text{elem}(n)$, $s(n)$, (n being integer values). They are usually named by the letters p , q , r .

Both arguments together, the hidden text t and an explicitly specified pointer p , constitute all necessary information: A part of t , namely $p(t)$ and the context of this part within t .

There are individual functions in the translator which in fact do not refer to the text t . As a general rule, one can say: all those functions, which handle components of t whose translation is context independent, have as arguments these components themselves and not their text pointers and do not refer to t . All those functions which have text pointers (or composite objects consisting of text pointers and possibly other components) as arguments, refer to the hidden argument t .

Context independent is the translation of constants, picture specifications and some very elementary program elements. Context dependent is in particular the translation of declarations and references and thereby of all parts containing them: e.g. expressions, statements, blocks, procedures.

Metavariables

t	is-c-program	the complete program text to be translated
p, q, r		text pointers, i.e. selectors composed of elem(n), s(n), for is-intg-val(n), to be applied to t

4.1 PROGRAM, PROCEDUREMetavariable

b	is-c-block•b(t)	pointer to a program, procedure or begin block
---	-----------------	--

4.1.1 PROGRAM

The translation of the program t is performed in two steps: First, the declaration part declaring all entry identifiers of the external procedures of the program and the body part of these external procedures are composed. This is performed nearly in the same way as the composition of declaration parts and body parts which are local to procedures or begin blocks.

Second, in an implementation defined way, all external entry identifiers declared throughout the program are resolved by incorporation of corresponding external procedures to be found, e.g., in a library. This process, described by the function incorporate, involves also the incorporation of external procedures referred to within the included procedures, and so on. The result is a program, in which to each contained external entry declaration there is a corresponding external procedure body.

(1) translate(t) =
 incorporate(translate-1(t), ext-entry-set(t), LIBRARY)

Note: LIBRARY is an implementation defined object.

(2) translate-1(t) =
 $\mu_0 (\langle s\text{-decl-part:progr-decl-part(I)} \rangle, \langle s\text{-body-pt:mk-body-pt(T)} \rangle)$

Ref.: mk-body-pt 4-5(15)

Note: I is the unity selector, i.e., the pointer to the text t itself

30 June 1969

TRANSLATION OF PL/I INTO ABSTRACT SYNTAX

(3) progr-decl-part(b) =
 $\mu(\text{mk-decl-part}(b); [\langle s \text{-scope} \cdot id : \text{EXT} \rangle \mid \text{is-id}(id) \wedge \neg \text{is-Q-id} \cdot \text{mk-decl-part}(b)])$

for:b = I

Ref.: mk-decl-part 4-8(27)

Note: mk-decl-part(I) contains just the declaration parts of all external entrys of the whole program. For them the external scope attribute is inserted.

(4) ext-entry-set(t) =
 $\{id \mid (\exists p) (id = \text{mk-id-1}(p) \wedge \text{is-decl-cont}(p) \wedge \text{is-ENTRY-CONST-type-attr} \cdot \text{prel-decl}(\langle p \rangle) \wedge \text{is-EXT-scope-attr} \cdot \text{prel-decl}(\langle p \rangle) \wedge \text{is-Q-id} \cdot \text{progr-decl-part}(I))\}$

Ref.: mk-id-1 4-90(353)
 $\text{is-decl-cont } 4-11(36)$
 $\text{type-attr } 4-24(87)$
 $\text{prel-decl } 4-9(30)$
 $\text{scope-attr } 4-26(93)$

(5) incorporate(program,id-set,library) =
 $\text{for:is-proper-program}(program), \text{is-id-set}(id-set), \text{is-LIBRARY}(library)$

Ref.: is-proper-program 5-1(1)

Note: This function is implementation defined. The result is an object program-1 satisfying the following conditions.

(6) is-proper-program(program-1),
 $(\forall \sigma) (\neg \text{is-Q-}\sigma(\text{program}) \Rightarrow \sigma(\text{program-1}) = \sigma(\text{program}))$,
 $(\forall id) (id \in \text{id-set} \Rightarrow \neg \text{is-Q-id} \cdot \text{s-decl-part}(\text{program-1}))$,
 $(\exists t-1) (\text{is-c-program}(t-1) \wedge \text{translate-1}(t-1) = \text{program-1} \wedge \text{is-}\{\} \cdot \text{ext-entry-set}(t-1))$,
 $\text{incorporate}(\text{program}, \{\}, \text{LIBRARY}) = \text{program}$

Ref.: is-proper-program 5-1(1)

4.1.2 AUXILIARY FUNCTIONS CONCERNING BLOCK STRUCTURE

The following functions concerning the block structure of a concrete PL/I program are needed throughout the translator definition, particularly for the determination of the scope of declarations.

(7) is-block-p(b) =
 $\text{is-c-block-}\ast b(t)$

```
(8)    block-p(p) =
        (Ub) (is-c-block•b(t) & is-local-to(b,p))

(9)    is-c-block =
        is-c-program v is-c-procedure v is-c-begin-block

(10)   is-local-to(b,p) =
        is-contained-in(b,p) & ~ (Eb-1) (is-contained-in(b,b-1) & is-contained-in(b-1,p))
```

Note: This function characterizes the "scope" of a block.

```
(11)   is-contained-in(b,p) =
        is-c-program•b(t) & b => p v is-c-begin-block•b(t) &
        (b => p v (S1•((Ub) (b = S2•q)))) => p) v is-c-procedure•b(t) & b => p &
        ~(S2•b) => p &
        (~ (Er-1) (is-c-entry•r-1(t) & (S1•r-1) => p) v
        (~r-2) (is-c-block•r-2(t) & b => r-2 => p)))
```

Note: This definition defines the relation between a block and a component contained in it. Prefixes are included, labels and entries excluded.

```
(12)   proc-p(p) =
        (Ub) (is-c-procedure•b(t) & b => p & ~ (Eb-1) (is-c-block•b-1(t) & b => b-1 => p))

(13)   ext-proc-p(p) =
        (Ub) (is-ext-proc-p(b) & is-contained-in(b,p))

(14)   is-ext-proc-p(b) =
        (~n) (b = Sn & ~is-Q•Sn(t))
```

4.1.3 PROCEDURE BODY

This section describes the construction of the body-part of a program, procedure body or begin block. It is performed by collecting all local procedures and translating them. Each single procedure is translated into a body which main parts are:

- (1) an entry part containing information about all entries of the procedure (but not that information given by an entry declaration).
- (2) a declaration part containing the fully specified declarations for all unqualified names declared explicitly or implicitly in the procedure (including entry declarations)
- (3) a body part handling the procedures contained in the procedure
- (4) a condition part containing the prefix condition information of the procedure

30. June 1969

TRANSLATION OF PL/I INTO ABSTRACT SYNTAX

(5) a statement list of all statements local to the procedure

(15) $\text{mk-body-pt}(b) =$

$$\mu_0(\{\langle \text{id:trans-body}(p) \rangle \mid \text{is-c-procedure} \cdot p(t) \wedge \text{is-local-to}(b,p) \wedge \\ \text{id} = \text{mk-id-1}(s_1 \cdot s_1 \cdot s_2 \cdot p)\})$$

Ref.: is-local-to 4-4(10)
 mk-id-1 4-90(353)

Note: The body selector "id" corresponds to the first entry-name of the selected procedure

(16) $\text{trans-body}(b) =$

$$\mu_0(\langle s\text{-entry-pt:mk-entry-pt}(b) \rangle, \langle s\text{-decl-pt:mk-decl-part}(b) \rangle, \\ \langle s\text{-body-pt:mk-body-pt}(b) \rangle, \langle s\text{-cond-pt:trans-cond-part}(s_1 \cdot b) \rangle, \\ \langle s\text{-st-list:mk-st-list}(b) \rangle, \langle s\text{-reorder:mk-reorder}(b) \rangle, \\ \langle s\text{-recursive:mk-recursive}(b) \rangle)$$

Ref.: mk-decl-part 4-8(27)
 trans-cond-part 4-62(243)

(17) $\text{mk-entry-pt}(b) =$

$$\mu_0(\{\langle \text{id:mk-entry-point}(p, \text{entry}_1) \rangle \mid \text{is-entry-cont}(p) \wedge b \Rightarrow p \wedge \\ \text{id} = \text{mk-id-1}(p) \wedge \neg(\exists q) (\text{is-c-procedure} \cdot q(t) \wedge b \Rightarrow q \Rightarrow p) \wedge \text{test}_1\})$$

where:

$$\begin{aligned} \text{entry}_1 &= (\exists \rightarrow (bq) (\text{is-c-entry} \cdot q(t) \wedge q \Rightarrow p), \\ &\quad T \rightarrow \text{proc-p}(p)) \\ \text{test}_1 &= (\neg(\exists q) ((\text{is-c-iterated-group} \vee \text{is-c-begin-block}) \cdot q(t) \wedge b \Rightarrow q \Rightarrow p) \rightarrow T) \end{aligned}$$

Ref.: is-entry-cont 4-11(40)
 mk-id-1 4-90(353)
 proc-p 4-4(12)

TRANSLATION OF PL/I INTO ABSTRACT SYNTAX

30 June 1969

(18) $\text{mk-entry-point}(p, q) =$

$\mu_0 \langle s\text{-st-loc}: \langle 1 \rangle, s\text{-param-list}: \text{LIST } \underset{n=1}{\text{mk-id-ref}}(s_n \cdot s_2 \cdot s_3 \cdot q),$
 $s\text{-length}: s_4 \cdot q(),$
 $s\text{-ret-type}: \text{mk-ret-type-1}(\text{nd}_1) \rangle$

$\mu_0 \langle s\text{-st-loc}: \text{mk-indexlist}(p, \text{proc-p}(p)),$
 $s\text{-length}: s_4 \cdot q(),$
 $s\text{-param-list}: \text{LIST } \underset{n=1}{\text{mk-id-ref}}(s_n \cdot s_2 \cdot s_3 \cdot q),$
 $s\text{-ret-type}: \text{mk-ret-type-1}(\text{nd}_1) \rangle$

where:

$\text{nd}_1 = \mu(\text{prel-decl}(p); \langle s\text{-attr-ds}: \{\text{ad} \mid \text{ret}_1\} \rangle)$
 $\text{ret}_1 = (\exists \rightarrow \text{bad}) (q \Rightarrow \text{ad} \wedge \text{ad} < s_6 \cdot q \wedge \text{is-c-returns-attribute-ad}(t)),$
 $T \rightarrow \top)$

for: $\text{is-c-basic-reference}\circ\text{p}(t), (\text{is-c-procedure} \vee \text{is-c-entry})\circ\text{a}(t)$

Ref.: mk-ret-type-1 4-45 (183)
 proc-p 4-4 (12)
 prel-decl 4-9 (30)

Note: The statement locator component specifies which statement is executed first, when entering the corresponding procedure or entry.

(19) $\text{mk-id-ref}(p) =$

$\mu_0 \langle s\text{-id}: \text{mk-id-1}(p) \rangle$

Ref.: mk-id-1 4-9^ (353)

(20) $\text{mk-indexlist}(p, q) =$

$s_1 \cdot q \Rightarrow p \vee s_2 \cdot \sigma \Rightarrow p \rightarrow \langle \rangle$

$\text{is-c-if-clause} \circ s_1 \cdot s_3 \cdot q(t) \rightarrow \langle s_2 \cdot s_3 \cdot q \Rightarrow p \rangle \text{~} \text{mk-indexlist}(p, (s(n_1)) \cdot s_3 \cdot q)$

$\text{is-c-access-statement} \circ s_3 \cdot \sigma(t) \wedge s_4 \cdot s_3 \cdot \pi \Rightarrow p \rightarrow \langle F \rangle \text{~} \text{mk-indexlist}(p, s_2 \cdot s_4 \cdot s_3 \cdot q)$

$T \rightarrow \langle \text{card}(\text{set}_1) + 1 \rangle \text{~} \text{mk-indexlist}(p, \text{sent}_1)$

where:

$n_1 = (\cup n) (s_n \cdot s_3 \cdot \sigma \Rightarrow p)$
 $\text{set}_1 = fr \mid \text{is-local-sentence}(r, q) \wedge r < p \wedge$
 $\{\text{is-c-declaration-sentence} \vee \text{is-c-statement}\} \circ r(t)\}$
 $\text{sent}_1 = (\cup r) (r \Rightarrow p \wedge \text{is-local-sentence}(r, \sigma))$

Note: The indexlist is a non empty list of positive integers, Ts and Fs constituting a trace to the statement which is executed first, when the label or entry p(t) is reached. An integer i denotes the i-th statement of the statementlist of a block, T the then-component of an if-statement and F the else component of an if- or access-statement. For a more detailed description of an indexlist see chapter 9 of /6/.

6 4. THE TRANSLATOR

30 June 1969

TRANSLATION OF PL/I INTO ABSTRACT SYNTAX

(21) $\text{is-local-sentence}(p, q) =$
 $(\text{is-c-block} \vee \text{is-c-group}) * q(t) \wedge$
 $(\text{is-c-sentence} \wedge \neg \text{is-c-format-sentence} \vee \text{is-c-end-clause}) * p(t) \wedge q \Rightarrow p \wedge$
 $\neg (\exists r) (\text{is-c-sentence} * r(t) \wedge q \Rightarrow r \Rightarrow p)$

(22) $\text{mk-st-list}(p) =$
 $\text{length-st-p-list}(p)$
 $\text{LIST } (\text{trans-st-elem}(n, \text{st-p-list}(p)))$
 $n=1$

Ref.: trans-st 4-60(239)

(23) $\text{st-p-list}(p) =$
 $\text{collect}(\{q \mid \text{is-local-sentence}(q, p) \wedge \neg (\text{is-c-procedure} \vee \text{is-c-entry}) * q(t)\})$

(24) $\text{mk-reorder}(b) =$
 $\exists \rightarrow \text{mk-opt-1}((\cup p) ((\text{is-c-ORDER} \vee \text{is-c-REORDER}) * p(t) \wedge (s_2 * b \Rightarrow p \vee s_5 * b \Rightarrow p)))$
 $\text{is-ext-proc-p}(b) \rightarrow \Omega$
 $T \rightarrow \text{mk-reorder}(\text{block-p}(b))$

Ref.: is-ext-proc-p 4-4(14)
block-p 4-4(8)

Note: If $b(t)$ does not contain an order or reorder option, the hierarchy of blocks in which $b(t)$ is nested is examined in bottom to top order until an order or reorder option, if any, is found.

(25) $\text{mk-opt-1}(p) =$
 $(\text{is-c-RECURSIVE} \vee \text{is-c-REORDER} \vee \text{is-c-REDUCIBLE}) * p(t) \rightarrow *$
 $T \rightarrow \Omega$

(26) $\text{mk-recursive}(b) =$
 $\exists \rightarrow \text{mk-opt-1}((\cup p) (s_5 * b \Rightarrow p \wedge \text{is-c-RECURSIVE} * p(t)))$
 $T \rightarrow \Omega$

4.2 DECLARATIONS

The translation of declarations is performed in three steps:

- (1) The recognition of all declarations, whether they are explicitly or contextually specified in the concrete program. For each declaration recognized a "preliminary declaration" is constructed. This is an object giving information about: the qualified name, i.e. the identifier list being declared, the scope, and the attributes specified explicitly or contextually. This step includes recognition of the successor relation of



structure declarations (by means of level numbers or the like attribute) and testing for illegal multiple declarations. For each block the set of preliminary declarations local to it is collected.

- (2) The completion of the preliminary declarations by the attributes specified by default sentences.
- (3) The construction of declarations as specified by the abstract syntax. This step in particular includes testing of attributes for consistency.

Metavariables

b	is-c-block•b(t)	a pointer to a begin block, procedure or program, specifying the scope of a declaration
pd	is-prel-decl	preliminary declaration
ad	is-attr-descr	attribute descriptor, i.e. either a pointer to an explicitly specified attribute or an elementary object denoting a contextual attribute
id	is-id	an identifier
idl	is-id-list	an identifier list
scr	is-succ-scr	list of pointers to declarations which are successors of each other.

(27) mk-decl-part(b) =

```
μo({<id:mk-decl(pd) | is-id(id) & pd ∈ prel-decl-set(b) &
s-id-list(pd) = <id>})
```

Ref.: mk-decl 4-23(81)
 prel-decl-set 4-9(28)

4.2.1 CONSTRUCTION OF PRELIMINARY DECLARATIONS

In this section for each block the set of its preliminary declarations is constructed. The function prel-decl-set-1(b) collects all level-one preliminary declarations having the block to which b points as scope.

For each declaration, whether it is declared explicitly or contextually a preliminary declaration is constructed. These preliminary declarations exist not only for level-one declarations but for each individual component of structure declarations too. These preliminary declarations consist of

- (1) the fully qualified name of the declared item,
- (2) the block pointer b giving the scope,
- (3) the set of attribute descriptors. These are the pointers to the explicitly specified attributes in declaration sentences, and elementary objects denoting attributes specified by context: PARAM, FORMAT, LABEL, ENTRY, BUILTIN, FILE, TASK, EVENT, PTR, AREA, COND, STRUCT, SUCC.

Since by means of the chaining specified by the like attribute the same declaration may be taken as successor of different structure declarations, a single declaration in the concrete program need not uniquely determine a declared

30 June 1969

TRANSLATION OF PL/I INTO ABSTRACT SYNTAX

item. Therefore in any case the complete sequence of pointers of successive structure declarations is taken as argument of the function prel-decl-1, which constructs the preliminary declaration.

The same declaration may be specified by more than one specifications. This is the case, e.g. for an identifier occurring in a parameter list on the one hand and in a declaration sentence on the other hand. In such cases the different contexts lead to the same preliminary declaration which collects the attributes of all of them. In particular, for a not explicitly declared identifier, all occurrences in references lead to the same preliminary declaration and contribute to its attribute descriptor set. If such an identifier occurs in conflicting contexts, e.g., as a file identifier and as a pointer variable, the final construction of a declaration will fail.

```
(28) prel-decl-set(b) =
      {x | (Ey) (y ∈ prel-decl-set-1(b) & x = μ(y;
                                                     <s-attr-ds:default(s-attr-ds(y),
                                                     s-block-p(y),
                                                     def-name•last•s-id-list(y))>)}
```

Ref.: default 4-19(69)

Note: This function completes in the same way as the function prel-decl the attribute descriptor set of preliminary declarations with respect to specified default sentences.

```
(29) prel-decl-set-1(b) =
      ~ (Eseq-1,seq-2) (is-local-to(b,head(seq-1)) & is-mult-decl(seq-1,seq-2)) &
      ~ (Ep) (is-local-to(b,p) & is-decl-cont(p) &
            (is-invalid-ln(p) ∨ is-invalid-like(p))) -->
      {pd | (Ex) ((is-succ-seq(x) ∨ is-non-expl-decl(x)) & pd = prel-decl-1(x)) &
           s-block-p(pd) = b}
```

Ref.: is-local-to 4-4(10)
 is-mult-decl 4-15(55)
 is-decl-cont 4-11(36)
 is-invalid-ln 4-13(47)
 is-invalid-like 4-15(54)
 is-succ-seq 4-12(41)
 is-non-expl-decl 4-16(57)

```
(30) prel-decl(x) =
      μ(pd1;
         <s-attr-ds:default(s-attr-ds(pd1),s-block-p(pd1),
                           def-name•last•s-id-list(pd1))>)
```

where:
 pd₁ = prel-decl-1(x)

for:(is-succ-seq ∨ is-non-expl-decl)(x)

Ref.: default 4-19(69)
 is-succ-seq 4-12(41)
 is-non-expl-decl 4-16(57)

(31) prel-decl-1(x) =

```

    is-succ-seq(x) --+
       $\mu_0$ (<s-id-list: LIST mk-id-1•elem(n,x)>, <s-block-p: block-p•head(x)>,
          length(x)
          n=1
          <s-attr-ds:(ad | (3seq) (is-eq-id1(seq,x) & is-attr-of(last(seq),ad)) &
          is-predec-dens(x,ad))>)

    is-non-expl-decl(x) --+
       $\mu_0$ (<s-id-list:ref-id-list(x)>, <s-block-p:ext-proc-p(x)>,
          <s-attr-ds:(ad |
          (3p) (is-non-expl-decl(p) & ref-id-list(p) = ref-id-list(x) &
          ext-proc-p(p) = ext-proc-p(x) & is-attr-of(p,ad)))>)

for:(is-succ-seq v is-non-expl-decl)(x)

Ref.:   is-succ-seq 4-12(41)
        mk-id-1 4-90(353)
        block-p 4-4(8)
        is-eq-id1 4-15(56)
        is-attr-of 4-18(67)
        is-predec-dens 4-18(68)
        is-non-expl-decl 4-16(57)
        ref-id-list 4-84(330)
        ext-proc-p 4-4(13)
```

(32) is-prel-decl =

```

(<s-id-list:is-id-list>,
 <s-block-p:is-block-p>,
 <s-attr-ds:is-attr-descr-set>)
```

Ref.: is-block-p 4-3(7)

(33) is-attr-descr(ad) =

```

(is-c-attribute v is-c-dimension-attribute v is-c-returns-attribute v
  is-c-value-clause)•ad(t) v
ad ∈
{FORMAT, LABEL, PARAM, ENTRY, FILE, BUILTIN, TASK, EVENT, AREA, PTR, COND, STRUCT, SUCC}
```

(34) def-name(id) =

```

~(3n) (is-POINT•elem(n,chli)) -- id
T -- def-name•mk-id(tail(chli))
```

where:
chli = (6chli)(id = mk-id(chli))

Ref.: mk-id 4-90(354)

Note: This function is necessary only for qualified initial labels, due to the special handling of them by the function mk-id-1.

30 June 1969

TRANSLATION OF PL/I INTO ABSTRACT SYNTAX

4.2.1.1 Explicit declaration contexts

In this section the predicates are defined which describe the property of a pointer to an identifier, to be in the context of an explicit declaration. These identifier contexts are the following: declarations in a declaration sentence, identifiers in parameter lists, label constants and entry identifiers.

Note, the initial labels are considered in this respect as identifiers. The translator creates for them identifiers and uses them as initial values for implied initial attributes.

- ```
(35) is-expl-decl =
 is-decl-cont ∨ is-param-cont ∨ is-label-cont ∨ is-format-cont ∨ is-entry-cont

(36) is-decl-cont(p) =
 (Ǝq) (is-c-declare-sentence*q(t) & is-c-identifier*p(t) & q => p)

(37) is-param-cont(p) =
 is-c-identifier*p(t) &
 (Ǝσ, n) (is-c-procedure*σ(t) & p = sn*s2*s4*q ∨ is-c-entry*q(t) & p = sn*s2*s3*q)

(38) is-label-cont(p) =
 -is-Ω*p(t) & (Ǝσ, n) (is-c-declaration-sentence*q(t) & p = s1*sn*s1*q ∨
 (is-c-statement ∨ is-c-balanced-statement ∨ is-c-end-clause)*q(t) &
 p = s1*sn*s2*q)

 for:is-c-basic-reference*p(t)

(39) is-format-cont(p) =
 -is-Ω*p(t) & (Ǝσ, n) (is-c-format-sentence*q(t) & p = s1*sn*s2*q)

 for:is-c-basic-reference*p(t)

(40) is-entry-cont(p) =
 -is-Ω*p(t) & (Ǝσ, n) (is-c-procedure*q(t) & p = s1*sn*s2*q ∨ is-c-entry*σ(t) &
 p = s1*sn*s1*q)

 for:is-c-basic-reference*p(t)
```

4.2.1.2 Successor relation for structure declarations

The successor declarations of a structure declaration are specified in a concrete PL/I program either by level numbers or by the like attribute. The relation between a declaration and its successors is specified in this section. Based on this relation the predicate is-succ-seq is specified which is true for lists of explicit declaration pointers, which, starting with a level-one declaration, are successors of each other. Though explicit declarations other than in declaration sentences never have successors, the definition of successor sequences includes the one-element-lists of such declarations.

## TRANSLATION OF PL/T INTO ABSTRACT SYNTAX

30 June 1969

(41) is-succ-seq (seq) =  
 is-list (seq) & is-expl-decl-head (seq) & ln-head (seq) = +1 &  
 $\text{length}(\text{seq}) - 1$   
 Et is-succ-of (elem (n, seq), elem (n + 1, seq))  
 $n = 1$

Ref.: is-expl-decl 4-11(35)  
 ln 4-13(45)

(42) is-succ-of (p, q) =  
 is-ln-succ-of (p, q)  $\vee$  is-like-succ-of (p, q)  
 for:is-decl-cont (p), is-decl-cont (q)  
 Ref.: is-ln-succ-of 4-12(43)  
 is-like-succ-of 4-13(48)  
 is-decl-cont 4-11(36)

4.2.1.2.1 SUCCESSORS SPECIFIED BY LEVEL NUMBERS

This section defines the successor relation specified by level numbers. Since a level-one declaration having no successors has the level number 1 by default without necessarily being specified, but on the other hand it has to be tested whether each major structure has explicitly the level number 1, the auxiliary default level number is taken to be -1 by the translator.

(43) is-ln-succ-of (p, q) =  
 is-ln-subel-of (p, q)  $\&$   $\neg (\exists r) (\text{is-ln-subel-of}(p, r) \& \text{is-ln-subel-of}(r, q))$   
 for:is-decl-cont (p), is-decl-cont (q)  
 Ref.: is-decl-cont 4-11(36)

(44) is-ln-subel-of (p, q) =  
 is-decl-cont (p)  $\&$  is-decl-cont (q)  $\&$   
 $(\exists r) (\text{is-c-declaration-sentence-r}(t) \& r \Rightarrow p \& r \Rightarrow q) \& p < q \&$   
 $1 < \ln(p) < \ln(q) \& \neg (\exists r) (\text{is-decl-cont}(r) \& p < r < q \& \ln(r) \leq \ln(p))$

Ref.: is-decl-cont 4-11(36)

Note: By this definition, declaration with default level number -1 may never have subelements.

30 June 1969

TRANSLATION OF PL/T INTO ABSTRACT SYNTAX

(45)  $\text{ln}(p) =$

$$\exists \rightarrow \text{const-val}((\exists q) (\text{is-ln-of}(p,q))) (t)$$

$$t \rightarrow -1$$

for:is-expl-decl(p)

Ref.: const-val 4-89(347)  
is-expl-decl 4-11(35)

(46)  $\text{is-ln-of}(p,q) =$

$$\text{is-decl-cont}(p) \& \text{is-c-integer} \cdot q(t) \&$$

$$(\exists r) (\text{is-c-declaration} \cdot r(t) \& r \Rightarrow p \& q = s_1 \cdot r)$$

Ref.: is-decl-cont 4-11(36)

(47)  $\text{is-invalid-ln}(p) =$

$$\text{ln}(p) \neq *1 \& \neg (\exists q) (\text{is-ln-succ-of}(q,p))$$

for:is-decl-cont(p)

Ref.: is-decl-cont 4-11(36)

Note: This predicate excludes : "major structures" not at level 1, major structures without explicit level number -1, and specification of level number 0. It is used by the function prel-decl-set-1(b).

#### 4.2.1.2.2 SUCCESSORS SPECIFIED BY THE LIKE-ATTRIBUTE

By means of the like-attribute one can specify, that the successors of a declaration shall be the same ones as the successors of another declaration referred to by an unsubscripted reference contained in the like-attribute. The referencing mechanism used for that purpose is nearly the same as the general referencing mechanism used to find a declaration corresponding to a reference. The difference is that within the block of the reference itself only level-number-successor sequences are referred to.

It is the intention of the language, that successor declarations specified by the like attribute are thought to be copied syntactically into the place of the like attribute. This has the consequence that expressions occurring in attributes of such successor declarations are to be translated as if they occurred in the block of the structure declaration even if the like attribute refers to a declaration in an outer block. For this purpose serve the functions env-trans-expr and env-trans-ref which have a block pointer as additional argument. This is relevant for expressions in dimension attribute, string length, area size, and in the initial attribute.

(48)  $\text{is-like-succ-of}(p,q) =$

$$\text{is-like}(p) \& \text{is-ln-succ-of}(\text{like-ref}(p),q)$$

for:is-decl-cont(p),is-decl-cont(q)

cont'd

## TRANSLATION OF PL/I INTO ABSTRACT SYNTAX

30 June 1969

Ref.: is-in-succ-of 4-12(43)  
 is-decl-cont 4-11(36)

(49) is-like(p) =  
 is-decl-cont(p) & ( $\exists q$ ) (is-c-like-attribute- $\sigma(t)$  & is-attr-of(p,q))

Ref.: is-decl-cont 4-11(36)  
 is-attr-of 4-18(67)

(50) like-ref(p) =  
 like-ref-3•like-ref-2(block-p(p),ref-id-list•like-ref<sub>1</sub>(p))

where:

like-ref<sub>1</sub> =  $s_2 \cdot ((\forall q) (is-c-like-attribute- $\sigma(t)$  & is-attr-of(p,q)))$

for:is-decl-cont(p)

Ref.: block-p 4-4(8)  
 ref-id-list 4-84(330)  
 is-attr-of 4-18(67)  
 is-decl-cont 4-11(36)

(51) like-ref-2(b,idl) =  
 $\exists \rightarrow (\exists pd) (is-in-epd(b,pd) \& idl = s-id-list(pd))$   
 $\exists \rightarrow (\exists pd) (is-in-epd(b,pd) \& is-id-list-match(idl,s-id-list(pd)))$   
 $\exists \rightarrow expl-decl-ref(block-p(b),idl)$

Ref.: is-id-list-match 4-84(329)  
 expl-decl-ref 4-94(327)  
 block-p 4-4(8)

Note: Compare the function expl-decl-ref, which generally determines the preliminary declaration referred to by a reference.

(52) is-in-epd(b,pd) =  
 $pd \in prel-decl-set(b) \&$   
 $(\exists seq) (is-succ-seq(seq) \& pd = prel-decl(seq) \&$   
 $(length(seq) = 1 \vee is-in-subel-of(head(seq),tail(seq))))$

Ref.: prel-decl-set 4-9(28)  
 is-succ-seq 4-12(41)  
 prel-decl 4-9(30)  
 is-in-subel-of 4-12(44)

(53) like-ref-3(pd) =  
 last(( $\forall seq$ ) (pd = prel-decl(seq) & is-decl-cont•last(seq)))

cont'd

30 June 1969

TRANSLATION OF PL/I INTO ABSTRACT SYNTAX

Ref.: prel-decl 4-9(30)  
is-decl-cont 4-11(36)

Note: This function is somehow the inverse function of prel-decl.

(54) is-invalid-like(p) =

```
is-like(p) &
(~(3q) (is-like-succ-of(p,q)) v (3q) (is-ln-succ-of(p,q)) v
(3q) ((q = like-ref(p) v is-ln-subel-of(like-ref(p),q)) & is-like(q)))
```

for:is-decl-cont(p)

Ref.: is-ln-succ-of 4-12(43)  
is-ln-subel-of 4-12(44)  
is-decl-cont 4-11(36)

Note: This predicate, testing illegal use of the like-attribute is used by the function prel-decl-set-1.

#### 4.2.1.3 Multiple declarations

More than one declaration of the same qualified name within the same block is illegal, except for the merging of an explicit declaration in a declaration sentence with an explicit declaration in a parameter or entry context. This is tested by the function is-mult-decl, which is used by the function prel-decl-set-1.

(55) is-mult-decl(seq-1,seq-2) =

```
seq-1 # seq-2 & is-eq-idl(seq-1,seq-2) &
~(is-decl-cont(p1) & is-param-cont(p2) v is-decl-cont(p1) & is-entry-cont(p2) v
is-param-cont(p1) & is-param-cont(p2))
```

where:

p<sub>1</sub> = head(seq-1)  
p<sub>2</sub> = head(seq-2)

Ref.: is-decl-cont 4-11(36)  
is-param-cont 4-11(37)  
is-entry-cont 4-11(40)

(56) is-eq-idl(seq-1,seq-2) =

```
is-succ-seq(seq-1) & is-succ-seq(seq-2) &
block-p-head(seq-1) = block-p-head(seq-2) & length(seq-1) = length(seq-2) &
length(seq-1)
Et elem(n,seq-1)(t) = elem(n,seq-2)(t)
n=1
```

Ref.: is-succ-seq 4-12(41)  
block-p 4-4(8)

4.2.1.4 Contextual declarations

Any reference occurring in a program, which has no matching declaration, leads to a non explicit declaration. The same is valid for single identifiers, i.e. identifiers occurring in other contexts than in a reference or an explicit declaration, e.g. for entry identifiers in call statements. The attributes given to these declarations are determined by the context of the reference or identifier. If the reference does not occur in one of the contexts specified in the following, the declaration is called "implicit".

- ```
(57) is-non-expl-decl(p) =
      (is-ref-cont v is-single-id-cont v is-sysin-cont v is-sysprint-cont)(p) &
      is-@expl-decl-ref(block-p(p),ref-id-list(p)) & length-ref-id-list(p) = 1

      Ref.:    is-ref-cont 4-83(325)
              is-single-id-cont 4-90(355)
              expl-decl-ref 4-84(327)
              block-p 4-4(8)
              ref-id-list 4-84(330)

(58) is-builtin-cont(p) =
      ( $\exists q$ ) ((is-c-call-statement $\circ q(t)$  v is-c-initial-call $\circ q(t)$ ) & p = s2 $\circ s_2 \circ q$ ) v
      is-c-basic-reference $\circ p(t)$  &  $\neg$ is-<> $\circ s$ -ap-trans-ref(p)

      for:is-non-expl-decl(p)

      Ref.:    trans-ref 4-83(323)

(59) is-file-cont(p) =
      ( $\exists \alpha$ ) ((is-c-FILE $\circ s_1 \circ \alpha(t)$  v is-c-named-io-condition $\circ q(t)$ ) & p = s2 $\circ s_3 \circ \alpha$ ) v
      is-sysin-cont(p) v is-sysprint-cont(p)

      for:is-non-expl-decl(p)

(60) is-sysin-cont(p) =
      is-c-GET $\circ s_1 \circ p(t)$  &  $\neg$ ( $\exists q$ ) ((is-c-FILE $\circ q(t)$  v is-c-STRING $\circ q(t)$ ) & p => q)

(61) is-sysprint-cont(p) =
      is-c-PUT $\circ s_1 \circ p(t)$  &  $\neg$ ( $\exists q$ ) ((is-c-FILE $\circ q(t)$  v is-c-STRING $\circ q(t)$ ) & p => q)

(62) is-task-cont(p) =
      ( $\exists q$ ) (is-c-TASK $\circ s_1 \circ q(t)$  & p = s2 $\circ s_2 \circ s_2 \circ q$  &  $\neg$ is-@p(t))

      for:is-non-expl-decl(p)
```

30 June 1969

TRANSLATION OF PL/I INTO ABSTRACT SYNTAX

```

(63)  is-event-cont(p) =
      (3q) (is-c-EVENT•s1•q(t) & p = s2•s3•q) ∨
            (3q,n) (is-c-wait-statement•q(t) & p = s2•sn•s3•q & ¬is-Q•p(t))

      for:is-non-expl-decl(p)

(64)  is-area-cont(p) =
      (3q) (is-c-IN•s1•q(t) & p = s2•s3•q)

      for:is-non-expl-decl(p)

(65)  is-ptr-cont(p) =
      (3q) (is-c-reference•q(t) & p = s2•s1•s1•q ∨ is-c-SET•s1•q(t) & p = s2•s3•q ∨
            is-c-based-attribute•q(t) & p = s2•s2•s2•q & ¬is-Q•p(t))

      for:is-non-expl-decl(p)

(66)  is-cond-cont(p) =
      (3q) (is-c-programmer-named-condition•q(t) & p = s3•q)

```

4.2.1.5 Attributes specified for a declaration

This section describes the association of explicitly or contextually specified attributes with a declaration. The relation is-attr-of determines for a declaration pointer p its attribute descriptions. These are either the pointers to the explicitly specified attributes in a declaration sentence, or elementary objects denoting attributes implied by the context.

The function is-predec-dens examines a successor sequence in bottom to top order until a density attribute is found, if any. This attribute is then added to the attribute descriptor set belonging to the successor sequence.

```
(67)  is-attr-of(p,ad) =
      is-decl-cont(p) --+
      (Eq) (is-succ-of(p,q)) & is-STRUCT(ad) v (Eq) (is-succ-of(q,p)) & is-SUCC(ad) v
      is-c-attribute-ad(t) &
      (Eq,n) (is-c-declaration-q(t) & q => p & ad = sn*s4*q) v
      is-c-dimension-attribute-ad(t) &
      (Eq) (is-c-declaration-q(t) & q => p & ad = s3*q)

      is-param-cont(p) --+ is-PARAM(ad)
      is-label-cont(p) --+ is-LABEL(ad)
      is-format-cont(p) --+ is-FORMAT(ad)
      is-entry-cont(p) --+ is-ENTRY(ad)
      is-builtin-cont(p) --+ is-BUILTIN(ad)
      is-file-cont(p) --+ is-FILE(ad)
      is-task-cont(p) --+ is-TASK(ad)
      is-event-cont(p) --+ is-EVENT(ad)
      is-area-cont(p) --+ is-AREA(ad)
      is-ptr-cont(p) --+ is-PTR(ad)
      is-cond-cont(p) --+ is-COND(ad)

      Ref.:   is-decl-cont 4-11(36)
              is-succ-of 4-12(42)
              is-param-cont 4-11(37)
              is-label-cont 4-11(38)
              is-format-cont 4-11(39)
              is-entry-cont 4-11(40)
              is-builtin-cont 4-16(58)
              is-file-cont 4-16(59)
              is-task-cont 4-16(62)
              is-event-cont 4-17(63)
              is-area-cont 4-17(64)
              is-ptr-cont 4-17(65)
              is-cond-cont 4-17(66)
```

```
(68)  is-predec-dens(seq,ad) =
      E --+ ad = (Up) (is-attr-of(last(seq),p) & (is-c-ALIGNED v is-c-UNALIGNED)*p(t))
      length(seq) = 1 --+ F
      T --+ is-predec-dens(first(seq),ad)

      for:is-succ-seq(seq)

      Ref.:   is-succ-seq 4-12(41)
```

4.2.1.6 Default attributes

The determination of default attributes is modeled according to the following algorithm:

30 June 1969

TRANSLATION OF PL/I TNTO ABSTRACT SYNTAX

Beginning with an identifier id declared in a block B and the set of attributes declared for id within B, all those default attributes of B which are applicable to id and compatible with all already known attributes of id (e.g., VARYING is compatible with CHARACTER or BIT but not with DECIMAL or FIXED) are added to the set of already known attributes. This process is iterated, until an attribute specification SYSTEM or the outmost block is reached.

A default attribute of a block B is "applicable" to id, if it is explicitly specified for id and compatible with all already known attributes of id, or if it is applicable to the identifier resulting from id when the last (rightmost) character is deleted.

This process is terminated, if a SYSTEM attribute specification is reached or the characters of id are already exhausted.

Not contained in the collection of default attributes are the system defaults. These are inserted, when the abstract objects for the proper data attributes are formed.

```
(69) default(ad-set,b,id) =
      b = I v (3p)(p ∈ block-defaults({},b,id) & is-c-SYSTEM•p(t)) -- def1
      T -- default(ad-set v def1,block-p(b),id)
```

where:

def₁ = def-extension(ad-set,block-defaults({},b,id))

Ref.: block-p 4-4(8)

```
(70) def-extension(ad-set-1,ad-set-2) =
      ad-set-1 u {ad | ad ∈ ad-set-2 & ¬is-c-SYSTEM•ad(t) &
      (Vp)(p ∈ ad-set-1 → is-compat(p,ad))}
```

```
(71) block-defaults(ad-set,b,id) =
      length(chl1) = 1 v (3p)(p ∈ expl-defaults(b,id) & is-c-SYSTEM•p(t)) -- def2
      T -- block-defaults(ad-set v def2,b,mk-id(first(chl1)))
```

where:

def₂ = def-extension(ad-set,expl-defaults(b,id))
chl₁ = (Uchl)(id = mk-id(chl))

Ref.: mk-id 4-90(354)

```
(72) is-compat(ad-1,ad-2) =
      ¬is-incompat(ad-1(t),ad-2(t)) &
      (type-attr-2(ad-1) = type-attr-2(ad-2) v type-attr-2(ad-1) = Ø v
       type-attr-2(ad-2) = Ø) &
      (((is-PROP-VAR v is-BASED v is-DEFINED) • type-attr-2(ad-1) &
       ¬is-Ø•type-attr-2(ad-2)) >
       (da-attr-2(ad-1) = da-attr-2(ad-2) v da-attr-2(ad-1) = Ø v
        da-attr-2(ad-2) = Ø))
```

Ref.: type-attr-2 4-26(92)
da-attr-2 4-32(119)

```

(73)  is-incompat(x,y) =
      is-compl(x,y) ∨ is-compl(y,x) ∨ x ≠ y & s1(x) = s1(y) &
      (is-c-dimension-attribute(x) ∨ s2(x) ≠ 0 ∨ s2(y) ≠ 0)

      for:(is-c-attribute ∨ is-c-dimension-attribute ∨ is-c-value-clause)(x)

(74)  is-compl(x,y) =
      is-c-REAL•s1(x) & is-c-COMPLEX•s1(y) ∨ is-c-DECIMAL•s1(x) & is-c-BINARY•s1(y) ∨
      is-c-FLOAT•s1(x) & is-c-FIXED•s1(y) ∨ is-c-BIT•s1(x) & is-c-CHARACTER•s1(y) ∨
      is-c-AUTOMATIC(x) & (is-c-STATIC(y) ∨ is-c-CONTROLLED(y)) ∨ is-c-STATIC(x) &
      is-c-CONTROLLED(y) ∨ is-c-DEFINED•s1(x) & is-c-POSITION•s1(y) ∨
      is-c-STREAM(x) & is-c-RECORD(y) ∨ is-c-INPUT(x) &
      (is-c-OUTPUT(y) ∨ is-c-UPDATE(y)) ∨ is-c-OUTPUT(x) & is-c-UPDATE(y) ∨
      is-c-SEQUENTIAL(x) & is-c-DIRECT(y) ∨ is-c-BUFFERED(x) & is-c-UNBUFFERED(y) ∨
      is-c-INTERNAL(x) & is-c-EXTERNAL(y) ∨ is-c-file-name-attribute(x) &
      is-c-entry-name-attribute(y)

      for:(is-c-attribute ∨ is-c-dimension-attribute ∨ is-c-value-clause)(x)

(75)  expl-defaults(b,id) =
      {q |
       is-default-attr(q) &
       (∃p)(is-local-to(b,p) &
             (is-c-default-option-1•p(t) & length(chl1) = 1 & s3•p => q ∨
              is-c-default-spec•p(t) & is-in-range-of(s1•p,id) & s2•p => q))}

      where:
      chl1 = {bchl} (id = mk-id(chl))

      Ref.:    is-local-to 4-4(10)
               mk-id 4-90(354)

(76)  is-default-attr(p) =
      (is-c-attribute ∨ is-c-dimension-attribute ∨ is-c-value-clause)•p(t) &
      ¬(∃q)(q => p & is-c-value-clause=q(t)) & def-attr-test(p)

      Note: Since the attributes in the value-clauses require a special treatment the
            pointer to the whole value-clause and not the pointers to the single
            attributes contained in it are collected into the attribute descriptor set.

(77)  def-attr-test(p) =
      ¬is-c-like-attribute•p(t) &
      ((is-c-AREA ∨ is-c-REAL ∨ is-c-COMPLEX ∨ is-c-DECIMAL ∨ is-c-BINARY ∨
        is-c-FLOAT ∨ is-c-FIXED ∨ is-c-BIT ∨ is-c-CHARACTER ∨
        is-c-LABEL)•s1•p(t) => is-0•s2•p(t)) &
      (is-c-INITIAL•s1•p(t) => is-c-CALL•s1•s2•p(t)) -->

      T

      for:(is-c-attribute ∨ is-c-dimension-attribute ∨ is-c-value-clause)•p(t)

```

30 June 1969

TRANSLATION OF PL/I INTO ABSTRACT SYNTAX

(78) is-in-range-of(p,id) =

$$(\exists q_1, q_2) (p \Rightarrow q_1 \Rightarrow q_2 \& \text{is-c-range-spec}(q_1(t)) \&
\text{is-c-identifier}(q_2(t)) \& \text{id} = \text{mk-id-1}(q_2) \vee \text{is-ASTER}(q_2(t)) \&
\text{length}(\text{chl}_1) = 1 \vee \text{is-COLON}(q_2(t)) \& \text{length}(\text{chl}_1) = 1 \&
\text{is-in-letter-range-of}(q_2(t), \text{id}))$$

where:

 $\text{chl}_1 = (\cup \text{chl}) (\text{id} = \text{mk-id}(\text{chl}))$ for: (is-c-range-spec \vee is-c-factored-default-spec) \bullet p(t)Ref.: mk-id-1 4-90 (353)
 mk-id 4-90 (354)

(79) is-in-letter-range-of(obj,id) =

$$\text{isnolet}_1 \cdot s_1(\text{obj}) \vee \text{isnolet}_1 \cdot s_3(\text{obj}) \vee \text{is-Z-CHAR}(s_1(\text{obj})) \& \neg \text{is-Z-CHAR}(s_3(\text{obj})) \rightarrow$$

error

$$\text{id} = \text{gen}_1 \cdot s_1(\text{obj}) \vee \text{id} = \text{gen}_1 \cdot s_3(\text{obj}) \rightarrow T$$

$$T \rightarrow \text{is-in-letter-range-of}(\mu(\text{obj}; \langle s_1 : \text{succ} \cdot s_1(\text{obj}) \rangle), \text{id})$$

where:

 $\text{isnolet}_1 = \text{is-DOLLAR} \vee \text{is-COMM-AT} \vee \text{is-NUMBER-SIGN}$
 $\text{gen}_1 = \text{mk-id} \cdot \text{generate-2} \cdot \text{generate-1}$ Ref.: mk-id 4-90 (354)
 generate-2 2-3 (4)
 generate-1 2-3 (3)

```
(80) succ(char) =
      is-A-CHAR(char) --> B-CHAR
      is-B-CHAR(char) --> C-CHAR
      is-C-CHAR(char) --> D-CHAR
      is-D-CHAR(char) --> E-CHAR
      is-E-CHAR(char) --> F-CHAR
      is-F-CHAR(char) --> G-CHAR
      is-G-CHAR(char) --> H-CHAR
      is-H-CHAR(char) --> I-CHAR
      is-I-CHAR(char) --> J-CHAR
      is-J-CHAR(char) --> K-CHAR
      is-K-CHAR(char) --> L-CHAR
      is-L-CHAR(char) --> M-CHAR
      is-M-CHAR(char) --> N-CHAR
      is-N-CHAR(char) --> O-CHAR
      is-O-CHAR(char) --> P-CHAR
      is-P-CHAR(char) --> Q-CHAR
      is-Q-CHAR(char) --> R-CHAR
      is-R-CHAR(char) --> S-CHAR
      is-S-CHAR(char) --> T-CHAR
      is-T-CHAR(char) --> U-CHAR
      is-U-CHAR(char) --> V-CHAR
      is-V-CHAR(char) --> W-CHAR
      is-W-CHAR(char) --> X-CHAR
      is-X-CHAR(char) --> Y-CHAR
      is-Y-CHAR(char) --> Z-CHAR
```

4.2.2 CONSTRUCTION OF DECLARATIONS

From the preliminary declarations produced as described in the last section the final declarations are constructed according to the abstract syntax. This includes testing of the attribute descriptor set of a preliminary declaration for consistency, determination of system default attributes and insertion of data attributes of the successors into the declaration of a structure.

Many of the functions defined in this section are applied not only to declarations but also to parameter descriptors and allocate items, which have a structure similar to that of declarations.

30 June 1969

TRANSLATION OF PL/I INTO ABSTRACT SYNTAX

(81) $\text{mk-decl}(\text{pd}) =$

$$\begin{aligned} \text{is-PROP-VAR-type-attr}(\text{pd}) &\rightarrow \text{mk-prop-var-decl}(\text{pd}) \\ \text{is-ENTRY-CONST-type-attr}(\text{pd}) &\rightarrow \text{mk-entry-decl}(\text{pd}) \\ \text{is-FILE-CONST-type-attr}(\text{pd}) &\rightarrow \text{mk-file-decl}(\text{pd}) \\ \text{is-LABEL-CONST-type-attr}(\text{pd}) &\rightarrow \text{mk-indexlist}((\text{b}\text{p}) (\langle \text{p} \rangle = \text{pd-seq}(\text{pd})), \text{block-p}(\text{p})) \\ \text{is-DEFINED-type-attr}(\text{pd}) &\rightarrow \text{mk-defined-decl}(\text{pd}) \\ \text{is-BASED-type-attr}(\text{pd}) &\rightarrow \text{mk-based-decl}(\text{pd}) \\ \text{is-FORMAT-type-attr}(\text{pd}) &\rightarrow \text{mk-format-decl}(\text{pd}) \\ \text{is-GENERIC-type-attr}(\text{pd}) &\rightarrow \text{mk-generic-decl}(\text{pd}) \\ \text{is-ATTN-type-attr}(\text{pd}) &\rightarrow \\ &\mu_0 (\langle \text{s-env-attr}: \text{mk-env-attr}(\text{s}_4 * ((\text{bad}) (\text{ad} \in \text{s-attr-ds}(\text{pd}) \& \\ &\quad \text{is-c-ATTENTION}=\text{s}_1 * \text{ad}(\text{t})))) \rangle) \\ \text{T} &\rightarrow \text{type-attr}(\text{pd}) \end{aligned}$$

Ref.: type-attr 4-24(87)
 mk-entry-decl 4-41(166)
 mk-file-decl 4-47(187)
 mk-indexlist 4-6(20)
 pd-seq 4-31(113)
 block-p 4-4(8)
 mk-defined-decl 4-47(190)
 mk-based-decl 4-48(191)
 mk-format-decl 4-48(193)
 mk-generic-decl 4-50(198)

(82) $\text{mk-env-attr}(\text{p}) =$

Note: This predicate is implementation defined, but satisfies the condition:

(83) $\text{is-env-attr} \circ \text{mk-env-attr}(\text{v})$

Ref.: is-env-attr 5-3(18)

(84) $\text{mk-prop-var-decl}(\text{pd}) =$

$$\mu_0 (\langle \text{s-scope:scope-attr}(\text{pd}), \text{s-stg-cl:stg-class-attr}(\text{pd}), \\ \langle \text{s-aggr}: \text{mk-aggr}(\text{pd}, \text{bdp-list}(\text{pd})) \rangle, \text{s-connected:conn-opt}(\text{pd}) \rangle)$$

Ref.: scope-attr 4-26(93)
 stg-class-attr 4-27(98)
 bdp-list 4-28(102)
 conn-opt 4-27(95)

(85) $\text{mk-aggr}(\text{pd}, \text{bdpl}) =$

$$\begin{aligned} \text{is-} \langle \rangle(\text{bdpl}) &\rightarrow \text{non-array-aggr}(\text{pd}) \\ \text{T} &\rightarrow \mu(\text{head}(\text{bdpl}); \langle \text{s-elem}: \text{mk-aggr}(\text{pd}, \text{tail}(\text{bdpl})) \rangle) \end{aligned}$$

(86) non-array-aggr(pd) =
 is-STRUCT•da-attr(pd) --> struct-aggr(pd)
 T --> μ₀(<s-da:mk-da(pd)>, <s-dens:dens-attr(pd)>, <s-init:mk-init(pd)>)

Ref.: da-attr 4-32(117)
 struct-aggr 4-30(110)
 mk-da 4-31(116)
 dens-attr 4-27(96)
 mk-init 4-54(215)

4.2.2.1 Classification of declaration types

There are the following 11 types of declarations. Proper variables, entry-constants, file-constants, defined variables, based variables, builtin functions, generic entries, statement label constants, format labels, programmer-named conditions and attentions. Additionally there is another type of preliminary declarations, namely successors of structures. It is the aim of this section, to classify all preliminary declarations into one of these 12 types and to exclude all those which would lead to more than one of them.

This is performed in three steps:

- (1) For each attribute descriptor the function type-attr-2 determines a "type attribute" giving the type implied by this attribute descriptor. For entry-name-attributes and the file attribute, for which a decision between variable or constant type is not possible without the knowledge of other attributes, the null object Ω is assumed as also for those attributes, e.g., the scope attributes, which allow no unique implication.
- (2) For each preliminary declaration the set of non-Ω type attributes implied by the set of its attribute descriptors is inspected by the function type-attr-1: if there is one type attribute, this one determines the type of the declaration; if there are more than one, it is an error since there are conflicting attributes specified; if there is no one, or if an attribute implying the type variable is specified, PROP-VAR is assumed. Otherwise a decision between entry- or file-constants is made.
- (3) Finally, by the function type-attr, some additional general tests are performed for the consistency of attributes.

(87) type-attr(pd) =
 (Battr)(attr = scope-attr(pd)) &
 (is-data-type•type-attr-1(pd) > (Battr)(attr = dens-attr(pd))) &
 (is-non-data-type•type-attr-1(pd) >
 ~(3ad)(ad ∈ s-attr-ds(pd) & is-data-attr-descr(ad))) &
 (is-non-data-type•type-attr-1(pd) > is-<>•mk-init(pd)) -->
 type-attr-1(pd)

Ref.: scope-attr 4-26(93)
 dens-attr 4-27(96)
 mk-init 4-54(215)

(88) is-data-type =
 is-PROP-VAR ∨ is-DEFINED ∨ is-BASED ∨ is-SUCC

30 June 1969

TRANSLATION OF PL/I INTO ABSTRACT SYNTAX

(89) is-non-data-type =
 is-ENTRY-CONST v is-FILE-CONST v is-GENERIC v is-LABEL-CONST v is-FORMAT v
 is-COND v is-ATTN

(90) is-data-attr-descr(ad) =
 is-c-data-attribute-ad(t) v is-c-dimension-attribute-ad(t) v
 ad ∈ {TASK, EVENT, PTR, AREA, STRUCT, SUCC}

(91) type-attr-1(pd) =
 (3ad-1,ad-2) (ad-1 ∈ ad-set₁ & ad-2 ∈ ad-set₁ &
 (is-c-entry-name-attribute-ad-1(t) v is-ENTRY(ad-1)) &
 (is-c-file-name-attribute-ad-2(t) v is-FILE(ad-2))) -->
 error
 E -->
 (battr)
 (3ad) (ad ∈ ad-set₁ & attr = type-attr-2(ad) & ~is-0(attr))
 (3ad) (ad ∈ ad-set₁ & is-data-attr-descr(ad)) --> PROP-VAR
 (3ad) (ad ∈ ad-set₁ & (is-c-entry-name-attribute-ad(t) v is-ENTRY(ad))) -->
 ENTRY-CONST
 (3ad) (ad ∈ ad-set₁ & is-c-FILE-ad(t)) --> FILE-CONST
 T --> PROP-VAR

where:

ad-set₁ = s-attr-ds(pd)

```
(92) type-attr-2(ad) =
      (is-c-storage-class-attribute v is-c-CONNECTED v is-c-SECONDARY)*ad(t) v
      is-PARAM(ad) -->
      PROP-VAR
      (is-c-file-name-attribute & -is-c-FILE)*ad(t) v is-FILE(ad) -- FILE-CONST
      is-c-defined-attribute*ad(t) -- DEFINED
      is-c-based-attribute*ad(t) -- BASED
      is-c-BUILTIN*ad(t) v is-BUILTIN(ad) -- BUILTIN
      is-c-generic-attribute*ad(t) -- GENERIC
      is-LABEL(ad) -- LABEL-CONST
      is-FORMAT(ad) -- FORMAT
      is-COND(ad) -- COND
      is-SUCC(ad) -- SUCC
      is-c-attention-attribute*ad(t) -- ATTN
      T -- Q
```

4.2.2.2 Scope, connectedness, density and storage class

Principally in the same way as the type of a preliminary declaration, the scope (for all declarations), the density (for all data type declarations) and the storage class (for proper variable declarations) are determined. This is performed by the functions scope-attr, dens-attr, stg-cl-attr.

The connected option may for non controlled parameters which are major structures, be specified.

```
(93) scope-attr(pd) =
      PARAM & s-attr-ds(pd) -- PARAM
      E --+
      (battr)
      (3ad) (attr = scope-attr-1(ad) & -is-Q(attr) & ad & s-attr-ds(pd))
      is-FILE-CONST*type-attr-1(pd) -- EXT
      T -- INT
```

Ref.: type-attr-1 4-25(91)

30 June 1969

TRANSLATION OF PL/I INTO ABSTRACT SYNTAX

(94) scope-attr-1(ad) =
 is-c-INTERNAL•ad(t) --> INT
 is-c-EXTERNAL•ad(t) --> EXT
 is-c-AUTOMATIC•ad(t) ∨ is-c-defined-attribute•ad(t) ∨
 is-c-based-attribute•ad(t) ∨ is-c-generic-attribute•ad(t) ∨ is-PARAM(ad) ∨
 is-LABEL(ad) ∨ is-ENTRY(ad) ∨ is-SUCC(ad) ∨ is-FORMAT•ad(t) -->
 INT
 is-c-BUILTIN•ad(t) ∨ is-BUILTIN(ad) ∨ is-COND(ad) --> EXT
 T --> Ω

(95) conn-opt(pd) =
 ~{ad} (ad ∈ s-attr-ds(pd) & is-c-CONNECTED) --> Ω
 PARAM ∈ s-attr-ds(pd) & STRUCT ∈ s-attr-ds(pd) & SUCC ∈ s-attr-ds(pd) &
 ~is-CTL•stg-cl-attr-1(pd) -->
 *
 T --> Ω

(96) dens-attr(pd) =
 ∃ --> {battr} (is-dens-attr(attr, pd))
 {ad} (is-c-string-attribute•ad(t) & ad ∈ s-attr-ds(pd)) --> UNAL
 T --> AL

(97) is-dens-attr(attr, pd) =
 {ad} (ad ∈ s-attr-ds(pd) & is-c-ALIGNED•ad(t)) --> attr = AL
 {ad} (ad ∈ s-attr-ds(pd) & is-c-UNALIGNED•ad(t)) --> attr = UNAL
 T --> F

(98) stg-class-attr(pd) =
 is-PARAM•scope-attr(pd) ∨ is-CTL•stg-cl-attr-1(pd) ∨ is-Ω•stg-cl-attr-1(pd) -->
 stg-cl-attr-1(pd)

(99) stg-cl-attr-1(pd) =
 ∃ --> {battr} (is-stg-cl-attr(attr, pd))
 is-INT•scope-attr(pd) --> AUTO
 is-EXT•scope-attr(pd) --> STATIC
 is-PARAM•scope-attr(pd) --> Ω

(100) is-stg-cl-attr(attr, pd) =
 {ad} (attr = stg-cl-attr-2(ad) & ~is-Ω(attr) & ad ∈ s-attr-ds(pd))

```
(101) stg-cl-attr-2(ad) =
      is-c-AUTOMATIC•ad(t) --> AUTO
      is-c-STATIC•ad(t) --> STATIC
      is-c-CONTROLLED•ad(t) --> CTL
      T --> Ω
```

4.2.2.3 The dimension attribute

The dimension attribute of a preliminary declaration is translated into a list of objects representing the bound pairs of each dimension.

```
(102) bdp-list(pd) =
       $\text{dim}_1 \underset{n=1}{\overset{\text{dim}_1}{\text{LIST}}} \text{env-trans-bdp}(pd, s_n * s_2 * \text{dim-p}(pd))$ 
```

where:
 $\text{dim}_1 = (\neg \text{is-Ω-dim-p}(pd) --> \text{slength} * s_2 * (\text{dim-p}(pd))(t),$
 $(T --> 0)$

```
(103) dim-p(pd) =
      ∃ -- (lbd)(ad ∈ s-attr-ds(pd) & is-c-dimension-attribute•ad(t))
      T --> Ω
```

```
(104) env-trans-bdp(pd,p) =
       $\mu_0 (< s-lbd : \text{env-trans-lbd}(pd,p) >, < s-ubd : ub_1 > )$ 
```

where:
 $ub_1 = (\text{is-ASTER}•p(t) --> *,$
 $T --> \text{mk-refer-expr}(pd, s_2 * p))$

for:is-c-bound-pair•p(t)

```
(105) env-trans-lbd(pd,v) =
      is-ASTER•p(t) --> *
      is-Ω•s_1•p(t) --> mk-const(<1-CHAR>)
      T --> mk-refer-expr(pd, s_1 * s_1 • p)
```

for:is-c-bound-pair•p(t)

Ref.: mk-const 4-87(341)

30 June 1969

TRANSLATION OF PL/I INTO ABSTRACT SYNTAX

```
(106) mk-refer-expr(pd,p) =
      is-PARAM•scope-attr(pd) & ~is-CTL•STG-CLASS-ATTR(pd) &
      ~is-SG-INTG-CONST(ex1) -->
      error
      is-0•S2•P(t) --> ex1
      T --> μ0(<s-expr:ex1,<s-refer:ref1>)
```

where:

ex₁ = env-trans-expr(pd-block-p(pd),S₁•P)
 ref₁ = (is-BASED•TYPE-ATTR•PREL-DECL(head₁) & MK-ID-1(head₁) = head(idl₁) -->
 tail(idl₁))
 head₁ = head•PD-SEQ(pd)
 idl₁ = S-ID-LIST•DECL-REF(pd-block-p(pd),S₃•S₄•P)

for:is-c-refer-expression•P(t)

Ref.: scope-attr 4-26(93)
 STG-CLASS-ATTR 4-27(98)
 ENV-TRANS-EXPR 4-82(320)
 TYPE-ATTR 4-24(87)
 PREL-DECL 4-9(30)
 MK-ID-1 4-90(353)
 PD-SEQ 4-31(113)
 DECL-REF 4-84(326)

Note: The test, that the structure component referred to has no successors and is to the left of the component containing the refer expression, is made by the Interpreter.

```
(107) pd-block-p(pd) =
      is-prel-decl(pd) --> s-block-p(pd)
      T --> block-p•s-p(pd)
```

Ref.: is-prel-decl 4-10(32)
 block-p 4-4(8)

Note: Since by the like attribute expressions may be translated in another block environment than that of their occurrence, the block pointer is needed as an argument for env-trans-bdp. This is the s-block-p component for preliminary declarations and the block determined by the descriptor itself for preliminary descriptors or allocate items.

```
(108) is-sg-intg-const =
      is-intg-const v
      (<s-opr:is-PLUS v is-MINUS>,
       <s-op:is-intg-const>)
```

```
(109) is-intg-const =
      (<s-v:is-intg-val>,
       <s-da:(<s-mode:is-REAL>,
              <s-base:is-DEC>,
              <s-scale:is-FIX>,
              <s-prec:is-intg-val>)>)
```

Ref.: is-intg-val 5-18(166)

4.2.2.4 Structure declarations

The structure data attributes are constructed from a preliminary declaration pd by determining the list of those preliminary declarations which are successors of pd and computing the list of data attributes of them. The successors of a preliminary declaration are determined by applying the inverse function of prel-decl, going to the successors and applying prel-decl again. The analogous way is valid for parameter descriptors and allocate items.

```
(110) struct-aggr(pd) =
      lg1
      LIST mk-succ-elem(n,succ-list(pd))
      n=1
```

where:
 $lg_1 = \text{length} \cdot \text{succ-list}(pd)$

```
(111) mk-succ(pd) =
      is-SUCC-type-attr(pd) -->
      μ0(<s-qual:last-s-id-list(pd)>, <s-aggr:mk-aggr(pd,bdp-list(pd))>)
```

Ref.: type-attr 4-24(87)
 mk-aggr 4-23(85)
 bdp-list 4-28(102)

```
(112) succ-list(pd) =
      is-prel-decl(pd) -->
      lg1
      LIST prel-decl(ps1^<elem(n,succ-p-list-last(ps1))>)
      n=1
      T -->
      LIST prel-descr-elem(n,descr-sucess-p-list-s-p(pd))
      n=1
```

where:
 $ps_1 = pd-\text{seq}(pd)$
 $lg_1 = \text{length} \cdot \text{succ-p-list-last}(ps_1)$
 $lg_2 = \text{length} \cdot \text{descr-sucess-p-list-s-p}(pd)$

Ref.: is-prel-decl 4-10(32)
 prel-decl 4-9(30)
 prel-descr 4-57(226)

30 June 1969

TRANSLATION OF PL/I INTO ABSTRACT SYNTAX

(113) pd-seq(pd) =
 (Useq) (is-succ-seq(seq) & is-decl-cont•head(seq) & pd = prel-decl(seq))

Ref.: is-succ-seq 4-12(41)
 is-decl-cont 4-11(36)
 prel-decl 4-9(30)

Note: This function may be considered as the inverse function of prel-decl, yielding the successor sequence belonging to a preliminary declaration.

(114) succ-p-list(p) =
 collect({q | is-succ-of(p,q)})

Ref.: is-succ-of 4-12(42)

(115) descr-succ-p-list(p) =
 collect({q | is-descr-succ-of(p,q)})

Ref.: is-descr-succ-of 4-58(232)

4.2.2.5 Classification of data attributes

The scalar data attributes of a variable declaration are compiled by the function mk-da. This function is defined by case distinction according to the following classes of data attributes: arithmetic, string, picture, area, label, entry, file, task, event, pointer, offset.

This classification is performed by the function da-attr, which is defined principally in the same way as the function type-attr. It yields elementary objects denoting the individual classes of data attributes.

(116) mk-da(pd) =
 is-ARITHM•da-attr(pd) -- arithm-da(pd)
 is-STRING•da-attr(pd) -- string-da(pd)
 is-PIC•da-attr(pd) -- pic-da(pd)
 is-AREA•da-attr(pd) -- area-da(pd)
 is-LABEL•da-attr(pd) -- label-da(pd)
 is-ENTRY•da-attr(pd) -- entry-da(pd)
 is-OFFSET•da-attr(pd) -- offset-da(pd)
 T -- da-attr(pd)

Ref.: arithm-da 4-33(123)
 string-da 4-35(139)
 pic-da 4-37(146)
 area-da 4-40(161)
 label-da 4-41(164)

TRANSLATION OF PL/I INTO ABSTRACT SYNTAX

30 June 1969

(117) da-attr(pd) =
 ((?ad) (ad ∈ s-attr-ds(pd) & is-c-COMPLEX•s₁•ad(t)) → is-ARITHM•da-attr-1(pd) ∨
 is-PIC•da-attr-1(pd)) &
 ((is-STRUCT•da-attr-1(pd) ∨ is-PARAM•scope-attr(pd) &
 ~is-CTL•stg-class-attr(pd)) → is-<>•mk-init(pd)) -->
 da-attr-1(pd)

Ref.: scope-attr 4-26(93)
 stg-class-attr 4-27(98)
 mk-init 4-54(215)

Note: The attribute COMPLEX does not necessarily imply arithmetic, it may also be given to a picture declaration.

(118) da-attr-1(pd) =
 ? -->
 (Lattr)
 (?ad) (attr = da-attr-2(ad) & ~is-Ω(attr) & ad ∈ s-attr-ds(pd))
 T --> ARITHM

(119) da-attr-2(ad) =
 is-STRUCT(ad) --> STRUCT
 is-c-arithmetic-attribute•ad(t) & ~is-c-COMPLEX•s₁•ad(t) --> ARITHM
 is-c-string-attribute•ad(t) ∨ is-c-VARYING•ad(t) --> STRING
 is-c-picture-attribute•ad(t) --> PIC
 is-c-area-attribute•ad(t) ∨ is-AREA(ad) --> AREA
 is-c-label-attribute•ad(t) --> LABEL
 is-c-entry-name-attribute•ad(t) --> ENTRY
 is-c-FILE•ad(t) --> FILE
 is-c-TASK•ad(t) ∨ is-TASK(ad) --> TASK
 is-c-EVENT•ad(t) ∨ is-EVENT(ad) --> EVENT
 is-c-POINTER•ad(t) ∨ is-PTR(ad) --> PTR
 is-c-offset-attribute•ad(t) --> OFFSET
 T --> Ω

(120) entry-da(pd) =
 $\mu_0 (<\text{s-descr-list:mk-descr-list-1(pd)}, <\text{s-ret-type:mk-ret-type-1(pd)},$
 $<\text{s-reducible:mk-red(pd)}>)$

Ref.: mk-descr-list-1 4-42(169)
 mk-ret-type-1 4-45(183)
 mk-red 4-46(196)

30 June 1969

TRANSLATION OF PL/I INTO ABSTRACT SYNTAX

(121) offset-da(pd) =
 is-Q*s₂*off-p₁(t) --> OFFSET
 T --> μ₀(<s-area:env-trans-ref(s-block-p(pd), s₂*s₃*off-p₁)>)

where:

off-p₁ = offset-attr-p(pd)

Ref.: env-trans-ref 4-83(324)

(122) offset-attr-p(pd) =
 (bad) (ad ∈ s-attr-ds(pd) & is-c-offset-attribute*ad(t))

4.2.2.6 Arithmetic data attributes

This section defines the construction of scalar arithmetic data attributes from the attribute descriptor set of a preliminary declaration. This includes testing of the arithmetic data attributes for consistency and insertion of system default attributes. If no arithmetic data attributes are specified at all, the default attributes depend on the first letter of the declared identifier.

The default precision and the default scale factor are either taken from a value clause of the attribute descriptor set with matching mode, base and scale or the implementation defined values are inserted.

(123) arithm-da(pd) =
 μ₀(<s-mode:arithm-mode(pd)>, <s-base:arithm-base(pd)>, <s-scale:arithm-scale(pd)>, <s-prec:arithm-prec(pd)>, <s-scale-f:arithm-scale-f(pd)>)

(124) arithm-mode(pd) =
 ~(3ad) (ad ∈ s-attr-ds(pd) & is-c-COMPLEX*s₁*ad(t)) --> REAL
 ~(3ad) (ad ∈ s-attr-ds(pd) & is-c-REAL*s₁*ad(t)) --> CPLX

(125) arithm-base(pd) =
 ~(3ad) (ad ∈ s-attr-ds(pd) & is-c-BINARY*s₁*ad(t)) & is-dec-flt-default(pd) -->
 DEC
 ~(3ad) (ad ∈ s-attr-ds(pd) & is-c-DECIMAL*s₁*ad(t)) --> BIN

(126) arithm-scale(pd) =
 ~(3ad) (ad ∈ s-attr-ds(pd) & is-c-FIXED*s₁*ad(t)) & is-Q*s₃*prec-p(pd) &
 is-dec-flt-default(pd) -->
 FLT
 ~(3ad) (ad ∈ s-attr-ds(pd) & is-c-FLOAT*s₁*ad(t)) --> FIX

TRANSLATION OF PL/I INTO ABSTRACT SYNTAX

30 June 1969

(127) arithm-prec(pd) =
 ~is-Ω•prec-p(pd) --> const-val•s₂•prec-p(pd)
 E --> const-val(s₂•((Up)(is-def-prec(p, pd))))
 T --> def-prec-1(arithm-base(pd), arithm-scale(pd))

Ref.: const-val 4-89(347)

(128) prec-p(pd) =
 E -->
 s₂•((bad)(ad ∈ s-attr-ds(pd) & is-c-arithmetic-attribute•ad(t) &
 ~is-Ω•s₂•ad(t)))
 T --> Ω

(129) is-def-prec(p, pd) =
 E -->
 p = s₂•((bad,_n)(ad ∈ s-attr-ds(pd) & is-c-value-clause•ad(t) &
 is-c-precision-spec•s_n•s₃•ad(t) & s_n•s₃•ad => q &
 is-c-arithmetic-attribute•q(t) & ~is-Ω•s₂•q(t) &
 is-data-matching(s_n•s₃•ad, pd)))
 T --> F

(130) is-data-matching(p, pd) =
 (3q)(p => q & is-c-COMPLEX•s₁•q(t)) =
 c-mode(pd) = s₁•((bad)(p => q & (is-c-COMPLEX v is-c-REAL)•s₁•q(t)))(t) &
 c-base(pd) = s₁•((bad)(p => q & (is-c-DECIMAL v is-c-BINARY)•s₁•q(t)))(t) &
 c-scale(pd) = s₁•((bad)(p => q & (is-c-FLOAT v is-c-FIXED)•s₁•q(t)))(t)
 for:is-c-precision-spec•p(t)

(131) c-mode(pd) =
 is-REAL•arithm-mode(pd) --> (Ux)(is-c-REAL(x))
 T --> (Ux)(is-c-COMPLEX(x))

(132) c-base(pd) =
 is-DEC•arithm-base(pd) --> (Ux)(is-c-DECIMAL(x))
 T --> (Ux)(is-c-BINARY(x))

(133) c-scale(pd) =
 is-FLT•arithm-scale(pd) --> (Ux)(is-c-FLOAT(x))
 T --> (Ux)(is-c-FIXED(x))

30 June 1969

TRANSLATION OF PL/I INTO ABSTRACT SYNTAX

```
(134) arithm-scale-f(pd) =
      ~is-0*s3*prec-p(pd) -- const-val*s2*s3*prec-p(pd)
      E -- const-val(s2*s3*((Up)(is-def-prec(p,pd) & ~is-0*s2*s3*p(t))))
      is-FLT*arithm-scale(pd) -- 0
      T -- 0
```

Ref.: const-val 4-89(347)

(135) def-prec-1(base,scale) =

Note: This function yields four implementation defined integer values.

(136) is-dec-flt-default(pd) =

$$(\exists ad) (ad \in s\text{-attr-ds}(pd) \& is-c-arithmetic-attribute*ad(t)) \vee$$
$$is-0*s\text{-id-list}(pd) \vee \neg is-i-to-n*first-letter*last*s\text{-id-list}(pd)$$

(137) first-letter(id) =

head((Uch1)(id = mk-id(ch1)))

Ref.: mk-id 4-90(354)

(138) is-i-to-n(ch) =

ch ∈ {I-CHAR, J-CHAR, K-CHAR, L-CHAR, M-CHAR, N-CHAR}

4.2.2.7 String data attributes

Similarly as the scalar arithmetic data attributes, the scalar string data attributes are constructed in this section. The default string length is compiled from a value clause of the attribute descriptor set with matching string-base or if no value clause is applicable the length 1 is inserted.

(139) string-da(pd) =

$$\mu_0(<s\text{-base:string-base}(pd), s\text{-length:string-length}(pd),$$

$$<s\text{-varying:varying-attr}(pd)>)$$

(140) string-base(pd) =

$$(\exists ad) (is-string-attr-p(ad,pd) \& is-c-BIT*s₁*ad(t)) \&$$
$$(\exists ad) (is-string-attr-p(ad,pd) \& is-c-CHARACTER*s₁*ad(t)) --$$

error

$$(\exists ad) (is-string-attr-p(ad,pd) \& is-c-CHARACTER*s₁*ad(t)) -- CHAR$$

T -- BIT

(141) is-string-attr-p(ad,pd) =

ad ∈ s-attr-ds(pd) & is-c-string-attribute*ad(t)

(142) string-length(pd) =

$\exists \rightarrow$

$\text{mk-refer-expr}(\text{pd}, s_2 * s_2 * ((\exists \text{ad}) (\text{is-string-attr-p}(\text{ad}, \text{pd}) \& \neg \text{is-Q} * s_2 * s_2 * \text{ad}(t))))$

$\exists \rightarrow$

$\text{mk-refer-expr}(\text{pd-block-p}(\text{pd}), (\exists p) (\text{is-def-string-length}(p, \text{pd}) \& \neg \text{is-Q} * p(t)))$

$T \rightarrow \text{mk-const}(<1-CHAR>)$

Ref.: mk-refer-expr 4-29(106)
 pd-block-p 4-29(107)
 mk-const 4-87(341)

Note: <1-CHAR> satisfies the predicate is-c-integer.

(143) is-def-string-length(p, pd) =

$\exists \rightarrow$

$p = s_2 * s_2 * ((\exists q)$
 $(\exists \text{ad}, n) (\text{ad} \in \text{s-attr-ds}(\text{pd}) \& \text{is-c-value-clause-ad}(t) \& q = s_n * s_3 * \text{ad} \&$
 $\text{is-c-string-attribute-q}(t) \& \text{is-string-matching}(q, \text{pd}))$

$T \rightarrow F$

(144) is-string-matching(p, pd) =

$\text{is-CHAR-string-base}(\text{pd}) \& \text{is-c-CHARACTER} * s_1 * q(t) \vee \text{is-BIT-string-base}(\text{pd}) \&$
 $\text{is-c-BIT} * s_1 * q(t) \rightarrow$

test₁

$T \rightarrow F$

where:

test₁ = ($\text{is-Q} * s_2 * s_2 * s_2 * p(t) \rightarrow T$)

for: is-c-string-attribute-p(t)

(145) varying-attr(pd) =

$(\exists \text{ad}) (\text{ad} \in \text{s-attr-ds}(\text{pd}) \& \text{is-c-VARYING-ad}(t)) \rightarrow \text{VAR}$

$T \rightarrow Q$

4.2.2.8 Picture data attributes

This section defines the translation of a picture specification (is-c-picture-specification) together with its mode into a data attribute satisfying the predicate is-pic. For picture specifications which turn out to be of sterling or character type, the mode serves for checking purposes only. The translation is erroneous whenever it is impossible to map the picture specification and the mode into a data attribute satisfying is-pic. However, the existence of the translation is only a necessary prerequisite for a picture

30 June 1969

TRANSLATION OF PL/I INTO ABSTRACT SYNTAX

specification which also has an interpretation, i.e. which is a proper picture attribute.

Metavariables

c-pic-spec	is-c-picture-specification	picture specification
pic	is-c-picture-string v is-Ω	picture specification without scaling factor
pcl	is-c-picture-character-list	list of picture characters
el	is-c-picture-character	picture character which actually may be composed of two characters
sf,i,j,k,n	is-intg-val v is-Ω	scaling factor, integers
scf		scaling factor in list form
mode	is-CPLX v is-REAL	complex or real mode
list	is-list v is-Ω	list

(146) pic-da(pd) =
 trans-pic(s₂ • {(bad) (ad ∈ s-attr-ds(pd) &
 is-c-picture-attribute=ad(t))} (t), pic-mode(pd))

(147) pic-mode(pd) =
 ~ (bad) (ad ∈ s-attr-ds(pd) & is-c-COMPLEX•s₁•ad(t)) -- Ω
 is-Ω•prec-p(pd) -- CPLX

Ref.: prec-p 4-34(128)

(148) trans-pic(c-pic-spec, mode) =
 is-pic(tp₁) -- tp₁

where:

tp₁ =
 trans-pic-1(expand-pic-spec•elem(2,c-pic-spec), scale-f•elem(3,c-pic-spec), mode)

Ref.: is-pic 5-7(66)

```
(149) expand-pic-spec(pic) =
      is-Ω(pic) --> <>
      T --> (  $\overbrace{\text{LIST}}^{\text{rep}_1}$  elem(2,pic)) * expand-pic-spec*elem(3,pic)
            n=1
```

where:

$$\begin{aligned} \text{rep}_1 &= (\text{is-Ω*elem}(1,\text{pic}) --> 1, \\ &\quad \text{T} --> \text{const-val*elem}_2\text{*elem}(1,\text{pic})) \end{aligned}$$

Ref.: const-val 4-89(347)

Note: The effect of iteration factors is the reason for the few constraints on picture specifications as given by the concrete syntax. For instance, a picture specification '(0)G(0)A(4)9' is treated as a correct decimal picture specification after expansion of iteration factors.

```
(150) scale-f(scf) =
      is-Ω(scf) --> Ω
      T --> sgn*elem(3,scf) . const-val*elem(4,scf)
```

Ref.: sgn 4-89(350)
const-val 4-89(347)

```
(151) trans-pic-1(pcl,sf,mode) =
      (A-CHAR ∈ elem-set(pcl) ∨ X-CHAR ∈ elem-set(pcl)) & is-Ω(sf) & is-Ω(mode) -->
      μ0(<s-field:pcl>)
      T --> μ(trans-pic-2(pcl,sf);<s-mode:mode>,<s-scale-f:sf>)
```

Note: Potential character pictures are treated by the first alternative.

```
(152) elem-set(list) =
      {el | ¬is-Ω(el) & (∀n)(el = elem(n,list))}
```

```
(153) pos-el(el,list) =
      el ∈ elem-set(list) --> (∀n)(el = elem(n,list))
      T --> Ω
```

```
(154) trans-pic-2(pcl,sf) =
      is-G-CHAR*head(pcl) --> trans-sterling-pic*tail(pcl)
      M-CHAR ∉ elem-set(pcl) --> trans-num-pic(pcl,sf)
```

Note: Potential sterling pictures are treated by the first alternative. The occurrence of the character M in another picture than a potential sterling picture is an error.

30 June 1969

TRANSLATION OF PL/I INTO ABSTRACT SYNTAX

(155) trans-sterling-pic(pcl) =
 $\mu_0(\langle s\text{-mt-field:trans-pcl(pcl)}, \langle s\text{-stat-part-end:elem}_1\text{*m-pos(pcl)} - 1\rangle,$
 $\langle s\text{-pound-end:elem}_2\text{*m-pos(pcl)} - 2\rangle, \langle s\text{-shill-end:elem}_3\text{*m-pos(pcl)} - 3\rangle,$
 $\langle s\text{-mt-unit:pos-v(pcl,4)}\rangle)$

(156) pos-v(pcl,n) =
 $is-\Omega\text{*pos-el(V-CHAR,pcl)} \rightarrow \Omega$
 $T \rightarrow pos-el(V-CHAR,pcl) - n$

(157) m-pos(pcl) =
 $(\exists list)$
 $(\exists i,j,k) (list = \langle i,j,k \rangle \& i < j < k \&$
 $elem-set(list) = \{n \mid is-M-CHAR\text{*elem}(n,pcl)\})$

Note: A potential sterling picture must contain exactly three characters M,
otherwise it is erroneous.

(158) trans-num-pic(pcl,sf) =
 $is-\Omega(pos_1) \rightarrow \mu_0(\langle s\text{-mt-field:trans-pcl(pcl)}, \langle s\text{-mt-unit:pos-v(pcl,1)}\rangle)$
 $is-\Omega(sf) \rightarrow$
 $\mu_0(\langle s\text{-mt-field:trans-pcl(pcl_1)}, \langle s\text{-mt-unit:pos-v(pcl,1)}\rangle,$
 $\langle s\text{-exp-field:trans-pcl(pcl_2)}, \langle s\text{-exp-sep:exp_1}\rangle)$

where:

$pos_1 = (\exists n) (n = pos-el(K-CHAR,pcl) \vee n = pos-el(E-CHAR,pcl))$
 $pcl_1 = LIST elem(n,pcl) \quad n=1 \quad length(pcl)$
 $pcl_2 = LIST elem(n,pcl) \quad n=pos_1+1$
 $exp_1 = (is-E-CHAR\text{*elem}(pos_1,pcl) \rightarrow E-CHAR,$
 $T \rightarrow \Omega)$

Note: A scaling factor with a floating point picture is erroneous.

(159) trans-pcl(pcl) =
 $length(pcl)$
 $CONC trans-pic-el\text{*elem}(n,pcl)$
 $n=1$

```
(160) trans-pic-el(el) =
      is-S-CHAR(el) --> <SIGN>
      is-H-CHAR(el) --> <S-CHAR>
      is-P-CHAR(el) --> <D-CHAR>
      is-M-CHAR(el) v is-V-CHAR(el) --> <>
      T --> <el>
```

Note: The mapping of the character M into the empty list requires the preceding test made in trans-pic-2.

4.2.2.9 Area attribute

The area attribute is translated nearly in the same way as the string attributes. If no area size is specified and also no value-clause leads to an area size, the implementation defined value DEF-SIZE is assumed.

```
(161) area-da(pd) =
      E -->
      μo(<s-size:mk-refer-expr(pd-block-p(pd),
                                s2*s2*((bad)(is-decl-area-attr(ad,nd))))>
      E --> μo(<s-size:mk-refer-expr(pd, (tp) (is-def-area-size(p, pd) & -is-Q*p(t)))>
      T --> μo(<s-size:mk-const(DEF-SIZE)>)
```

Ref.: mk-refer-expr 4-29(106)
 pd-block-p 4-29(107)
 mk-const 4-87(341)

Note: DEF-SIZE is an implementation defined object satisfying the predicate is-c-integer.

```
(162) is-decl-area-attr(ad, pd) =
      ad ∈ s-attr-ds(pd) & is-c-area-attribute-ad(t) & -is-Q*s2*s2*ad(t)
```

```
(163) is-def-area-size(p, pd) =
      E -->
      p = s2*s2*((q)
      (bad, n) (ad ∈ s-attr-ds(pd) & is-c-value-clause-ad(t) & q = sn*s1*ad &
      is-c-area-attribute-q(t) & test1))
      T --> F
```

where:
 test₁ = (is-Q*s₂*s₂*s₂*q(t) --> T)

30 June 1969

TRANSLATION OF PL/I INTO ABSTRACT SYNTAX

4.2.2.10 Label attribute

If a label variable is declared without a restricting label constant list, or if it is a parameter descriptor, the corresponding data attribute in the abstract program is just the elementary object LABEL. Otherwise, the restricting label constant list is inserted.

(164) $\text{label-da}(\text{pd}) =$
 $(\exists \text{ad}) (\text{is-label-attr-p}(\text{ad}, \text{pd}) \wedge \text{is-Q-s_2-ad(t)}) \vee \text{is-Q-s-block-p}(\text{pd}) \rightarrow \text{LABEL}$
 $\text{slength-label}_1(\text{t})$
 $\text{T} \rightarrow \mu_0(\langle \text{s-label-list}: \text{LIST } \underset{n=1}{\text{mk-id-ref}}(\text{s}_n \cdot \text{label}_1) \rangle)$

where:

$\text{label}_1 = \text{s}_2 \cdot \text{s}_2 \cdot \{ (\exists \text{ad}) (\text{is-label-attr-p}(\text{ad}, \text{pd})) \}$

Ref.: mk-id-ref 4-6(19)

(165) $\text{is-label-attr-p}(\text{ad}, \text{pd}) =$
 $\text{ad} \in \text{s-attr-ds}(\text{pd}) \wedge \text{is-c-label-attribute-ad(t)}$

4.2.2.11 Entry declaration

Analogously as by the function prel-decl-set-1(b) all level-one preliminary declarations belonging to the scope of the block pointed to by b are collected, in section 4.2.3 by the function prel-descr-list(p) all level-one preliminary descriptors belonging to the entry attribute (or allocate statement) pointed to by p are collected. These preliminary descriptors have a very similar form to the preliminary declarations, so that they are handled by most of the functions defined in section 4.2.2 as well.

The translation of an entry declaration mainly consists of describing the parameter descriptors which are constructed from the list of level-one preliminary descriptors of the entry attribute (if there is one) similar to the construction of declarations.

A return type from an entry declaration without returns attribute is taken from the entry point for which the entry is declared. If also here a returns attribute is missing, default attributes from the entry name are derived. In all cases objects similarly to preliminary declarations are constructed as a basis for the translation.

The body name component of an entry declaration is the abstract representation of the first entry name of the procedure for which the entry is declared. It serves as a connection to the body to which the entry belongs.

(166) $\text{mk-entry-decl}(\text{pd}) =$
 $\mu_0(\langle \text{s-scope:scope-attr}(\text{pd}) \rangle, \langle \text{s-descr-list:mk-descr-list}(\text{pd}) \rangle,$
 $\langle \text{s-ret-type:mk-ret-type}(\text{pd}) \rangle, \langle \text{s-body:mk-body-name}(\text{pd}) \rangle,$
 $\langle \text{s-reducible:mk-red}(\text{pd}) \rangle)$

Ref.: scope-attr 4-26(93)

```
(167) mk-descr-list(pd) =
      is-EXT*scope-attr(pd) & s-block-p(pd) * I v is-data-type*type-attr(pd) --*
      mk-descr-list-1(pd)
      T --> mk-descr-list-2(pd)
```

Ref.: scope-attr 4-26(93)
 is-data-type 4-24(88)
 type-attr 4-24(87)

```
(168) mk-descr-list-1(pd) =
      ~ (3ad) (ad ∈ s-attr-ds(pd) & is-c-ENTRY*s1*ad(t) & ~is-0*s2*ad(t)) -- *
      length(prel1)
      T --> LISTn=1 mk-descr-1*elem(n, prel1)
```

where:
 prel₁ = prel-descr-list*entry-attr-p(pd)

Ref.: prel-descr-list 4-56(222)

```
(169) entry-attr-p(pd) =
      (bad) (ad ∈ s-attr-ds(pd) & is-c-ENTRY*s1*ad(t))
```

```
(170) mk-descr-1(pd) =
      is-*(pd) -- *
      T --> mk-descr(Ω, pd)
```

```
(171) mk-descr-list-2(pd) =
      ~ (3ad) (ad ∈ s-attr-ds(pd) & is-c-ENTRY*s1*ad(t) & ~is-0*s2*ad(t)) -- *
      slength(par1)
      LISTn=1 mk-descr(sn*par1, *)
      slength(par1) = slength*s2*s2(ent1) --
      slength(par1)
      LISTn=1 mk-descr(sn*par1, elem(n, prel-descr-list(ent1)))
```

where:
 ent₁ = entry-attr-p(pd)
 par₁ = s₂*mk-par-p*entry-p(pd)

Ref.: prel-descr-list 4-56(222)

30 June 1969

TRANSLATION OF PL/I INTO ABSTRACT SYNTAX

(172) **mk-descr(p,pd) =**

$$\begin{aligned} & \text{is-*}(pd) \rightarrow \text{mk-descr}(\Omega, \text{prel-decl}(p)) \\ & T \rightarrow \\ & \mu_0(<\text{s-stg-cl:stg-class-attr}(pd), \text{s-aggr:mk-descr-aggr}(pd, \text{bdp-list}(pd))>, \\ & \quad <\text{s-connected:conn-opt}(pd)>) \\ & \text{for:is-}\Omega(p) \vee \text{is-c-identifier*}p(t) \end{aligned}$$

Ref.: prel-decl 4-9(30)
 stg-class-attr 4-27(98)
 bdp-list 4-28(102)
 conn-opt 4-27(95)

(173) **entry-p(pd) =**
 $\{\cup_0\} (\text{is-entry-cont}(p) \wedge pd = \text{prel-decl}(p))$

Ref.: is-entry-cont 4-11(40)
 prel-decl 4-9(30)

(174) **mk-par-p(p) =**
 $\begin{aligned} & S \rightarrow s_4 \cdot (\{\cup_0\} (\text{is-c-procedure*}q(t) \wedge s_2 \cdot \sigma \Rightarrow p)) \\ & T \rightarrow s_3 \cdot (\{\cup_0\} (\text{is-c-entry*}q(t) \wedge s_1 \cdot \sigma \Rightarrow p)) \end{aligned}$

(175) **mk-descr-aggr(pd,bdp1) =**
 $\begin{aligned} & \text{is-}\leftrightarrow(bdp1) \rightarrow \text{mk-descr-non-array}(pd) \\ & T \rightarrow \\ & \mu_0(<\text{s-lbd:descr-ext}(pd, s-lbd \cdot \text{head}(bdp1)), \\ & \quad <\text{s-ubd:descr-ext}(pd, s-ubd \cdot \text{head}(bdp1)), \\ & \quad <\text{s-elem:mk-descr-aggr}(pd, \text{tail}(bdp1))>) \end{aligned}$

(176) **descr-ext(pd,expr) =**
 $\begin{aligned} & \text{is-*}(expr) \vee \text{is-}\Omega\text{-s-p}(pd) \wedge \text{is-CTL*stg-class-attr}(pd) \wedge \\ & \neg \text{is-sg-intg-const}(expr) \rightarrow \\ & * \\ & \text{is-sg-intg-const}(expr) \rightarrow \text{sgn}\text{-s-opr}(expr) \cdot \text{sel}_1(expr) \end{aligned}$

where:
 $\text{sel}_1 = (\neg \text{is-}\Omega\text{-s-opr}(expr) \rightarrow \text{s-v}\text{-s-op},$
 $T \rightarrow \text{s-v})$

for:is-expr(expr)

Ref.: stg-class-attr 4-27(98)
 is-sg-intg-const 4-29(108)
 sgn 4-89(350)
 is-expr 5-17(151)

Note: Since the extents are computed by functions valid only for non entry variables a special test for the necessary restrictions is to be made.

(177) **mk-descr-non-array(pd) =**

```
length-spec-list(pd)
is-STRUCT*da-attr(pd) --> LIST  $\mu_0(<s\text{-aggr:mk\text{-descr\text{-aggr}(pd_1,bdp\text{-list}(pd_1))}>)$ 
n=1
```

T --> mk-descr-scalar(pd)

where:

pd₁ = elem(n,succ-list(pd))

Ref.: da-attr 4-32(117)
bdp-list 4-28(102)
succ-list 4-30(112)

(178) **mk-descr-scalar(pd) =**

```
 $\mu_0(<s\text{-da:mk\text{-descr\text{-da}(pd)}>,<s\text{-dens:dens\text{-attr}(pd)}>)$ 
```

Ref.: dens-attr 4-27(96)

(179) **mk-descr-da(pd) =**

```
is-STRING(da1) --> descr-string-da(pd)
is-AREA(da1) --> descr-area-da(pd)
is-LABEL(da1) --> LABEL
is-OFFSET(da1) & is-0*s-p(pd) &
(3p)(is-c-reference-p(t) & offset-attr-p(pd) => p &
bl1 = s-block-p-decl-ref(bl1,p)) -->
OFFSET
T --> mk-da(pd)
```

where:

bl₁ = s-block-p(pd)
da₁ = da-attr(pd)

Ref.: offset-attr-p 4-33(122)
decl-ref 4-84(326)
mk-da 4-31(116)
da-attr 4-32(117)

Note: A reference to an area of an offset returns attribute of an entry point is not overtaken to the entry declaration with missing returns attribute, when the area is not declared in the outer block.

30 June 1969

TRANSLATION OF PL/I INTO ABSTRACT SYNTAX

```
(180) descr-string-da(pd) =
       $\mu(str_1; <s\text{-length}:descr-ext(pd, s\text{-length}(str_1))>)$ 

      where:
      str_1 = string-da(pd)

      Ref.: string-da 4-35(139)

(181) descr-area-da(pd) =
       $\mu_0(<s\text{-size}:descr-ext(pd, s\text{-size}\text{*}area-da(pd))>)$ 

      Ref.: area-da 4-40(161)

(182) mk-ret-type(pd) =
      is-EXTERNAL=scope-attr(pd) & s-block-p(pd) * I v is-PROP-VAR-type-attr(pd) ++
      mk-ret-type-1(pd)

      ~ (3ad) (ad ∈ s-attr-ds(pd) & is-c-RETURNS•s_1•ad(t)) ++
      s-ret-type•(mk-id-1(entry-p(pd)))•s-entry-pt•(mk-body-name(pd))•mk-body-pt•
      s-block-p(pd)

      T ++ mk-ret-type-2(pd)

      Ref.: scope-attr 4-26(93)
            type-attr 4-24(87)
            mk-id-1 4-90(353)
            mk-body-pt 4-5(15)

(183) mk-ret-type-1(pd) =
      ~ (3ad) (ad ∈ s-attr-ds(pd) & is-c-RETURNS•s_1•ad(t)) ++ mk-descr-scalar(pd_1)
      T ++ mk-ret-type-2(pd)

      where:
      pd_1 =  $\mu(pd; <s\text{-attr-ds}:default(\{\}, s\text{-block-p}(pd), def-name\text{*}last\text{*}s\text{-id-list}(pd))>)$ 

      Ref.: default 4-19(69)
            def-name 4-10(34)
```

TRANSLATION OF PL/I INTO ABSTRACT SYNTAX

30 June 1969

(184) $\text{mk-ret-type-2}(\text{pd}) =$

```

is-ENTRY•da-attr(pd1) &
~(Bad) (ad ∈ s-attr-ds(pd1) & is-c-returns-attribute•ad(t)) -->
μ(descr1; <s-ret-type•s-da:descr2>)

T --> descr1

```

where:

 $\begin{aligned}
pd_1 &= \mu(\text{pd}; \\
&\quad <\text{s-attr-ds:ad} \mid \text{is-PARAM(ad)} \vee \\
&\quad (\exists n \neg \text{is-Q•ad}(t) \wedge ad = s_n \cdot s_3 \cdot \text{ret}_1(t))\}>) \\
\text{ret}_1 &= (\text{bad}) (\text{ad} \in \text{s-attr-ds}(\text{pd}) \wedge \text{is-c-returns-attribute•ad}(t)) \\
\text{descr}_1 &= \text{mk-descr-scalar}(pd_1) \\
\text{descr}_2 &= \mu_0(<\text{s-da:ar}_1>, <\text{s-dens:AL}>) \\
\text{ar}_1 &= \mu_0(<\text{s-mode:REAL}>, <\text{s-base:DEC}>, <\text{s-scale:FLT}>, <\text{s-prec:DEF-PREC-DEC}>)
\end{aligned}$

Ref.: da-attr 4-32(117)

Note: DEF-PREC-DEC is an implementation defined integer value.

(185) $\text{mk-body-name}(\text{pd}) =$

```

is-EXT•scope-attr(pd) & s-block-p(pd) * T --> Q
T --> mk-id-1(s1•s1•s2•proc-p•ent1)

```

where:

 $\text{ent}_1 = (\text{bp}) (\text{is-entry-cont(p)} \wedge \text{pd} = \text{prel-decl(p)})$

Ref.: scope-attr 4-26(93)
mk-id-1 4-90(353)
proc-p 4-4(12)
is-entry-cont 4-11(40)
prel-decl 4-9(30)

(186) $\text{mk-red}(\text{pd}) =$

```

(Bad) (ad ∈ s-attr-ds(pd) & is-c-REDUCIBLE•ad(t)) --> red1
T --> Q

```

where:

 $\text{red}_1 = (\neg \text{Bad}) (\text{ad} \in \text{s-attr-ds} \wedge \text{is-c-IRREDUCIBLE•ad}(t)) --> *$
4.2.2.12 File declaration

The computation of a file variable consists mainly of a simple collecting of all explicitly declared file attributes. The set of declared file attributes is augmented by the file attributes implied by them in the Interpreter.

30 June 1969

TRANSLATION OF PL/I INTO ABSTRACT SYNTAX

(187) `mk-file-decl(pd) =`
 $\mu_0(<\text{s-scope:scope-attr}(pd)>, <\text{s-file-attr:fa}> | \exists \text{ad} (\text{ad} \in \text{s-attr-ds}(pd) \& \text{is-c-file-attribute=ad}(t) \& \text{fa} = \text{trans-file-attr=ad}(t)))>, <\text{s-env-attr:mk-env-attr=s, env-attr-p}(pd)>)$

Ref.: scope-attr 4-26(93)
 mk-env-attr 4-23(82)

(188) `trans-file-attr(c-fa) =`
`is-c-BITSTREAM(c-fa) --> BST`
`is-c-STREAM(c-fa) --> CST`
`is-c-RECORD(c-fa) --> REC`
`is-c-INPUT(c-fa) --> INP`
`is-c-OUTPUT(c-fa) --> OUT`
`is-c-UPDATE(c-fa) --> UPD`
`is-c-SEQUENTIAL(c-fa) --> SEQ`
`is-c-DIRECT(c-fa) --> DIR`
`is-c-BUFFERED(c-fa) --> BUF`
`is-c-UNBUFFERED(c-fa) --> UNB`
`is-c-KEYED(c-fa) --> KEY`
`is-c-PRINT(c-fa) --> PRT`
`is-c-BACKWARDS(c-fa) --> BAC`
`is-c-EXCLUSIVE(c-fa) --> EXC`
`is-c-TRANSIENT(c-fa) --> TRA`

(189) `env-attr-p(pd) =`
 $\exists \rightarrow (\exists \text{ad} (\text{ad} \in \text{s-attr-ds}(pd) \& \text{is-c-ENVIRONMENT=s}_1\text{-ad}(t))$
 $\text{T} \rightarrow \Omega)$

4.2.2.13 Defined and based declarations

In this section the construction of defined and based declarations from their preliminary declarations is described.

(190) `mk-defined-decl(pd) =`
`is-<> * mk-init(pd) -->`
 $\mu_0(<\text{s-base:trans-ref}(s_2 * ((\exists \text{ad}) (\text{ad} \in \text{s-attr-ds}(pd) \& \text{is-c-DEFINED=s}_1\text{-ad}(t))))>, <\text{s-aggr:mk-aggr}(pd, bdp-list(pd))>, <\text{s-pos:trans-opt-exp}(s_3 * ((\exists \text{ad}) (\text{ad} \in \text{s-attr-ds}(pd) \& \text{is-c-POSITION=s}_1\text{-ad}(t))))>)$

cont'd

Ref.: mk-init 4-54(215)
 trans-ref 4-83(323)
 mk-aggr 4-23(85)
 bdp-list 4-28(102)
 trans-opt-expr 4-82(322)

(191) mk-based-decl(pd) =
 $\mu_0(<s\text{-ptr:based-}\mu_0(s\text{-ptr}(pd)), s\text{-aggr:mk-aggr}(pd, bdp-list(pd)))>)$

Ref.: mk-aggr 4-23(85)
 bdp-list 4-28(102)

(192) based-ptr(pd) =
 $\neg(\exists ad)(ad \in s\text{-attr-ds}(pd) \wedge is-c-BASED \cdot s_1 \cdot ad(t) \wedge \neg is-\Omega \cdot s_2 \cdot ad(t)) \rightarrow \Omega$
 $T \rightarrow trans-ref(s_2 \cdot s_2 \cdot bas_1)$

where:
 $bas_1 = (\exists ad)(ad \in s\text{-attr-ds}(pd) \wedge is-c-BASED \cdot s_1 \cdot ad(t))$

Ref.: trans-ref 4-83(323)

4.2.2.14 Format label declaration

Format sentences are translated into declarations of format labels and do not appear in any statement list of the abstract program. The declaration of a format label consists of the translation of the statement condition part and of the translation of the format list. The latter translation occurs also for format lists of get and put statements.

(193) mk-format-decl(pd) =
 $\mu_0(<s\text{-format-list:trans-format-list}(s_4 \cdot (format-attr-p(pd))),$
 $<s\text{-cond-part:trans-st-cond-part}(s_1 \cdot (format-attr-p(pd))),$
 $<s\text{-ident:mk-id-1}(s_1 \cdot s_1 \cdot s_2 \cdot format-attr-p(pd))>)$

Ref.: trans-st-cond-part 4-62(242)
 mk-id-1 4-90(353)

(194) format-attr-p(pd) =
 $(\forall q)(is-c-format-sentence \cdot q(t) \wedge q \Rightarrow (\forall p)(\langle p \rangle = pd-seq(pd)))$

Ref.: pd-seq 4-31(113)

(195) trans-format-list(p) =
 $is-c-formatlist \cdot p(t) \rightarrow \begin{array}{c} \text{length} \cdot s_1 \cdot p(t) \\ \text{LIST } \end{array} \sum_{n=1}^{\text{length}} trans-format(s_n \cdot s_2 \cdot p)$
 $T \rightarrow \langle trans-format(p) \rangle$

for: (is-c-formatlist v is-c-format-item) · p(t)

30 June 1969

TRANSLATION OF PL/I INTO ABSTRACT SYNTAX

(196) trans-format(p) =
 is-c-format-iteration*p(t) -->
 $\mu_0(<s\text{-rep-factor:rep}_1>, <s\text{-format-list:trans-format-list}(s_2\text{*}p)>)$
 $T \rightarrow \mu_0(\text{trans-format-item}(p); <s\text{-format-type:type}_1>)$

where:

$t_1 = s_1\text{*}p(t)$
 $\text{rep}_1 = (\text{is-c-integer}(t_1) \rightarrow \text{mk-const}(t_1),$
 $T \rightarrow \text{trans-expr}(s_2\text{*}s_1\text{*}p))$
 $\text{type}_1 = (\text{is-c-BX}(t_1) \rightarrow \text{BSPACE},$
 $\text{is-c-BCOLUMN}(t_1) \rightarrow \text{BCOL},$
 $\text{is-c-BB}(t_1) \rightarrow \text{BBIT},$
 $\text{is-c-BP}(t_1) \rightarrow \text{BPIC},$
 $\text{is-E-CHAR}(t_1) \rightarrow \text{FLT},$
 $\text{is-F-CHAR}(t_1) \rightarrow \text{FIX},$
 $\text{is-C-CHAR}(t_1) \rightarrow \text{CPLX},$
 $\text{is-B-CHAR}(t_1) \rightarrow \text{BIT},$
 $\text{is-A-CHAR}(t_1) \rightarrow \text{CHAR},$
 $\text{is-P-CHAR}(t_1) \rightarrow \text{PIC},$
 $\text{is-c-COLUMN}(t_1) \rightarrow \text{COL},$
 $\text{is-c-LINE}(t_1) \rightarrow \text{LINE},$
 $\text{is-c-PAGE}(t_1) \rightarrow \text{PAGE},$
 $\text{is-c-SKIP}(t_1) \rightarrow \text{SKIP},$
 $\text{is-X-CHAR}(t_1) \rightarrow \text{SPACE},$
 $\text{is-R-CHAR}(t_1) \rightarrow \text{REMOTE})$

for:(is-c-format-iteration * is-c-format-item)*p(t)

Ref.: mk-const 4-87(341)
 trans-expr 4-81(319)

(197) trans-format-item(p) =
 is-E-CHAR*s_1*p(t) & is-Q*s_*p(t) * is-c-PAGE*s_1*p(t) & ~is-Q*s_*p(t) -- error
 is-c-real-format*p(t) -->
 $\mu_0(<s\text{-w:trans-expr}(s_3\text{*}p)>, <s\text{-d:d}_1>, <s\text{-p-s:trans-opt-expr}(s_2\text{*}s_3\text{*}s_*\text{*}p)>)$
 is-c-complex-format*p(t) -->
 $\mu_0(<s\text{-real:trans-format}(s_1\text{*}p)>, <s\text{-imag:trans-format}(imag_1)>)$
 is-c-string-format*p(t) * is-c-control-format*p(t) -->
 $\mu_0(<s\text{-w:trans-opt-expr}(s_2\text{*}s_2\text{*}p)>)$
 is-c-picture-format*p(t) --> $\mu_0(<s\text{-pic:trans-pic}(s_2\text{*}p(t), Q)>)$
 is-c-remote-format*p(t) --> $\mu_0(<s\text{-ref:trans-ref}(s_2\text{*}p)>)$

cont'd

TRANSLATION OF PL/I INTO ABSTRACT SYNTAX

30 June 1969

where:

$$\begin{aligned} d_1 &= (\text{is-c-expression} \cdot s_2 \cdot s_4 \cdot p(t) \rightarrow \text{trans-expr}(s_2 \cdot s_4 \cdot p), \\ T &\rightarrow \text{mk-const}(<0-\text{CHAR}>)) \\ \text{image}_1 &= (\text{is-0} \cdot s_4 \cdot p(t) \rightarrow s_3 \cdot 0, \\ T &\rightarrow s_2 \cdot s_4 \cdot p) \end{aligned}$$

for:is-c-format-item-p(t)

Ref.: trans-expr 4-81(319)
 trans-opt-expr 4-82(322)
 trans-pic 4-37(148)
 trans-ref 4-83(323)
 mk-const 4-87(341)

4.2.2.15 Generic declaration

A generic declaration is a list of generic members, which are computed in a way very similar to the computation of an entry declaration. The main difference is, that for generic descriptors no default attributes are supplied neither by the function prel-descr-list nor during translation.

(198) mk-generic-decl(pd) =

$$\begin{aligned} &\text{lg}_1 \\ &\text{LIST } \underset{n=1}{\text{mk-gen-member}} \cdot s_n \cdot s_3 \cdot \text{generic-attr-p}(pd) \end{aligned}$$

where:

$$\text{lg}_1 = \text{slength} \cdot s_3 \cdot \text{generic-attr-p}(pd)$$

(199) generic-attr-p(pd) =

$$(\text{bad}) (\text{ad} \in \text{s-attr-ds}(pd) \& \text{is-c-generic-attribute}=\text{ad}(t))$$

(200) mk-gen-member(p) =

$$\begin{aligned} &\mu_0 (<\text{s-entry}:\text{trans-ref}(s_1 \cdot p)>, \\ &\quad \underset{n=1}{\text{lg}_1} \\ &\quad <\text{s-descr-list}: \text{LIST } \underset{n=1}{\text{mk-gen-descr-elem}}(n, \text{prel-descr-list}(p))>) \end{aligned}$$

where:

$$\text{lg}_1 = \text{length} \cdot \text{prel-descr-list}(p)$$

Ref.: trans-ref 4-83(323)
 prel-descr-list 4-56(222)

(201) mk-gen-descr(pd) =

$$\mu_0 (<\text{s-stg-cl}:\text{stg-class-attr}(pd), <\text{s-aggr}:\text{mk-gen-aggr}(pd, \text{hdp-list}(pd))>)$$

Ref.: stg-class-attr 4-27(98)
 hdp-list 4-28(102)

30 June 1969

TRANSLATION OF PL/I INTO ABSTRACT SYNTAX

```
(202) mk-gen-aggr(pd,bdpl) =
      is-<>(bdpl) --> mk-gen-non-array(pd)
      is-**head(bdpl) --> μ₀(<s-elem:mk-gen-aggr(pd,tail(bdpl))>)

(203) mk-gen-non-array(pd) =
      length·succ-list(pd)
      is-STRUCT·da-attr(pd) --> LIST μ₀(<s-aggr:mk-gen-aggr(pd₁,bdp-list(pd₁))>
                                              n=1)
```

T -->

```
      μ₀(<s-da:mk-gen-da(pd)>,<s-dens:
          3 --> (battr)(is-dens-attr(attr,pd))
          T --> R>)
```

where:

pd₁ = elem(n,succ-list(pd))

Ref.: da-attr 4-32(117)
 bdp-list 4-28(102)
 is-dens-attr 4-27(97)
 succ-list 4-30(112)

```
(204) mk-gen-da(pd) =
      ~ (Bad) (ad ∈ s-attr-ds(pd) &
              ~ (is-c-ALIGNED ∨ is-c-UNALIGNED ∨ is-c-dimension-attribute) • ad(t) -->
              *
              is-ARITHM·da-attr(pd) --> mk-gen-arithm(pd)
              is-STRING·da-attr(pd) --> mk-gen-string(pd)
              is-AREA·da-attr(pd) --> mk-gen-area(pd)
              is-PIC·da-attr(pd) --> pic-da(pd)
              (is-LABEL ∨ is-PTR ∨ is-OFFSET ∨ is-TASK ∨ is-EVENT ∨ is-ENTRY ∨
               is-FILE) • da-attr(pd) -->
              da-attr(pd))
```

Ref.: da-attr 4-32(117)
 pic-da 4-37(146)

```
(205) mk-gen-arithm(pd) =
      μ₀(<s-mode:gen-mode•s-attr-ds(pd)>,<s-base:gen-base•s-attr-ds(pd)>,
          <s-scale:gen-scale•s-attr-ds(pd)>,<s-spec:gen-prec•prec-p(pd)>,
          <s-scale-f:gen-scale-f•prec-p(pd)>)
```

Ref.: prec-p 4-34(128)

TRANSLATION OF PL/I INTO ABSTRACT SYNTAX

30 June 1969

```
(206) gen-mode(ad-set) =
      E --> trans-attr•((bad) (ad ∈ ad-set & (is-c-REAL ∨ is-c-COMPLEX) • ad(t))) (t)
      T --> Ω

(207) gen-base(ad-set) =
      E --> trans-attr•((bad) (ad ∈ ad-set & (is-c-DECIMAL ∨ is-c-BINARY) • ad(t))) (t)
      T --> Ω

(208) gen-scale(ad-set) =
      E --> trans-attr•((bad) (ad ∈ ad-set & (is-c-FIXED ∨ is-c-FLOAT) • ad(t))) (t)
      T --> Ω

(209) trans-attr(attr) =
      is-c-REAL(attr) --> REAL
      is-c-COMPLEX(attr) --> CPLX
      is-c-DECIMAL(attr) --> DEC
      is-c-BINARY(attr) --> BTN
      is-c-FIXED(attr) --> FIX
      is-c-FLOAT(attr) --> FLT
      is-c-CHARACTER(attr) --> CHAR
      is-c-BIT(attr) --> BIT

(210) gen-prec(prec) =
      is-Ω(prec) --> Ω
      T --> const-val•s2(prec)
```

Ref.: const-val 4-89 (347)

```
(211) gen-scale-f(prec) =
      is-Ω(prec) --> Ω
      -is-Ω•s2•s3(prec) --> const-val•s2•s3(prec)
```

Ref.: const-val 4-89 (347)

```
(212) mk-gen-string(pd) =
      μ0(<s-base:gen-base-1(pd)>, <s-varying:varying-attr(pd)>)
```

Ref.: varying-attr 4-36 (145)

30 June 1969

TRANSLATION OF PL/T INTO ABSTRACT SYNTAX

(213) gen-base-1(pd) =
 ~ (bad) (is-string-attr-p(ad, pd)) -- Ø
 (is-Ø v is-*) *s₂*s₂(str₁) -- trans-attr*s₁(str₁)

where:

str₁ = ((bad) (is-string-attr-p(ad, pd))) (t)

Ref.: is-string-attr-p 4-35(141)

Note: The only string length specification which is allowed in a generic descriptor is an asterisk.

(214) mk-gen-area(pd) =
 (is-Ø v
 is-*) *s₂*s₂* ((bad) (ad ∈ s-attr-ds(pd) & is-c-area-attribute*ad(t))) (t) --
 AREA

Note: The only area size specification which is allowed in a generic descriptor is an asterisk.

4.2.2.16 Initial attribute and initial label

Each scalar variable may either have exactly one initial attribute or be connected with a list of initial labels.

Initial labels are translated into objects with two components:

one containing the list of subscripts,

the other containing the abstract name of the initial label.

The latter is a true translation of the concrete denotation, including parentheses, subscripts and qualification points. The computation of the initial label list will fail, if there are elements with equal subscriptlists in it.

TRANSLATION OF PL/I INTO ABSTRACT SYNTAX

30 JUNE 1969

```
(215) mk-init(pd) =
      ( $\exists ad$ ) ( $ad \in s\text{-attr-ds}(pd)$  &  $is\text{-c-initial-attribute}\circ ad(t)$ ) &
      ( $\exists p$ ) ( $is\text{-init-label-cont}(p)$  &  $pd = decl\text{-ref}(block\text{-p}(p), p)$ ) -->
      error
      B -->
      env-trans-init-spec(pd-block-p(pd), ( $\exists ad$ ) ( $ad \in s\text{-attr-ds}(pd)$  &
       $is\text{-c-initial-attribute}\circ ad(t)$ ))
      ( $\exists p$ ) ( $is\text{-init-label-cont}(p)$  &  $pd = decl\text{-ref}(block\text{-p}(p), p)$ ) -->
      ( $\neg (\exists n, m)$  ( $n \neq m$  &  $n < length(tr\text{-inl}_1)$  &
       $s\text{-sl-elem}(n, tr\text{-inl}_1) = s\text{-sl-elem}(m, tr\text{-inl}_1)$ ) -->
      tr-inl_1)
      T --> <>
```

where:

 $tr\text{-inl}_1 = trans\text{-spec-init-list}(pd)$

Ref.: decl-ref 4-84(326)
 block-p 4-4(8)
 pd-block-p 4-29(107)

```
(216) env-trans-init-spec(b, p) =
      is-c-initial-call $\circ s_2 \circ p(t)$  --> env-trans-call-st(b,  $s_2 \circ p$ )
      slength $\circ s_1 \circ p(t)$ 
      is-c-initial-itemlist $\circ s_2 \circ p(t)$  --> LIST  $\underset{n=1}{env\text{-trans-init}(b, s_n \circ s_2 \circ p)}$ 
```

Ref.: env-trans-call-st 4-66(259)

```
(217) env-trans-init(b, p) =
      is-c-initial-iteration $\circ p(t)$  -->
       $\mu_0(<s\text{-rep}:env\text{-trans-expr}(b, s_2 \circ p)>, <s\text{-init}:
      slength\circ s_1 \circ s_2 \circ p(t)$ 
      is-c-initial-itemlist $\circ s_4 \circ p(t)$  --> LIST  $\underset{n=1}{env\text{-trans-init}(b, s_n \circ s_2 \circ s_4 \circ p)}$ 
      T --> <env-trans-init(b, s_4 \circ p)>>
      is-ASTER $\circ p(t)$  --> *
      (is-c-initial-constant v is-c-simple-string-constant) $\circ p(t)$  -->
      env-trans-init-const(b, p)
      T --> env-trans-expr(b, p)
```

Ref.: env-trans-expr 4-82(320)

30 June 1969

TRANSLATION OF PL/I INTO ABSTRACT SYNTAX

(218) env-trans-init-const(b,p) =
 is-c-arithmetic-init-constant•p(t) & -is-0•s₃•p(t) --
 $\mu_0(<s\text{-operator:ADD}, <s\text{-op-1:mk-const}(6(p(t); s_3)), <s\text{-op-2:mk-const}\cdot s_3\cdot p(t)>)$
 T --> mk-const•p(t)

Ref.: mk-const 4-87(341)

(219) trans-spec-init-list(pd) =
 length(coll,
 LIST trans-init-label•elem(n,coll₁)
 n=1

where:
 coll₁ = collect({p | is-init-label-cont(p) & pd = decl-ref(block-p(p),p)})

Ref.: decl-ref 4-84(326)
 block-p 4-4(8)

(220) trans-init-label(p) =
 $\mu_0(<s\text{-id:mk-id-1}(p), <s\text{-sl:s-sl}\cdot trans-ref(p)>)$

for:is-c-basic-reference•p(t)

Ref.: mk-id-1 4-90(353)
 trans-ref 4-83(323)

(221) is-init-label-cont(p) =
 (is-entry-cont v is-label-cont)(p) & -is-<>•s-sl•trans-ref(p)

for:is-c-basic-reference•p(t)

Ref.: is-entry-cont 4-11(40)
 is-label-cont 4-11(38)
 trans-ref 4-83(323)

4.2.3 PARAMETER DESCRIPTORS AND ALLOCATE ITEMS

Most of the functions in section 4.2.2 for declarations are applicable and useful for parameter descriptors and allocate items as well. Only two functions (pd-block-p and succ-list) need a case distinction for their inclusion.

This is possible if analogously to the preliminary declarations "preliminary descriptors" are constructed for the parameter descriptors and allocate items, which have a very similar form to the preliminary declarations.

This section defines all those functions necessary for the translation of parameter descriptors and allocate items, which are analogous to the functions defined in section 4.2.1 for declarations. For the construction of the declaration part of a block one needs the set of all level-one preliminary declarations belonging to that block, defined by the function prel-decl-set-1.

Analogously for the translation of an entry attribute, a generic member or an allocate statement, one needs the list of all level-one preliminary descriptors belonging to that entry attribute, generic member or allocate statement, defined by the function prel-descr-list.

For the computation of the attribute descriptor set of an entry descriptor descriptor default attributes have been considered analogously to the computation of all other defaults. Generic members and allocate items are not supported by default sentences.

```
(222) prel-descr-list(p) =
      ~ (3n) (is-invalid-descr-ln=elem(n,descr-p-list(p))) &
      ~ (3n,ad) (ad ∈ s-attr-ds*prel-descr*elem(n,descr-p-list(p)) &
      is-c-like-attribute=ad(t)) -->
      lg1
      LIST prel-descr*elem(n,descr-p-list-1(p))
      n=1
```

where:

lg₁ = length*descr-p-list-1(p)

for:(is-c-entry-name-attribute ∨ is-c-generic-element ∨
is-c-allocate-statement)*p(t)

```
(223) descr-p-list-1(p) =
      collect({q | q ∈ elem-set*descr-p-list(p) & descr-ln(q) = ±1})
      for:(is-c-entry-name-attribute ∨ is-c-generic-element ∨  
is-c-allocate-statement)*p(t)
```

Ref.: elem-set 4-38(152)

```
(224) descr-p-list(p) =
      is-c-ENTRY*s1*p(t) --> entry-descr-p-list(<>,s2*s3*p)
      is-c-reference*s1*p(t) --> entry-descr-p-list(<>,s4*p)
      is-c-allocate-statement*p(t) --> LIST sn*s2*p
      n=1
```

```
(225) entry-descr-p-list(p-list,p) =
      is-0*p(t) --> p-list
      T --> entry-descr-p-list(p-list^<s1*p>,s2*s3*p)
```

Note: Since parameter descriptors may be completely missing in a concrete program, the concrete syntax has to describe the parameter descriptor list recursively instead of using the list notation. Therefore the function entry-descr-p-list has to collect the pointers to parameter descriptors into a proper list.

30 June 1969

TRANSLATION OF PL/I INTO ABSTRACT SYNTAX

(226) prel-descr(p) =
 is-Q*p(t) --> *
 is-c-descriptor*p(t) & (Eq)(q => p & is-c-entry-name-attribute*q(t)) -->
 $\mu_0(<s-p:p>, <s-attr-ds:descr-def(decl-descr-attr(p), block-p(p))>)$
 is-c-descriptor*p(t) & (Eq)(q => p & is-c-generic-attribute*q(t)) -->
 $\mu_0(<s-p:p>, <s-attr-ds:decl-descr-attr(p)>)$
 is-c-controlled-allocate-item*p(t) -->
 $\mu_0(<s-id-list:mk-al-id-list(p)>, <s-p:p>, <s-attr-ds:[ad | is-al-attr-of(p, ad)]>)$
 is-c-based-allocate-item*p(t) --> p

for:(is-c-descriptor v is-c-controlled-allocate-item v
 is-c-based-allocate-item)*p(t)

Ref.: block-p 4-4(8)

(227) descr-def(ad-set,b) =
 b = I v (Eq)(p & descriptor-defaults(b) & is-c-SYSTEM*p(t)) --> def₁
 T --> descr-def(ad-set u def₁, block-p(b))

where:

def₁ = def-extension(ad-set,descriptor-defaults(b))

Ref.: block-p 4-4(8)
 def-extension 4-19(70)

(228) descriptor-defaults(b) =

{g |
 (Eq)(b => p & is-local-to(b,p) &
 (is-c-default-option-1*p(t) & s₃*p => q v is-c-default-spec*p(t) &
 (Eq)(s₁*p => r & is-c-DESCRIPTIONS*r(t)) & s₂*p => q) &
 is-default-attr(q))}

Ref.: is-local-to 4-4(10)
 is-default-attr 4-20(76)

(229) decl-descr-attr(p) =

{ad | is-descr-attr-of(p,ad) v is-descr-dens(p,ad)}

for:is-c-descriptor*p(t)

(230) $\text{is-descr-attr-of}(p, ad) =$

$$\begin{aligned} & \text{is-c-attribute-}ad(t) \wedge (\exists n) (ad = s_n * s_3 * p) \vee \text{is-c-dimension-}attribute-}ad(t) \wedge \\ & ad = s_2 * p \vee (\exists q) (\text{is-descr-succ-of}(p, q)) \wedge \text{is-STRUCT}(ad) \vee \\ & (\exists q) (\text{is-descr-succ-of}(q, p)) \wedge \text{is-SUCC}(ad) \vee \text{is-PARAM}(ad) \end{aligned}$$

for:is-c-descriptor•p(t)

(231) $\text{is-descr-dens}(p, ad) =$

$$\begin{aligned} & \exists \rightarrow ad = (\exists q) ((\text{is-c-ALIGNED} \vee \text{is-c-UNALIGNED}) * q(t) \wedge \text{is-descr-attr-of}(p, q)) \\ & \exists \rightarrow \text{is-descr-dens}((\exists q) (\text{is-descr-succ-of}(p, q)), ad) \\ & T \rightarrow F \end{aligned}$$

for:is-c-descriptor•p(t)

(232) $\text{is-descr-succ-of}(p, q) =$

$$\begin{aligned} & (\exists r, p-list) (p-list = \text{descr-p-list}(r) \wedge \text{is-descr-subel-of}(p, q, p-list) \wedge \\ & \neg(\exists p-1) (\text{is-descr-subel-of}(p, p-1, p-list) \wedge \text{is-descr-subel-of}(p-1, q, p-list))) \end{aligned}$$

for:(is-c-descriptor \vee is-c-controlled-allocate-item \vee
is-c-based-allocate-item)•p(t), (is-c-descriptor \vee
is-c-controlled-allocate-item \vee is-c-based-allocate-item)•q(t)

(233) $\text{is-descr-subel-of}(p, q, p-list) =$

$$\begin{aligned} & (\exists n, m) (p = \text{elem}(n, p-list) \wedge q = \text{elem}(n, p-list) \wedge n < m \wedge \\ & 1 \leq \text{descr-ln}(p) < \text{descr-ln}(q) \wedge \\ & \neg(\exists k) (n < k < m \wedge \text{descr-ln} \cdot \text{elem}(k, p-list) \leq \text{descr-ln}(p))) \end{aligned}$$

for:(is-c-descriptor \vee is-c-controlled-allocate-item \vee
is-c-based-allocate-item)•p(t), (is-c-descriptor \vee
is-c-controlled-allocate-item \vee is-c-based-allocate-item)•q(t)

(234) $\text{descr-ln}(p) =$

$$\begin{aligned} & \text{is-0} * s_1 * p(t) \vee \text{is-c-based-allocate-item} * p(t) \rightarrow -1 \\ & T \rightarrow \text{const-val} * s_1 * p(t) \end{aligned}$$

for:(is-c-descriptor \vee is-c-controlled-allocate-item \vee
is-c-based-allocate-item)•p(t)

Ref.: const-val 4-89(347)

(235) $\text{is-invalid-descr-ln}(p) =$

$$\text{descr-ln}(p) \neq \pm 1 \wedge \neg(\exists q) (\text{is-descr-succ-of}(q, p))$$

30 June 1969

TRANSLATION OF PL/I INTO ABSTRACT SYNTAX

(236) is-al-attr-of(p,ad) =
 is-c-attribute-ad(t) & ($\exists n$) (ad = $s_n \cdot s_{n+1} \cdot p$) \vee is-c-dimension-attribute-ad(t) &
 $ad = s_1 \cdot p$ \vee ($\exists q$) (is-descr-succ-of(p,q)) & is-STRUCT(ad) \vee
 $\neg (\exists q)$ (is-descr-succ-of(q,p)) & is-SUCC(ad)

for:is-c-controlled-allocate-item-p(t)

(237) mk-al-id-list(p) =
 lg,
 $\text{LIST} \cdot \text{mk-id-1}(s_2 \cdot (\text{elem}(n, \text{al-succ-list}(p))))$
 $n=1$

where:

lg₁ = length-al-succ-list(pd)

for:is-c-based-allocate-item-p(t)

Ref.: mk-id-1 4-90 (353)

(238) al-succ-list(p) =

($\cup \text{seq}$) (is-list(seq) & last(seq) = p & descr-ln-head(seq) = ±1 &
 $\text{length}(\text{seq}) - 1$
 $\quad \quad \quad \text{Et } \text{is-descr-succ-of}(\text{elem}(n, \text{seq}), \text{elem}(n + 1, \text{seq}))$
 $n=1$

4.3 STATEMENTS

The translation of statements is in general a simple one to one mapping of the abstract representation of the concrete statement texts into the corresponding objects of the abstract syntax.

Anticipated is the translation of the label- and condition prefixes. The translation of proper statements is performed in the following sections:

- (1) blocks and groups
- (2) flow of control statements
- (3) storage manipulating statements
- (4) condition and attention handling statements
- (5) input and output statements

TRANSLATION OF PL/I INTO ABSTRACT SYNTAX

30 June 1969

(239) trans-st(p) =

$$\mu_0(\text{is-c-declaration-sentence} \cdot p(t) \rightarrow \mu_0(\langle s\text{-cond-part:empty-cond-pt}, s\text{-label-list:trans-label-list}(s_1 \cdot p) \rangle, \langle s\text{-st} \cdot s\text{-prop-st:NULL} \rangle)$$

T →

$$\mu_0(\langle s\text{-cond-part:trans-st-cond-part}(s_1 \cdot p) \rangle, \langle s\text{-label-list:trans-label-list}(s_2 \cdot p) \rangle, \langle s\text{-prop-st:trans-prop-st}(s_3 \cdot p) \rangle)$$

for:(is-c-declaration-sentence ∨ is-c-statement ∨ is-c-end-clause) • p(t)

Ref.: trans-label-list 4-64(250)
trans-st-cond-part 4-62(242)

Note: Declaration sentences are translated into labeled null-statements.

(240) empty-cond-pt =

$$\mu_0(\langle s\text{-on:}<>, s\text{-no:}<> \rangle)$$

30 June 1969

TRANSLATION OF PL/I INTO ABSTRACT SYNTAX

```
(241) trans-prop-st(p) =
    is-c-begin-block•p(t) --> trans-block(p)
    is-c-group•p(t) --> trans-group(p)
    is-c-if-clause•s1•p(t) --> trans-if-st(p)
    is-c-goto-statement•p(t) --> trans-goto-st(p)
    is-c-call-statement•p(t) --> env-trans-call-st(block-p(p), p)
    is-c-return-statement•p(t) --> trans-return-st(p)
    is-c-incorporate-statement•p(t) --> trans-incorporate-st(p)
    is-c-fetch-statement•p(t) --> trans-fetch-st(p)
    is-c-release-statement•p(t) --> trans-release-st(p)
    is-c-wait-statement•p(t) --> trans-wait-st(p)
    is-c-delay-statement•p(t) --> trans-delay-st(p)
    is-c-exit-statement•p(t) --> μ0(<s-st:EXIT>)
    is-c-stop-statement•p(t) --> μ0(<s-st:STOP>)
    is-c-assignment-statement•p(t) --> trans-assign-st(p)
    is-c-allocate-statement•p(t) --> trans-allocate-st(p)
    is-c-free-statement•p(t) --> trans-free-st(p)
    is-c-on-statement•p(t) --> trans-on-st(p)
    is-c-revert-statement•p(t) --> trans-revert-st(p)
    is-c-signal-statement•p(t) --> trans-signal-st(p)
    is-c-access-statement•p(t) --> trans-access-st(p)
    is-c-enable-statement•p(t) --> trans-enable-st(p)
    is-c-disable-statement•p(t) --> trans-disable-st(p)
    is-c-open-statement•p(t) --> trans-open-st(p)
    is-c-close-statement•p(t) --> trans-close-st(p)
    is-c-stream-io-statement•p(t) --> trans-stream-io-st(p)
    is-c-record-io-statement•p(t) --> trans-record-io-st(p)
    is-c-display-statement•p(t) --> trans-display-st(p)
    is-c-null-statement•p(t) ∨ is-c-END•p(t) --> μ0(<s-st:NULL>)
```

Ref.: trans-block 4-64(251)
 trans-group 4-65(252)
 trans-if-st 4-66(256)
 trans-goto-st 4-66(258)
 env-trans-call-st 4-66(259)
 block-p 4-4(8)
 trans-return-st 4-67(265)
 trans-incorporate-st 4-67(266)
 trans-fetch-st 4-68(268)

cont'd

TRANSLATION OF PL/I INTO ABSTRACT SYNTAX

30 June 1969

```

trans-release-st 4-68(269)
trans-wait-st 4-68(270)
trans-delay-st 4-68(271)
trans-assign-st 4-69(272)
trans-allocate-st 4-69(274)
trans-free-st 4-71(283)
trans-on-st 4-72(286)
trans-revert-st 4-72(288)
trans-signal-st 4-72(289)
trans-access-st 4-74(293)
trans-enable-st 4-74(294)
trans-disable-st 4-74(299)
trans-open-st 4-77(306)
trans-close-st 4-78(308)
trans-stream-io-st 4-78(309)
trans-record-io-st 4-80(315)
trans-display-st 4-80(316)

```

4.3.1 STATEMENT PREFIXES

Condition prefixes are translated into objects containing lists of the enabled or disabled conditions and of lists of the references to be checked. The expansion of references to be checked to all scalar or array components of the referenced item is made by the Interpreter.

```
(242) trans-st-cond-part(p) =
      is-c-begin-block•s3•q1(t) --> empty-cond-pt
      ~( $\exists g$ ) (p => q & (is-c-check-condition v is-c-no-check-condition)•o(t)) -->
      trans-cond-part(p)
```

where:

$$q_1 = (\cup g) (p = s_1 \cdot q \vee p = s_2 \cdot q)$$

for: (is-c-prefixlist v is-Q)•p(t)

Ref.: empty-cond-pt 4-60(240)

```
(243) trans-cond-part(p) =
       $\mu_0(\langle s\text{-on:mk-on-pref-list}(p, 1), s\text{-no:mk-no-pref-list}(p, 1) \rangle)$ 
      for: (is-c-prefixlist v is-Q)•p(t)
```

```
(244) mk-on-pref-list(p, n) =
      is-Q•sn•p(t) --> <>
      T --> mk-on-pref-list-1(s2•sn•p, 1) ^ mk-on-pref-list(p, n + 1)
      for: (is-c-prefixlist v is-Q)•p(t)
```

30 June 1969

TRANSLATION OF PL/I INTO ABSTRACT SYNTAX

(245) $\text{mk-on-pref-list-1}(p, n) =$
 $\text{is-}\Omega\text{-}\text{s}_n\text{-}\text{p}(t) \rightarrow \text{<}>$
 $(\text{is-c-no-prefix} \vee \text{is-c-no-check-condition})\text{-}\text{s}_n\text{-}\text{p}(t) \rightarrow$
 $\text{mk-on-pref-list-1}(p, n + 1)$
 $\text{is-c-prefix-}\text{s}_n\text{-}\text{p}(t) \rightarrow \text{<} \text{mk-pref}(\text{s}_n\text{-}\text{p}(t)) \text{>}^* \text{mk-on-pref-list-1}(p, n + 1)$
 $T \rightarrow \text{<} \text{mk-check-cond}(\text{s}_3\text{-}\text{s}_n\text{-}\text{p}) \text{>}^* \text{mk-on-pref-list-1}(p, n + 1)$

where:

 $l_{g_1} = \text{slength-}\text{s}_3\text{-}\text{s}_n\text{-}\text{p}(t)$ for: (is-c-prefix-element \vee is- Ω) \cdot p(t)

(246) $\text{mk-pref(pref)} =$
 $\text{is-c-CONVERSION(pref)} \vee \text{is-c-NOCONVERSION(pref)} \rightarrow \text{CONV}$
 $\text{is-c-FIXEDOVERFLOW(pref)} \vee \text{is-c-NOFIXEDOVERFLOW(pref)} \rightarrow \text{FOFL}$
 $\text{is-c-OVERFLOW(pref)} \vee \text{is-c-NOOVERFLOW(pref)} \rightarrow \text{OFL}$
 $\text{is-c-SIZE(pref)} \vee \text{is-c-NOSIZE(pref)} \rightarrow \text{SIZE}$
 $\text{is-c-STRINGSIZE(pref)} \vee \text{is-c-NOSTRINGSIZE(pref)} \rightarrow \text{STRZ}$
 $\text{is-c-STRINGRANGE(pref)} \vee \text{is-c-NOSTRINGRANGE(pref)} \rightarrow \text{STRG}$
 $\text{is-c-SUBSCRIPTRANGE(pref)} \vee \text{is-c-NOSUBSCRIPTRANGE(pref)} \rightarrow \text{SUBRG}$
 $\text{is-c-UNDERFLOW(pref)} \vee \text{is-c-NOUNDERFLOW(pref)} \rightarrow \text{UPL}$
 $\text{is-c-ZERODIVIDE(pref)} \vee \text{is-c-NOZERODIVIDE(pref)} \rightarrow \text{ZDIV}$

for: (is-c-prefix \vee is-c-no-prefix) (pref)

(247) $\text{mk-check-cond}(p) =$
 $\text{slength-}\text{p}(t)$
 $\mu_0 (\text{<} \text{s-ref-list: LIST } \text{trans-ref}(\text{s}_n\text{-}\text{p}), \text{<} \text{s-cond:CHECK} \text{>})$

for: is-c-unsubscripted-reference \cdot p(t)

Ref.: trans-ref 4-83(323)

(248) $\text{mk-no-pref-list}(p, n) =$
 $\text{is-}\Omega\text{-}\text{s}_n\text{-}\text{p}(t) \rightarrow \text{<}>$
 $T \rightarrow \text{mk-no-pref-list-1}(\text{s}_3\text{-}\text{s}_n\text{-}\text{o}, 1)^* \text{mk-no-pref-list}(p, n + 1)$
 $\text{for: (is-c-prefixlist} \vee \text{is-}\Omega\text{)} \cdot \text{p}(t)$

TRANSLATION OF PL/I INTO ABSTRACT SYNTAX

30 June 1969

```
(249) mk-no-pref-list-1(p,n) =
      is-0*sn*p(t) -->
      (is-c-prefix v is-c-check-condition)*sn*p(t) --> mk-no-pref-list-1(p,n + 1)
      is-c-no-prefix*sn*p(t) --> <mk-pref(sn*p(t))>^mk-no-pref-list-1(p,n + 1)
      T --> <mk-check-cond(s3*sn*p)>^mk-no-pref-list-1(p,n + 1)
```

where:

lq₁ = slength*s₃*s_n*p(t)

for:(is-c-prefix-element v is-0)*p(t)

```
(250) trans-label-list(p) =
      slength*p(t)
      LIST mk-id-ref(s1*sn*p)
      n=1
```

for:is-c-basic-reference*p(t)

Ref.: mk-id-ref 4-6(19)

4.3.2 BLOCK AND GROUPS

A begin block is structured very similarly to a procedure body, but it differs from it in the fact that it is a special case of a statement, i.e. generally an element of a statement list. A simple group, i.e. a group without iteration specification is translated into a statement list, since the 'DO' and 'END' have in this case only a parentheses function.

```
(251) trans-block(b) =
      μ0(<s-decl-pt:mk-decl-part(b)>, <s-body-pt:mk-body-pt(b)>,
            <s-cond-part:trans-cond-part(s1*{(b p) (b = s3*p)})>,
            <s-st-list:mk-st-list(b)>, <s-reorder:mk-reorder(b)>)
```

Ref.: mk-decl-part 4-8(27)
 mk-body-pt 4-5(15)
 trans-cond-part 4-62(243)
 mk-st-list 4-7(22)
 mk-reorder 4-7(24)

30 June 1969

TRANSLATION OF PL/I INTO ABSTRACT SYNTAX

(252) `trans-group(p) =`

$$\begin{aligned} & \text{is-c-simple-group-p(t)} \rightarrow \text{mk-st-list}(p) \\ & \text{is-c-WHILE}\circ s_1\circ s_2\circ p(t) \rightarrow \\ & \quad \mu_0(\langle s\text{-while-expr:trans-expr}(s_3\circ s_2\circ p) \rangle, \langle s\text{-do-list:mk-st-list}(p) \rangle) \\ & T \rightarrow \\ & \quad \text{slength}'s_3\circ p(t) \\ & \quad \mu_0(\langle s\text{-contr-var:trans-ref}(s_1\circ p) \rangle, \langle s\text{-spec-list: LIST } \underset{n=1}{\text{trans-spec}}(s_n\circ s_3\circ p) \rangle, \\ & \quad \langle s\text{-do-list:mk-st-list}(p) \rangle) \end{aligned}$$

`for:is-c-group-p(t)`

Ref.: `mk-st-list` 4-7(22)
 `trans-expr` 4-81(319)
 `trans-ref` 4-83(323)
 `trans-spec` 4-65(253)

(253) `trans-spec(p) =`

$$\mu_0(\langle s\text{-init-expr:trans-expr}(s_1\circ p) \rangle, \langle s\text{-by-expr:mk-by-expr}(s_2\circ p) \rangle, \\ \langle s\text{-to-expr:mk-to-expr}(s_2\circ p) \rangle, \langle s\text{-while-expr:trans-opt-expr}(s_3\circ p) \rangle)$$

`for:is-c-specification-p(t)`

Ref.: `trans-expr` 4-81(319)
 `trans-opt-expr` 4-82(322)

(254) `mk-by-expr(p) =`

$$\begin{aligned} & \text{is-c-BY}\circ s_1\circ p(t) \rightarrow \text{trans-expr}(s_2\circ p) \\ & T \rightarrow \text{trans-opt-expr}(s_2\circ s_3\circ p) \end{aligned}$$

Ref.: `trans-expr` 4-81(319)
 `trans-opt-expr` 4-82(322)

(255) `mk-to-expr(p) =`

$$\begin{aligned} & \text{is-c-TO}\circ s_1\circ p(t) \rightarrow \text{trans-expr}(s_2\circ p) \\ & T \rightarrow \text{trans-opt-expr}(s_2\circ s_3\circ p) \end{aligned}$$

Ref.: `trans-expr` 4-81(319)
 `trans-opt-expr` 4-82(322)

4.3.3 FLOW OF CONTROL STATEMENTS

This section defines the translation of the if-, goto-, call-, return-, incorporate-, fetch-, release-, wait- and delay statement.

Since the translation of the call statement is used also for the initial call in the initial attribute, which might be translated in another block than that of

TRANSLATION OF PL/I INTO ABSTRACT SYNTAX

30 June 1969

its occurrence for the call statement its block pointer is used. The call options list is translated using the method for translation of option lists described for input and output statements.

```
(256) trans-if-st(p) =
      μ0(<s-st:IF>,<s-expr:trans-expr(s2*s1*p)>,<s-then-st:trans-st(s2*p)>,
            <s-else-st:trans-else-st(s1*p)>)

      for:(is-c-if-statement ∨ is-c-balanced-statement)*p(t)

      Ref.:    trans-expr 4-81(319)
              trans-st 4-60(239)

(257) trans-else-st(p) =
      is-0*p(t) --+
      μ0(<s-cond-part:empty-cond-pt>,<s-label-list:<>>,<s-st*s-prop-st:NULL>)
      T --+ trans-st(p)

      for:(is-c-statement ∨ is-0)*p(t)

      Ref.:    empty-cond-pt 4-60(240)
              trans-st 4-60(239)

(258) trans-goto-st(p) =
      μ0(<s-st:GOTO>,<s-ref:trans-ref(s2*p)>)

      for:is-c-goto-statement*p(t)

      Ref.:    trans-ref 4-83(323)

(259) env-trans-call-st(b,o) =
      μ0(<s-st:CALL>,<s-entry:mk-call-ref(b,s2*p)>,<s-arg-list:mk-arg-list(b,s2*p)>,
            <s-pa-option:trans-pa-opt(s3*p)>)

      for:(is-c-call-statement ∨ is-c-initial-call)*p(t)

(260) mk-call-ref(b,p) =
      is-<>*s-ap*env-trans-ref(b,p) --+ env-trans-ref(b,p)
      T --+ δ(env-trans-ref(b,p);last*s-ap)

      for:is-c-reference*p(t)

      Ref.:    env-trans-ref 4-83(324)
```

30 June 1969

TRANSLATION OF PL/I INTO ABSTRACT SYNTAX

- (261) **mk-arg-list(b,p) =**
- ```

 is-<>*s-ap*env-trans-ref(b,p) --> <>
 T --> last*s-ap*env-trans-ref(b,p)
 for:is-c-reference*p(t)

 Ref.: env-trans-ref 4-83(324)
```
- (262) **trans-pa-opt(p) =**
- ```

    is-Ω*p(t) --> Ω
    is-pa-opt*insert-aster*trans-optionslist(pd) -->
        insert-aster*trans-optionslist(p)
    for:is-c-call-optionslist*p(t)

    Ref.: is-pa-opt 5-11(97)
          trans-optionslist 4-75(300)
```
- (263) **insert-aster(pa-opt) =**
- ```

 μ0(<s-task:insert-aster-1*s-task(pa-opt)>,
 <s-event:insert-aster-1*s-event(pa-opt)>,<s-pri:s-pri(pa-opt)>)

 Note: This function inserts into an incomplete pa-option asterisks as task and
 event components if they are Ω.
```
- (264) **insert-aster-1(x) =**
- ```

    is-Ω(x) --> *
    T --> x
```
- (265) **trans-return-st(p) =**
- ```

 μ0(<s-st:RETURN>,<s-expr:trans-opt-expr(s2*s2*p)>)
 for:is-c-return-statement*p(t)

 Ref.: trans-opt-expr 4-82(322)
```
- (266) **trans-incorporate-st(p) =**
- ```

    μ0(<s-st:INCORPORATE>,<s-text:trans-incorporate-spec(s3*p(t))>)
    for:is-c-incorporate-statement*p(t)
```
- (267) **trans-incorporate-spec(x) =**

Note: This function is implementation defined.

```
(269) trans-fetch-st(p) =
       $\mu_0(\langle s-st:FETCH \rangle, \langle s-entry-list: \text{LIST}_{n=1}^{\text{length}\cdot s_3\cdot p(t)} \text{env-trans-ref}(\text{block-p}(p), s_n\cdot s_3\cdot p) \rangle)$ 

      for:is-c-fetch-statement•p(t)

      Ref.: env-trans-ref 4-83(324)
            block-p 4-4(8)

(269) trans-release-st(p) =
       $\mu_0(\langle s-st:RELEASE \rangle, \langle s-entry-list: \text{LIST}_{n=1}^{\text{length}\cdot s_3\cdot p(t)} \text{env-trans-ref}(\text{block-p}(p), s_n\cdot s_3\cdot p) \rangle)$ 

      for:is-c-release-statement•p(t)

      Ref.: env-trans-ref 4-83(324)
            block-p 4-4(8)

(270) trans-wait-st(p) =
       $\mu_0(\langle s-st:WAIT \rangle, \langle s-event-list: \text{LIST}_{n=1}^{\text{length}\cdot s_3\cdot p(t)} \text{trans-ref}(s_n\cdot s_3\cdot p),$ 
            $\langle s-event-number:trans-opt-expr(s_2\cdot s_3\cdot p) \rangle \rangle$ 

      for:is-c-wait-statement•p(t)

      Ref.: trans-ref 4-83(323)
            trans-opt-expr 4-82(322)

(271) trans-delay-st(p) =
       $\mu_0(\langle s-st:DELAY \rangle, \langle s-time:trans-expr(s_3\cdot p) \rangle)$ 

      for:is-c-delay-statement•p(t)

      Ref.: trans-expr 4-81(319)
```

4.3.4 STORAGE MANIPULATING STATEMENTS

The translation of the assignment statement and the free statement is just a one-to-one mapping of the concrete structure into the abstract one.

In the allocate statement the controlled allocate items are translated principally in the same way as declarations: First the list of (level-one) preliminary descriptors is formed as described in section 4.2.3, then from these preliminary descriptors the abstract allocate items are built similarly as the declarations, thereby using functions defined there.

30 June 1969

TRANSLATION OF PL/I INTO ABSTRACT SYNTAX

It is a requirement of the language, that a controlled allocate item either does not say anything about the structuring of its corresponding declaration (i.e. just the level-one identifier and possibly a dimension attribute is specified), or the structuring and the successor identifiers have to be specified completely and correctly. This is tested by the predicate `is-eq-id-struct`. After this test the successor identifiers are ignored by the translator.

Metavariables

`pd` preliminary descriptor of an allocate item

(272) `trans-assign-st(p) =`

$$\mu_0 (\langle s\text{-st:ASSIGN} \rangle, \langle s\text{-lp: LIST} \rangle, \langle s\text{-rp:trans-ref}(s_n\text{*}s_i\text{*}p) \rangle, \langle s\text{-rp:trans-expr}(s_3\text{*}p) \rangle,$$

$$\langle s\text{-byname:mk-opt}\circ s\text{*}p(t) \rangle)$$

`for:is-c-assignment-statement*p(t)`

Ref.: trans-ref 4-83(323)
trans-expr 4-81(319)

(273) `mk-opt(x) =`

$$\begin{aligned} \text{is-}\Omega(x) &\rightarrow \Omega \\ T &\rightarrow T \end{aligned}$$

(274) `trans-allocate-st(p) =`

$$\mu_0 (\langle s\text{-st:ALLOCATE} \rangle, \langle s\text{-list:trans-allocate-list}(p) \rangle)$$

`for:is-c-allocate-statement*p(t)`

(275) `trans-allocate-lis+(p) =`

$$\begin{aligned} l_{g_1}, \\ \text{LIST } \underset{n=1}{\text{mk-allocate-elem}}(n, \text{prel-descr-list}(p)) \end{aligned}$$

where:

$l_{g_1} = \text{length}\text{*}\text{prel-descr-list}(p)$

`for:is-c-allocate-statement*p(t)`

Ref.: prel-descr-list 4-56(222)

TRANSLATION OF PL/T INTO ABSTRACT SYNTAX

30 June 1969

(276) $\text{mk-allocate}(\text{pd}) =$

$$\begin{aligned} & \text{is-c-controlled-allocate-item}(\text{s-p(pd)}) (\text{t}) \wedge \\ & (\neg \text{is-STRUCT-da-attr}(\text{pd}) \vee \\ & \quad \text{is-eq-id-struct}(\text{pd}, \text{decl-ref}(\text{block-p} \cdot \text{s-p(pd)}, \text{s}_2 \cdot (\text{s-p(pd)})))) \dashv \\ & \mu_0(\langle \text{s-id:head} \cdot \text{s-id-list}(\text{pd}) \rangle, \langle \text{s-aggr:mk-al-aggr}(\text{pd}, \text{bdp-list}(\text{pd})) \rangle) \\ & \text{is-c-based-allocate-item}(\text{pd}(\text{t})) \dashv \\ & \mu_0(\langle \text{s-id:trans-id}(\text{s}_1 \cdot \text{pd}) \rangle, \langle \text{s-aggr:}^* \rangle, \langle \text{s-ptr:mk-ptr-ref}(\text{s}_2 \cdot \text{pd}) \rangle, \\ & \quad \langle \text{s-area:mk-area-ref}(\text{s}_2 \cdot \text{pd}) \rangle) \end{aligned}$$

Ref.: da-attr 4-32(117)
 decl-ref 4-84(326)
 block-p 4-4(8)
 bdp-list 4-28(102)
 trans-id 4-90(351)

(277) $\text{mk-al-aggr}(\text{pd}, \text{bdpl}) =$

$$\begin{aligned} & \text{is-} \langle \rangle(\text{bdpl}) \dashv \text{mk-al-non-array}(\text{pd}) \\ & \text{T} \dashv \mu(\text{head}(\text{bdpl}); \langle \text{s-elem:mk-al-aggr}(\text{pd}, \text{tail}(\text{bdpl})) \rangle) \end{aligned}$$

(278) $\text{mk-al-non-array}(\text{pd}) =$

$$\begin{aligned} & \text{is-STRUCT-da-attr}(\text{pd}) \dashv \text{LIST } \mu_0(\langle \text{s-aggr:mk-al-aggr}(\text{succ}_1, \text{bdpl}(\text{succ}_1)) \rangle) \\ & \quad \text{n=1} \\ & \text{T} \dashv \text{mk-al-scalar}(\text{pd}) \end{aligned}$$

where:
 $\text{succ}_1 = \text{elem}(\text{n}, \text{succ-list}(\text{pd}))$

Ref.: da-attr 4-32(117)
 succ-list 4-30(112)

(279) $\text{mk-al-scalar}(\text{pd}) =$

$$\begin{aligned} & \text{is-STRING-da-attr}(\text{pd}) \dashv \mu_0(\langle \text{s-da:string-da}(\text{pd}) \rangle, \langle \text{s-init:mk-init}(\text{pd}) \rangle) \\ & \text{is-AREA-da-attr}(\text{pd}) \dashv \mu_0(\langle \text{s-da:area-da}(\text{pd}) \rangle, \langle \text{s-init:mk-init}(\text{pd}) \rangle) \\ & \text{T} \dashv \mu_0(\langle \text{s-da:Q} \rangle, \langle \text{s-init:} \rangle \rangle) \end{aligned}$$

Ref.: da-attr 4-32(117)
 string-da 4-35(139)
 mk-init 4-54(215)
 area-da 4-40(161)

30 June 1969

TRANSLATION OF PL/I INTO ABSTRACT SYNTAX

(280) **is-eq-id-struct**(pd-1, pd-2) =
 s-id-list(pd-1) = s-id-list(pd-2) &
 length•succ-list(pd-1) = length•succ-list(pd-2) &
 length•succ-list(pd)
 Et **is-eq-id-struct**(elem(n,succ-list(pd-1)), elem(n,succ-list(pd-2)))
 n=1

Ref.: succ-list 4-30(112)

Note: If pd has no successors, the function succ-list(pd) yields \emptyset by its definition. Therefore the recursive definition of is-eq-id-struct comes trivially to an end.

(281) **mk-ptr-ref**(p) =
 is-c-SET•s₁•p(t) --> trans-ref(s₃•p)
 is-c-SET•s₁•s₅•p(t) --> trans-ref(s₃•s₅•p)
 T --> \emptyset

Ref.: trans-ref 4-83(323)

(282) **mk-area-ref**(p) =
 is-c-IN•s₁•s₅•p(t) --> trans-ref(s₃•s₅•p)
 is-c-IN•s₁•p(t) --> trans-ref(s₃•p)
 T --> \emptyset

Ref.: trans-ref 4-83(323)

(283) **trans-free-st**(p) =
 $\mu_0(\langle s-st:FREE \rangle, \langle s-list: \text{LIST } \underset{n=1}{\text{trans-free}}(s_n \cdot s_2 \cdot p) \rangle)$

for:is-c-free-statement•p(t)

(284) **trans-free**(v) =
 $\mu_0(\langle s-ref:mk-free-ref(s₁•p, block-p(p)) \rangle, \langle s-area:trans-opt-expr(s₃•s₂•p) \rangle)$
 for:is-c-reference•p(t)

Ref.: block-p 4-4(8)
 trans-opt-expr 4-82(322)

Note: The function trans-opt-expr, designed for optional expressions, translates as special case also optional references, yielding either \emptyset or the translated reference.

TRANSLATION OF PL/I INTO ABSTRACT SYNTAX

30 June 1969

```
(285) mk-free-ref(p,b) =
      is-0•s1•s2•p(t) & is-0•s3•s4•p(t) & length•s-id-list•decl-ref(b,p) = 1 -->
      env-trans-ref(b,p)

      for:is-c-reference•p(t)

      Ref.:   decl-ref 4-84(326)
              env-trans-ref 4-83(324)
```

4.3.5 CONDITION HANDLING STATEMENTS

The on-, revert and signal statements contain a condition specification which is to be translated into an object satisfying the predicate is-cond.

In the on-statement the on unit has to be tested to have no labels and not to be a group, return statement or on-statement.

```
(286) trans-on-st(p) =
      μ0(<s-st:ON>,<s-cond:trans-cond(s2•p)>,<s-snap:mk-opt•s3•p(t)>,
           <s-on-unit:trans-on-unit(s4•p)>)

      for:is-c-on-statement•p(t)

      Ref.:   mk-opt 4-69(273)
```

```
(287) trans-on-unit(p) =
      is-c-SYSTEM•s1•p(t) --> SYSTEM
      is-0•s2•p(t) &
      ~(is-c-group•s3•p(t) ∨ is-c-return-statement•s3•p(t) ∨
        is-c-on-statement•s3•p(t)) -->
      trans-st(p)

      Ref.:   trans-st 4-60(239)
```

```
(288) trans-revert-st(p) =
      μ0(<s-st:REVERT>,<s-cond:trans-cond(s2•p)>)

      for:is-c-revert-statement•p(t)
```

```
(289) trans-signal-st(p) =
      μ0(<s-st:SIGNAL>,<s-cond:trans-cond(s2•p)>)

      for:is-c-signal-statement•p(t)
```

30 June 1969

TRANSLATION OF PL/I INTO ABSTRACT SYNTAX

(290) trans-cond(p) =

is-c-prefix•p(t) --> mk-pref(p(t))
 is-c-check-condition•p(t) --> mk-check-cond(s₃•p)
 is-c-AREA•p(t) --> AREA
 is-c-named-io-condition•p(t) -->
 $\mu_0(<s\text{-ref:env-trans-ref}(s_3•p, \text{block-p}(p)), <s\text{-cond:trans-io-cond}\cdot s_1•p(t)>)$
 is-c-ERROR•p(t) --> ERROR
 is-c-FINISH•p(t) --> FINISH
 is-c-programmer-named-condition•p(t) --> $\mu_0(<s\text{-id:trans-id}(s_3•p), <s\text{-cond:COND}>)$
 is-c-attention-condition•p(t) --> mk-att-cond(s₃•p)

for:is-c-condition•p(t)

Ref.: mk-pref 4-63(246)
 mk-check-cond 4-63(247)
 env-trans-ref 4-83(324)
 block-p 4-4(8)
 trans-id 4-90(351)

(291) trans-io-cond(x) =

is-c-BEGINVOLUME(x) --> BOV
 is-c-ENDFILE(x) --> ENDF
 is-c-ENDPAGE(x) --> ENDP
 is-c-ENDVOLUME(x) --> EOF
 is-c-KEY(x) --> KEY
 is-c-NAME(x) --> NAME
 is-c-PENDING(x) --> PEND
 is-c-RECORD(x) --> REC
 is-c-TRANSMIT(x) --> TMT
 is-c-UNDEFINEDFILE(x) --> UNDF

(292) mk-att-cond(p) =

$\mu_0(<s\text{-attn-list: LIST}_{n=1}^{\text{length-p(t)}} \text{mk-id-ref}(s_n•p), <s\text{-cond:ATTN}>)$

Ref.: mk-id-ref 4-6(19)

(293) `trans-access-st(p) =`

$$\mu_0(\langle s-st:ACCESS \rangle, \langle s-cond:mk-att-cond(s_3 \cdot s_3 \cdot p) \rangle, \langle s-else: \\ is-SEMIC \cdot s_4 \cdot p(t) \rightarrow \Omega \\ T \rightarrow \mu_0(\langle s-st:trans-else-st(s_2 \cdot s_4 \cdot p) \rangle) \rangle)$$

`for:is-c-access-statement \cdot p(t)`

Ref.: trans-else-st 4-66(257)

(294) `trans-enable-st(p) =`

$$\mu_0(\langle s-st:ENABLE \rangle, \langle s-list: \text{LIST}_{n=1}^{slength \cdot s_1 \cdot p(t)} \text{mk-enable}(s_n \cdot s_2 \cdot p) \rangle)$$

`for:is-c-enable-statement \cdot p(t)`

(295) `mk-enable(p) =`

$$\mu_0(\langle s-cond: \text{LIST}_{n=1}^{slength \cdot s_1 \cdot p(t)} \text{mk-att-cond}(s_n \cdot s_3 \cdot s_1 \cdot p) \rangle, \langle s-spec:mk-access-spec(s_2 \cdot p) \rangle, \\ \langle s-event:mk-event-ref(s_2 \cdot p) \rangle)$$

(296) `mk-access-spec(p) =`

$$\exists \rightarrow \text{trans-spec}((\forall q) ((\text{is-c-ACCESS} \vee \text{is-c-ASYNC}) \cdot q(t) \& (\exists n) (q = s_n \cdot p)))$$

`T \rightarrow ACC`

(297) `trans-spec(p) =`

`is-c-ACCESS \cdot p(t) \rightarrow ACC`

`T \rightarrow ASYN`

(298) `mk-event-ref(p) =`

$$\exists \rightarrow \text{env-trans-ref}(\text{block-p}(p), s_3 \cdot (s((\forall n) (\text{is-c-EVENT} \cdot s_1 \cdot s_n \cdot p(t)))) \cdot p(t))$$

`T \rightarrow \Omega`

`for:is-c-reference \cdot p(t)`

Ref.: env-trans-ref 4-83(324)
block-p 4-4(8)

(299) `trans-disable-st(p) =`

$$\mu_0(\langle s-st:DISABLE \rangle, \langle s-cond:mk-att-cond(s_3 \cdot s_2 \cdot p) \rangle)$$

`for:is-c-disable-statement \cdot p(t)`

30 June 1969

TRANSLATION OF PL/I INTO ABSTRACT SYNTAX

4.3.6 INPUT AND OUTPUT STATEMENTS

The translation of the I/O statements (open, close, stream-io, record-io, and display) involves translation of the so called statement options. These are components of the statement text which occur in unspecified order but no more than once. The translation of the options consists of a check on multiple occurrence which is followed by the translation of single options, and finally the translated options (or the whole statement) is checked against the abstract syntax.

Statement options are not confined to I/O statements, e.g. the call statement, do-specification etc. contain them. In some instances where only two statement options can occur translation is done directly and does not use the function trans-optionslist and its descendants.

```
(300) trans-optionslist(p) =
      ( $\forall q_1, q_2$ ) ( $q_1 \neq q_2 \wedge \text{is-opt-keyw}(q_1, p) \wedge \text{is-opt-keyw}(q_2, p) \Rightarrow$ 
       $\text{optsel} \cdot q_1(t) \neq \text{optsel} \cdot q_2(t)$ )  $\rightarrow$ 
       $\mu_0(\{\langle \text{optsel} \cdot s_n \cdot p(t) : \text{trans-option}(s_n \cdot p) \rangle \mid$ 
       $\neg (\text{is-Q} \vee \text{is-c-file-attribute}) \cdot s_n \cdot p(t)\})$ 
```

```
for: ( $\text{is-c-call-optionslist} \vee \text{is-c-open-optionslist} \vee \text{is-c-close-optionslist} \vee$ 
       $\text{is-c-stream-optionslist} \vee \text{is-c-record-optionslist}) \cdot p(t)$ 
```

Note: Though file-attributes occur as options in open-optionslists they are neither translated nor checked on multiple occurrence by the function trans-optionslist.

```
(301) is-opt-keyw(q, p) =
       $\neg (\text{is-Q} \vee \text{is-c-file-attribute}) \cdot q(t) \wedge (\exists n) (q = s_n \cdot p \vee q = s_1 \cdot s_n \cdot p) \wedge$ 
       $\neg \text{is-**} \cdot \text{optsel}(q)$ 
```

```
(302) opt-sel(opt) =
      is-c-data-specification(opt) --> s-spec
      (is-c-BITSTRING v is-c-STRING)*s1(opt) --> s-string
      is-c-COPY(opt) --> s-copy
      is-c-ENVIRONMENT*s1(opt) --> s-env-attr
      is-c-EVENT*s1(opt) --> s-event
      is-c-FILE*s1(opt) --> s-file
      is-c-FROM*s1(opt) --> s-from
      is-c-IGNORE*s1(opt) --> s-ignore
      is-c-INTO*s1(opt) --> s-into
      (is-c-KEY v is-c-KEYFROM)*s1(opt) --> s-ident
      is-c-KEYTO*s1(opt) --> s-idto
      is-c-LINE*s1(opt) --> s-line
      (is-c-LINESIZE v is-c-BLINESIZE)*s1(opt) --> s-lsz
      is-c-NOLOCK(opt) --> s-nolock
      is-c-PAGE(opt) --> s-page
      is-c-PAGESIZE*s1(opt) --> s-psz
      is-c-PRIORITY*s1(opt) --> s-pri
      is-c-SET*s1(opt) --> s-ptr
      is-c-SKIP*s1(opt) --> s-skip
      is-c-TASK*s1(opt) --> s-task
      is-c-TITLE*s1(opt) --> s-title
      is-c-volume(opt) --> s-volume
      T --> *

(303) trans-cption(p) =
      is-c-data-specification*p(t) --> trans-data-spec(p)
      is-c-identifier*s3*p(t) --> trans-id(s3*p)
      is-c-expression*s3*p(t) --> trans-expr(s3*p)
      is-c-expression*s2*s2*p(t) --> trans-expr(s2*s2*p)
      is-c-ENVIRONMENT*s1*p(t) --> mk-env-attr(s3*p)
      is-c-SKIP*s1*p(t) --> mk-const(<1-CHAR>)
      is-c-TASK*s1*p(t) v (is-c-COPY v is-c-NOLOCK v is-c-PAGE)*p(t) --> *
```

cont'd

30 June 1969

TRANSLATION OF PL/I INTO ABSTRACT SYNTAX

Ref.: trans-id 4-90(351)
 trans-expr 4-81(319)
 mk-env-attr 4-23(82)
 mk-const 4-87(341)

Note: This function translates single options occurring in the call, open, close, stream- and record-I/O statements. The third case (expression) includes in particular those cases, where only a reference is allowed by the syntax.

(304) st-opt-test(type,stmt) =
 is-prop-st(io₁) -- io₁

where:

io₁ = μ(stmt; <s-st:trans-io-st-type(type)>)

Ref.: is-prop-st 5-9(87)

Note: This function tests whether the translated statement satisfies the abstract syntax.

(305) trans-io-st-type(type) =
 is-c-OPEN(type) -- OPEN
 is-c-CLOSE(type) -- CLOSE
 is-c-GET(type) -- GET
 is-c-PUT(type) -- PUT
 is-c-READ(type) -- READ
 is-c-WRITE(type) -- WRITE
 is-c-REWRITE(type) -- REWRITE
 is-c-LOCATE(type) -- LOCATE
 is-c-DELETE(type) -- DELETE
 is-c-UNLOCK(type) -- UNLOCK

(306) trans-open-st(p) =
 slength•s₂•p(t)
 st-opt-test(s₁•p(t), μ₀(<s-list: LIST trans-open-options(s_n•s₂•p)>))
 n=1

for:is-c-open-statement•p(t)

(307) trans-open-options(p) =
 $\mu(\text{trans-optionslist}(p); \langle s\text{-open-attr:fa-set}_1 \rangle, \langle s\text{-blsz:blsz}_1 \rangle)$

where:
 $\text{fa-set}_1 = \{\text{fa}\}$
 $(\exists n)(\text{is-c-file-attribute} \cdot s_n \cdot p(t) \wedge \text{fa} = \text{trans-file-attr} \cdot s_n \cdot p(t))$
 $\text{blsz}_1 = ((\exists n)(\text{is-c-BLINESIZE} \cdot s_1 \cdot s_n \cdot p(t)) \rightarrow *,$
 $T \rightarrow \emptyset)$

for:is-c-open-statement•p(t)

Ref.: trans-file-attr 4-47(188)

(308) trans-close-st(p) =
 $\text{st-opt-test}(s_1 \cdot p(t), \mu_0(\langle s\text{-list: LIST}_{n=1}^{\text{length-}s_1 \cdot p(t)} \text{trans-optionslist}(s_n \cdot s_2 \cdot p) \rangle))$

for:is-c-close-statement•p(t)

(309) trans-stream-io-st(p) =
 $\text{st-opt-test}(s_1 \cdot p(t), \text{trans-stream-options}(p, \text{trans-optionslist}(s_2 \cdot p)))$

for:is-c-stream-io-statement•p(t)

(310) trans-stream-options(p,stmt) =
 $\neg \text{is-0-s-string(stmt)} \rightarrow \mu(\text{stmt}; \langle s\text{-base:bas}_1 \rangle)$
 $\text{is-0-s-file(stmt)} \rightarrow \mu(\text{stmt}; \langle s\text{-file:ref}_1 \rangle)$
 $T \rightarrow \text{stmt}$

where:
 $\text{bas}_1 = ((\exists n)(\text{is-c-BITSTRING} \cdot s_1 \cdot s_n \cdot s_2 \cdot p(t) \rightarrow \text{BIT},$
 $T \rightarrow \text{CHAR}))$
 $\text{ref}_1 = \mu_0(\langle s\text{-id-list:head=ref-id-list}(p) \rangle, \langle s\text{-ap:<>} \rangle, \langle s\text{-sl:<>} \rangle)$

for:is-c-stream-io-statement•p(t)

Ref.: ref-id-list 4-84(330)

30 June 1969

TRANSLATION OF PL/I INTO ABSTRACT SYNTAX

```
(311) trans-data-spec(p) =
      is-c-DATA•s1•p(t) & is-Ω•s2•p(t) --+
          length(dat1)
          μ0(<s-data-list: LIST μ0(<s-id-list:s(n,dat1),<s-sl:<>,<s-ap:<>)>,
                 <s-type:ALL-DATA>)
      is-c-DATA•s1•p(t) -- μ0(<s-data-list:trans-item-list(s2•s3•p)>,<s-type:DATA>)
      is-c-EDIT•s1•p(t) --
          slength•s1•p(t)
          LIST μ0(<s-data-list:trans-item-list(s2•sn•s3•p)>,
                     <s-format-list:trans-format-list(s4•sn•s5•p)>)
      is-c-LIST•s1•p(t) -- μ0(<s-data-list:trans-item-list(s3•p)>,<s-type:LIST>)
```

where:

```
dat1 =
order-set({s-id-list(x) | is-prel-decl(x) & s-block-p(x) = block-p(p) &
           is-PROP-VAR•type-attr(x) & PARAM & s-attr-ds(x) & STRUCT & s-attr-ds(x)},Ω)
```

for:is-c-data-specification•p(t)

Ref.: trans-format-list 4-48(195)
 is-prel-decl 4-10(32)
 block-p 4-4(8)
 type-attr 4-24(87)

Note: Omitted datalists are reconstructed by the Translator. A check of this inserted datalists is performed in the Interpreter.

(312) trans-item-list(p) =

```
      slength•p(t)
      LIST trans-item(sn•p)
                     n=1
```

for:(is-c-datalist v is-Ω)•p(t)

(313) order-set(set,obj) =

Note: cf. /5/.

TRANSLATION OF PL/I INTO ABSTRACT SYNTAX

30 June 1969

```
(314) trans-item(p) =
      is-c-datalist•s1•p(t) --+
      μ0(<s-contr-var:trans-ref(s1•s4•p)>,
            length'sn•sn•p(t)
            <s-spec-list: LISTn=1 .trans-spec(sn•s3•s4•p)>,
            <s-do-list:trans-item-list(s4•p)>
      T --+ trans-expr(p)
```

for:is-c-datalist-element•p(t)

Ref.: trans-ref 4-83(323)
 trans-spec 4-65(253)
 trans-expr 4-81(319)

```
(315) trans-record-io-st(p) =
      is-c-LOCATE•s1•s1•p(t) --+
      st-opt-test(s1•s1•p(t), μ(trans-optionslist(s2•p):<s-id:trans-id(s2•s1•p)>))
      T --+ st-opt-test(s1•p(t), trans-optionslist(s2•p))
```

for:is-c-record-io-statement•p(t)

Ref.: trans-id 4-90(351)

```
(316) trans-display-st(p) =
      μ0(<s-st:DISPLAY>, <s-ident:trans-expr(s3•p)>, <s-idto:mk-reply(s5•p)>,
            <s-event:mk-event(s5•p)>)

      for:is-c-display-statement•p(t)

      Ref.:      trans-expr 4-81(319)
```

```
(317) mk-reply(p) =
      is-c-REPLY•s1•p(t) --+ trans-ref(s3•p)
      T --+ trans-opt-expr(s7•p)
```

Ref.: trans-ref 4-83(323)
 trans-opt-expr 4-82(322)

Note: The function trans-opt-expr also translates optional references.

30 Juhe 1969

TRANSLATION OF PL/I INTO ABSTRACT SYNTAX

```
(318) mk-event(p) =
      is-c-REPLY•s1•p(t) --> trans-opt-expr(s1,•p)
      T --> trans-opt-expr(s3,•p)
      Ref.: trans-opt-expr 4-82(322)
```

4.4 EXPRESSIONS

The concrete syntax already decomposes expressions into their appropriate structure determining the operands of each operator, thereby in particular resolving the precedence rules of operators. By this structuring the following forms of expressions appear:

infix expression: expression operator expression

prefix expression: operator expression

parenthesized expression: LEFT-PAR expression RIGHT-PAR

reference

constant

isub

The translation of expressions is just a one-to-one mapping of this structure. The translation of references which is some more complicated by the insertion of fully qualified names, is described in section 4.4.1. The translation of constants, which has to determine value and data attributes of a concrete constant, is described in section 4.4.2.

Since the translation of references, and thereby the translation of expressions, depends on the block, in which the references occur and since sometimes by the like attribute expressions are to be handled as copied into another block than that of their occurrence, functions env-trans-expr and env-trans-ref are needed, which have a block pointer as additional argument.

```
(319) trans-expr(p) =
      env-trans-expr(block-p(p),p)
      for:is-c-expression•p(t) ∨ is-c-signed-integer•p(t) ∨ is-ASTER•p(t)
```

Ref.: block-p 4-4(8)

TRANSLATION OF PL/I INTO ABSTRACT SYNTAX

30 June 1969

```
(320) env-trans-expr(b,p) =
      is-c-reference•p(t) --> env-trans-ref(b,p)
      is-c-constant•p(t) --> mk-const•p(t)
      is-c-isub•p(t) --> μ0(<s-i:const-val•s1•p(t)>)
      is-ASTER•p(t) --> *
      is-LEFT-PAR•s1•p(t) --> μ0(<s-op:env-trans-expr(b,s1•p)>)
      slength•p(t) = 3 -->
      μ0(<s-operator:inf-operator•s2•p(t)>,<s-op-1:env-trans-expr(b,s1•p)>,
           <s-op-2:env-trans-expr(b,s3•p)>)
      slength•p(t) = 2 --> μ0(<s-operator:s1•p(t)>,<s-op:env-trans-expr(b,s2•p)>)

for:is-c-expression•p(t) ∨ is-ASTER•p(t)
```

Ref.: env-trans-ref 4-83(324)
 mk-const 4-87(341)
 const-val 4-89(347)

Note: Since in some instances (subscripts, string length, area size) an asterisk may appear in the position of an expression, this case is included here.

```
(321) inf-operator(x) =
      is-PLUS(x) --> ADD
      is-MINUS(x) --> SUBTR
      is-ASTER(x) --> MULT
      is-SLASH(x) --> DIV
      x = <OR,OR> --> CAT
      x = <ASTER,ASTER> --> EXP
      x = <GT,EQ> ∨ x = <NOT,LT> --> GE
      x = <LT,EQ> ∨ x = <NOT,GT> --> LE
      x = <NOT,EQ> --> ND
      T --> x
```

```
(322) trans-opt-expr(p) =
      is-Ω•p(t) --> Ω
      T --> trans-expr(p)
```

4.4.1 REFERENCES

The translation of basic references (and of unsubscripted references as well) is performed in the following steps:

30 June 1969

TRANSLATION OF PL/I INTO ABSTRACT SYNTAX

- (1) Collect the identifier list and argument list of the reference text.
- (2) Try to find a preliminary declaration of an explicitly specified declaration matching the identifier list of the reference. If none is found, make a preliminary declaration from the reference itself (cf. 4.2.1.4, contextual declarations).
- (3) Replace the identifier list of the reference by the identifier list of the preliminary declaration.

The searching for a matching preliminary declaration is performed by the function expl-decl-ref. The language requires the following algorithm: First look in the current block for a declaration having exactly the same identifier list, second look in the current block for a declaration having a matching identifier list, third repeat the same in the next surrounding block, and so on, until one either finds a matching declaration or has inspected the containing external procedure without success. In the latter case only the entry identifiers of the containing external procedure and not of other parallel external procedures are to be inspected.

For the function env-trans-ref(b,p) which assumes the reference to be copied into the block pointed to by b, the searching process starts with this block.

Since for the recognition of contextual declarations (cf. 4.2.1.4) not only references, but also single identifiers (e.g. the entry identifier in a call statement) have to be inspected, the functions determining the matching preliminary declaration are defined also for these cases.

Metavariables

```

idl      is-id-list(idl)
b       is-c-block•b(t)
pd     is-prel-decl(pd)

```

(323) trans-ref(p) =
 env-trans-ref(block-p(p),p)

for:is-c-reference•p(t) • is-ref-cont(p)

Ref.: block-p 4-4(8)

(324) env-trans-ref(b,p) =
 is-c-reference•p(t) -->
 μ(env-trans-ref(b,s₁•p);<s₂:env-trans-ref(b,s₁•s₁•p)>)
 is-ref-cont(p) --> <s₃:env-trans-subscr-list(b,p)>,
 μ<<s₄:id-list:s₅:id-list•decl-ref(b,p)>,<s₆:env-trans-subscr-list(b,p)>,
 <s₇:ap:env-trans-arg-list(b,p)>>

(325) is-ref-cont(p) =
 is-c-basic-reference•p(t) • is-c-unsubscripted-reference•p(t)

(326) decl-ref(b,p) =
 -is-Q*expl-decl-ref(b,ref-id-list(p)) -- expl-decl-ref(b,ref-id-list(p))
 is-non-expl-decl(p) -- prel-decl(p)

Ref.: is-non-expl-decl 4-16(57)
 prel-decl 4-9(30)

(327) expl-decl-ref(b,idl) =
 E -- (Upd)(is-epd(b,pd) & idl = s-id-list(pd))
 E -- (Upd)(is-epd(b,pd) & is-id-list-match(idl,s-id-list(pd)))
 is-ext-proc-p(b) &
 ~(Exp)(is-entry-cont(p) & idl = <mk-id-1(p)> & proc-p(p) = b) --
 Q
 T -- expl-decl-ref(block-p(b),idl)

Ref.: is-ext-proc-p 4-4(14)
 is-entry-cont 4-11(40)
 mk-id-1 4-90(353)
 proc-p 4-4(12)
 block-p 4-4(8)

(328) is-epd(b,pd) =
 pd ∈ prel-decl-set(b) & (Exseq)(is-succ-seq(seq) & pd = prel-decl(seq))

Ref.: prel-decl-set 4-9(28)
 is-succ-seq 4-12(41)
 prel-decl 4-9(30)

(329) is-id-list-match(idl-1,idl-2) =
 is-<>(idl-1) & is-<>(idl-2) -- T
 is-<>(idl-1) v is-<>(idl-2) -- F
 head(idl-1) = head(idl-2) -- is-id-list-match(tail(idl-1),tail(idl-2))
 T -- is-id-list-match(idl-1,tail(idl-2))

(330) ref-id-list(p) =
 is-sysin-cont(p) -- <mk-id((Lx)(is-c-SYSIN(x)))>
 is-sysprint-cont(p) -- <mk-id((Lx)(is-c-SYSPRINT(x)))>
 length:id-p-list(p)
 T -- LIST mk-id-1*elem(n,id-p-list(p))
 n=1
 for:(is-ref-cont v is-single-id-cont v is-sysin-cont v is-sysprint-cont)(p) v
 is-c-identifier*p(t)

cont'd

30 June 1969

TRANSLATION OF PL/I INTO ABSTRACT SYNTAX

Ref.: is-sysin-cont 4-16(60)
 mk-id 4-90(354)
 is-sysprint-cont 4-16(61)
 mk-id-1 4-90(353)
 is-single-id-cont 4-90(355)

(331) id-p-list(p) =

$$\begin{aligned} \text{is-c-basic-reference-p(t)} &\rightarrow \underset{n=1}{\overset{\lg_1}{\text{LIST}}} s_1 \cdot s_n \cdot p \\ \text{is-c-unsubscripted-reference-p(t)} &\rightarrow \underset{n=1}{\overset{\lg_1}{\text{LIST}}} s_n \cdot p \\ \text{is-c-identifier-p(t)} &\rightarrow \langle p \rangle \end{aligned}$$

where:

 $\lg_1 = \text{slength-p}(t)$

(332) env-trans-subscr-list(b,p) =

$$\underset{n=1}{\overset{\lg_1}{\text{LIST}}} \text{env-trans-expr}(b, \text{elem}(n, \text{subscr-p-list}(p)))$$

where:

 $\lg_1 = \text{subscr-p-list}(p)$

for:is-ref-cont(p)

Ref.: env-trans-expr 4-82(320)

(333) subscr-p-list(p) =

$$\begin{aligned} \text{is-c-basic-reference-p(t)} &\rightarrow \\ \text{collect}(\{q \mid (\text{is-c-expression-q}(t) \vee \text{is-ASTER-q}(t)) \wedge s_1 \cdot p \Rightarrow q \wedge \\ &\quad \neg(\exists r) (\text{is-c-expression-r}(t) \wedge s_1 \cdot p \Rightarrow r \Rightarrow q) \vee \text{is-rest-subscr}(p, q)\}) \end{aligned}$$

is-c-unsubscripted-reference-p(t) → < >

for:is-ref-cont(p)

(334) is-rest-subscr(p,q) =

 $\text{missing-subscrs}(p) = 0 \vee \text{is-Q-s}_3 \cdot p(t) \rightarrow F$ $\text{slength-s}_2 \cdot s_1 \cdot s_3 \cdot p(t) \leq \text{missing-subscrs}(p) \rightarrow$ $(\exists n) (q = s_n \cdot s_2 \cdot s_1 \cdot s_3 \cdot p \wedge \neg \text{is-Q-q}(t))$

for:is-ref-cont(p)

Note: The concrete syntax does not allow to recognize if the first parenthesized expression following the rightmost identifier in a reference is a

TRANSLATION OF PL/I INTO ABSTRACT SYNTAX

30 June 1969

subscript- or an argumentlist. The number of subscripts is therefore compared with the declared number of dimensions.

(335) missing-subscr(p) =
 $\sum_{n=1}^{\text{length}(pd_1)} \text{dimensionality} = s - \text{attr} - ds - \text{prel-decl}(pd_1) - \text{subscr-nb}(s_1 * p, 1)$

where:
 $pd_1 = \text{pd-seq-decl-ref}(\text{block-p}(p), p)$

for:is-ref-cont(p)

Ref.: prel-decl 4-9(30)
pd-seq 4-31(113)
block-p 4-4(8)

(336) dimensionality(ad-set) =
 $\exists \rightarrow s\text{length} * s_2 * ((\forall ad) (ad \in ad-set \wedge \text{is-c-dimension-attribute} * ad(t))) (t)$
 $T \rightarrow 0$

(337) subscr-nb(p,n) =
 $\text{is-0} * s_n * p(t) \rightarrow 0$
 $T \rightarrow s\text{length} * s_2 * s_n * p(t) + \text{subscr-nb}(p, n + 1)$

(338) env-trans-arg-list(b,p) =
 $(\exists q) (\text{is-rest-subscr}(p,q)) \rightarrow \text{arg-p-list}(b, s_3 * p, 2)$
 $T \rightarrow \text{arg-p-list}(b, s_3 * p, 1)$

for:is-ref-cont(p)

(339) arg-p-list(b,p) =
 $\text{is-0} * s_n * p(t) \rightarrow \langle \rangle$
 $T \rightarrow \langle \text{prop-arg-p-list}(b, s_2 * s_n * p, 1) \rangle^* \text{arg-p-list}(b, p, n + 1)$

(340) prop-arg-p-list(b,p) =
 $\text{is-0} * s_n * p(t) \rightarrow \langle \rangle$
 $\text{is-ASTER} * s_n * p(t) \rightarrow \text{error}$
 $T \rightarrow \langle \text{env-trans-expr}(b, s_n * p) \rangle^* \text{prop-arg-p-list}(b, p, n + 1)$

Ref.: env-trans-expr 4-82(320)

30 June 1969

TRANSLATION OF PL/I INTO ABSTRACT SYNTAX

4.4.2 CONSTANTS

Constants including signed integers and the real and imaginary part of arithmetic initial constants are translated into operands by the function mk-const. One step of the translation is the computation of the apparent attributes which are bounded by implementation-defined limits, the other step is the computation of the value.

Metavariables

x	is-c-constant v is-c-constant*s ₂ v is-c-integer v is-c-signed-integer	constant
da	is-arithm & is-c-REAL*s-mode	data attribute for a real value
v	is-real-val	real value

- (341) mk-const(x) =
 $\mu_0(\langle s\text{-}da:\text{const}\text{-}da(x)\rangle, \langle s\text{-}v:\text{const}\text{-}val(x)\rangle)$
- (342) const-da(x) =
 is-c-constant*s(2,x) v is-c-signed-integer(x) --> const-da*s(2,x)
 is-c-integer(x) --> const-da($\mu_0(\langle \text{elem}_1\text{:}\text{elem}_1:x\rangle)$)
 is-c-real-constant(x) --> rest-da-1(app-da(elem(1,x),bs₁),real*const-val(x))
 is-c-imaginary-constant(x) --> $\mu(\text{const}\text{-}da\text{-}\text{elem}(1,x);\langle s\text{-mode:CPLX}\rangle)$
 is-c-sterling-constant(x) --> $\mu(\text{const}\text{-}da(x_5);\langle s\text{-prec:lg}_1 + lg\text{-}\text{elem}_3\text{-}\text{elem}(5,x)\rangle)$
 is-c-replicated-string-constant(x) v is-c-simple-string-constant(x) -->
 $\mu_0(\langle s\text{-base:}\text{strb}_1\rangle, \langle s\text{-length:}\text{strlg}_1\rangle)$

where:

bs₁ = {is-B-CHAR*elem(2,x) --> BIN,
 T --> DEC}
 lg₁ = {(Un)(10 + (n - 1) ≤ const-val(x) < 10 + n)}
 x₅ = $\mu_0(\langle \text{elem}_1\text{:}\text{elem}(5,x)\rangle)$
 str₁ = {is-c-simple-string-constant(x) --> x,
 T --> s(4,x)}
 strb₁ = {is-c-bit-string(str₁) --> BIT,
 T --> CHAR}
 strlg₁ = {is-c-simple-string-constant(x) --> lg*elem(2,str₁),
 T --> const-val*s(2,x) . lg*elem(2,str₁)}

Note: The precision of the resulting data attribute is not greater than a maximum precision and will accommodate the value. There is no restriction assumed for the length of string constants and for the length of the exponent in floating point constants.

```

(343) app-da(z,bs) =
      is-c-fixed-constant(z) --+
      μo(<s-mode:REAL>,<s-base:bs>,<s-scale:FIX>,
          <s-prec:lg*elem(1,z) + lg*elem(3,z)>,<s-scale-f:lg*elem(3,z)>)
      is-c-float-constant(z) --+ μ(δ(app-da(elem(1,z),bs);s-scale-f);<s-scale:FLT>)

      for:(is-c-fixed-constant(z) ∨ is-c-float-constant(z)) & (is-BIN(bs) ∨ is-DEC(bs))

(344) rest-da-1(da,v) =
      ~is-size-cond(da1,v) -- da1

      where:
      da1 = μ(da;<s-prec:min(max-prec(da),s-prec(da))>)

(345) max-prec(da) =
      Note: cf. /5/.
```

(346) is-size-cond(da,v) =
 Note: cf. /5/.

(347) const-val(x) =
 is-c-constant-s(2,x) ∨ is-c-signed-integer(x) --+ sgn*s(1,x) . const-val-s(2,x)
 is-c-integer(x) --+ intg-val(x,10)
 is-c-fixed-constant(x₁) --+
 intg-val(elem(1,x₁),bs₁) + intg-val(elem(3,x₁),bs₁) . bs₁ + (-lg*elem(3,x₁))
 is-c-float-constant(x₁) --+
 const-val(elem(1,x₁),bs₁) . bs₁ + (sgn*elem(3,x₁) . const-val*elem(4,x₁))
 is-c-imaginary-constant(x) --+ cplx(0,const-val*elem(1,x))
 is-c-sterling-constant(x) & const-val*elem(3,x) < 20 & const-val(x₅) < 12 --+
 240 . const-val*elem(1,x) + 12 . const-val*elem(3,x) + const-val(x₅)
 is-c-character-string(x) --+ LIST_{n=1} char₁
 is-c-bit-string(x) --+ LIST_{n=1} bit₁
 is-c-replicated-string-constant(x) --+ CONC_{n=1} const-val-s(2,x)

cont'd

30 June 1969

TRANSLATION OF PL/I INTO ABSTRACT SYNTAX

```

where: bs1 = (is-B-CHAR•elem(2,x) --> 2,
      T --> 10)
      xn = elemn•elem(2,x)
      chari = (xn = <APOSTR,APOSTR> --> APOSTR,
                 T --> xn)
      biti = (is-0-CHAR(xn) --> 0-BIT,
                 T --> 1-BIT)
      x1 = elem(1,x)
      x5 = μ0(<elem1:elem(5,x)>)

```

```

(348) intg-val(y,bs) =
      is-Ω(y) ∨ is-0-CHAR(y) --> 0
      is-1-CHAR(y) --> 1
      is-c-digit(y) & bs = 2 -- error
      is-2-CHAR(y) --> 2
      is-3-CHAR(y) --> 3
      is-4-CHAR(y) --> 4
      is-5-CHAR(y) --> 5
      is-6-CHAR(y) --> 6
      is-7-CHAR(y) --> 7
      is-8-CHAR(y) --> 8
      is-9-CHAR(y) --> 9
      length(y)
      is-c-integer --> Σn=1length(y) intg-val(elem(n,y),bs) . bs + (length(y) - n)

```

for:(is-Ω(y) ∨ is-c-digit(y) ∨ is-c-integer(y)) & (bs = 2 ∨ bs = 10)

```

(349) lg(u) =
      is-list(u) --> length(u)
      T --> 0

```

```

(350) sgn(u) =
      is-MINUS(u) --> -1
      T --> 1

```

4.5 IDENTIFIERS

Identifiers are translated by applying the function mk-id to the character lists representing them in the concrete program text. This function maps one-to-one character lists into elementary objects satisfying the predicate is-id, i.e. it maps one-to-one "concrete identifiers" (and other character lists, used in this document for initial labels, too) into corresponding "abstract identifiers".

To translate a pointer of an identifier, one has first to generate the corresponding character list by the functions defined in chapter 2 and then to apply the function mk-id.

In many instances, identifiers have to be handled like references; their corresponding preliminary declarations have to be determined; if none exist, contextual declarations have to be created; instead of replacing an identifier list as in section 4.4.1, it has to be tested that the corresponding preliminary declaration is at level one.

```
(351) trans-id(p) =
      env-trans-id(block-p(p), p)

      for:is-c-identifier•p(t)

      Ref.: block-p 4-4(8)
```

```
(352) env-trans-id(b,p) =
      length•s-id-list•decl-ref(b,p) = 1 -- mk-id-1(p)

      for:is-c-block•b(t), is-c-identifier•p(t)

      Ref.: decl-ref 4-84(326)
```

```
(353) mk-id-1(p) =
      is-c-identifier•p(t) -- gen1
      is-<>•s-ap(ref1) &
      ( $\forall n$ ) ( $\neg$ is-Q•sn•s-sl(ref1)  $\Rightarrow$   $\neg$ is-*•descr-ext( $\Omega$ , sn•s-sl(ref1))) -- gen1

      where:
      gen1 = mk-id•generate-2•generate-1•p(t)
      ref1 = trans-ref(p)
```

```
for:(is-c-identifier  $\vee$  is-c-basic-reference)•p(t)

Ref.: descr-ext 4-43(176)
      generate-2 2-3(4)
      generate-1 2-3(3)
      trans-ref 4-83(323)
```

```
(354) mk-id(chl) =
```

Note: cf. /5/.

```
(355) is-single-id-cont(p) =
      is-c-identifier•p(t) & ( $\exists n, q$ ) (is-c-attention-condition•q(t) & p = sn•s3•p  $\vee$ 
      is-c-access-statement•q(t) &  $\neg$ is-Q•s3•q(t) & p = sn•s2•s3  $\vee$ 
      is-c-LOCATE•s1•s1•q(t) & p = s2•s1•p  $\vee$  is-c-CONDITION•s1•q(t) & p = s3•q)
```

30 June 1969

TRANSLATION OF PL/I INTO ABSTRACT SYNTAX

Note: This predicate lists all those contexts of single identifiers which, if no matching explicit declarations exist, lead to non-explicit declarations. There are some other contexts of single identifiers (i.e. identifiers not in references and not in their own declaration) in the language, e.g. in the label attribute, or in allocate items; but in those cases there has to be a corresponding explicit declaration.

30 June 1969

TRANSLATION OF PL/I INTO ABSTRACT SYNTAX

5. ABSTRACT SYNTAX

This chapter defines the predicate `is-proper-program`, giving the syntactic structure of abstract programs produced by the function `translate` defined in chapter 4.

It should be noted that not each abstract program, i.e. not each object satisfying the predicate `is-proper-program`, may be produced by the function `translate` from a concrete program. The range of the `translate` function is a subset of the set of all abstract programs.

5.1 PROGRAM, PROCEDURE BODY

- (1) `is-proper-program` =
`(<s-decl-part:is-decl-part>,`
`<s-body-part:is-body-part>)`
- (2) `is-body-part` =
`({<id:is-body> | | is-id(id)})`
- (3) `is-body` =
`(<s-entry-part:is-entry-part>,`
`<s-decl-part:is-decl-part>,`
`<s-body-part:is-body-part>,`
`<s-cond-part:is-cond-part>,`
`<s-st-list:is-st-list-1>,`
`<s-reorder:is-opt>,`
`<s-recursive:is-opt>)`
- (4) `is-entry-part` =
`({<id:is-entry-point> | | is-id(id)})`
- (5) `is-entry-point` =
`(<s-st-loc:is-index-list-1>,`
`<s-param-list:is-id-ref-list>,`
`<s-ret-type:is-descr-scalar>)`
- (6) `is-id-ref` =
`{<s-id:is-id>}`
- (7) `is-cond-part` =
`(<s-on:is-prefix-cond-list>,`
`<s-no:is-prefix-cond-list>)`
- (8) `is-opt` =
`is-* v is-0`

5.2 DECLARATIONS

5.2.1 GENERAL

```
(9)    is-decl-part =
        ({<id:is-decl> | | is-id(id)})  

(10)   is-decl =
        is-prop-var v is-defined v is-based v is-entry-const v is-file-const v
        is-label-const v is-format-const v is-generic v is-BUILTIN v is-COND v is-attn  

(11)   is-prop-var =
        (<s-scope:is-INT v is-EXT v is-PARAM>,
         <s-stg-cl:is-AUTO v is-STATIC v is-CTL v is-Q>,
         <s-aggr:is-prop-aggr>,
         <s-connected:is-opt>)  

(12)   is-defined =
        (<s-base:is-ref>,
         <s-aggr:is-prop-aggr>,
         <s-pos:is-opt-expr>)  

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

(14)   is-entry-const =
        (<s-scope:is-INT v is-EXT>,
         <s-descr-list:is-descr-list v is-*>,
         <s-ret-type:is-descr-scalar>,
         <s-body:is-id v is-Q>,
         <s-reducible:is-opt>)  

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

(16)   is-file-const =
        (<s-scope:is-INT v is-EXT>,
         <s-file-attr:is-file-attr-set>,
         <s-env-attr:is-env-attr>)
```

30 June 1969

TRANSLATION OF PL/I INTO ABSTRACT SYNTAX

```

(17)  is-file-attr =
      is-CST v is-BST v is-REC v is-INP v is-OUT v is-UPD v is-SEQ v is-TRA v
      is-DIR v is-BUF v is-UNB v is-PRT v is-BAC v is-EXC v is-KEY

(18)  is-env-attr =
      Note: This predicate is implementation defined.

(19)  is-label-const =
      is-index-list-1

(20)  is-index =
      is-intg-val v is-T v is-F

(21)  is-format-const =
      (<s-cond-part:is-cond-part>,
       <s-format-list:is-format-list-1>,
       <s-ident:is-id>)

(22)  is-generic =
      is-generic-member-list-1

(23)  is-generic-member =
      (<s-entry:is-ref>,
       <s-descr-list:is-generic-descr-list>)

(24)  is-generic-descr =
      (<s-stg-cl:is-CTL v is-Q>,
       <s-aggr:is-generic-aggr>)

(25)  is-attn =
      (<s-env-attr:is-env-attr>)

```

5.2.2 AGGREGATE ATTRIBUTES

5.2.2.1 Aggregates in variable declarations

```

(26)  is-prop-aggr =
      is-prop-array v is-prop-struct v is-prop-scalar

(27)  is-prop-array =
      (<s-lbd:is-expr v is-* v is-refer>,
       <s-ubd:is-expr v is-* v is-refer>,
       <s-elem:is-prop-aggr>)

```

```

(28)  is-refer =
      (<s-expr:is-expr>,
       <s-refer:is-id-list>)

(29)  is-prop-struct =
      is-prop-succ-list-1

(30)  is-prop-succ =
      (<s-qual:is-id>,
       <s-aggr:is-prop-aggr>)

(31)  is-prop-scalar =
      (<s-da:is-prop-da>,
       <s-dens:is-AL v is-UNAL>,
       <s-init:is-init>)

(32)  is-prop-da =
      is-prop-arithm v is-prop-string v is-pic v is-prop-area v is-entry-da v
      is-label-da v is-offset v is-PTR v is-FILE v is-TASK v is-EVENT

(33)  is-prop-arithm =
      (<s-mode:is-REAL v is-CPLX>,
       <s-base:is-DEC v is-BIN>,
       <s-scale:is-FLT v is-7IX>,
       <s-prec:is-intg-val>,
       <s-scale-f:is-intg-val v is-0>)

(34)  is-prop-string =
      (<s-base:is-CHAR v is-BIT>,
       <s-length:is-expr v is-* v is-refer>,
       <s-varying:is-VAR v is-0>)

(35)  is-prop-area =
      (<s-size:is-expr v is-* v is-refer>)

(36)  is-entry-da =
      (<s-descr-list:is-descr-list v is-*>,
       <s-ret-type:is-descr-scalar>,
       <s-reducible:is-opt>)

(37)  is-label-da =
      is-LABEL v
      (<s-label-list:is-id-ref-list>)

```

30 June 1969

TRANSLATION OF PL/T INTO ABSTRACT SYNTAX

```
(38)  is-offset =
      is-OFFSET v
      (<s-area:is-ref>)

(39)  is-init =
      is-init-elem-list v is-call-st v is-spec-init-elem-list

(40)  is-init-elem =
      is-init-iter v is-expr v is-*

(41)  is-init-iter =
      (<s-rep:is-expr>,
       <s-init:is-init-elem-list-1>)

(42)  is-spec-init-elem =
      (<s-sl:is-intg-val-list-1>,
       <s-id:is-id>)
```

5.2.2.2 Aggregates in parameter descriptors

```
(43)  is-descr-aggr =
      is-descr-array v is-descr-struct v is-descr-scalar

(44)  is-descr-array =
      (<s-lbd:is-intg-val v is-*>,
       <s-ubd:is-intg-val v is-*>,
       <s-elem:is-descr-aggr>)

(45)  is-descr-struct =
      is-descr-succ-list-1

(46)  is-descr-succ =
      (<s-aggr:is-descr-aggr>)

(47)  is-descr-scalar =
      (<s-da:is-descr-da>,
       <s-dens:is-AL v is-UNAL>)

(48)  is-descr-da =
      is-prop-arithm v is-descr-string v is-pic v is-descr-area v is-entry-da v
      is-LABEL v is-offset v is-PTR v is-FILE v is-TASK v is-EVENT
```

(49) is-descr-string =
 (<s-base:is-CHAR v is-BIT>,
 <s-length:is-intg-val v is-*>,
 <s-varying:is-VAR v is-0>)

(50) is-descr-area =
 (<s-size:is-intg-val v is-*>)

5.2.2.3 Aggregates in generic parameter descriptors

(51) is-generic-aggr =
 is-generic-array v is-generic-struct v is-generic-scalar

(52) is-generic-array =
 (<s-elem:is-generic-aggr>)

(53) is-generic-struct =
 is-generic-succ-list-1

(54) is-generic-succ =
 (<s-aggr:is-generic-aggr>)

(55) is-generic-scalar =
 (<s-da:is-generic-da v is-*>,
 <s-dens:is-AL v is-UNAL v is-0>)

(56) is-generic-da =
 is-generic-arith v is-generic-string v is-pic v is-AREA v is-ENTRY v
 is-LABEL v is-OFFSET v is-PTR v is-FILE v is-TASK v is-EVENT

(57) is-generic-arith =
 (<s-mode:is-REAL v is-CPLX v is-0>,
 <s-base:is-DEC v is-BIN v is-0>,
 <s-scale:is-FLT v is-FIX v is-0>,
 <s-prec:is-intg-val v is-0>,
 <s-scale-f:is-intg-val v is-0>)

(58) is-generic-string =
 (<s-base:is-CHAR v is-BIT v is-0>,
 <s-varying:is-VAR v is-0>)

30 June 1969

TRANSLATION OF PL/I INTO ABSTRACT SYNTAX

5.2.2.4 Aggregates in allocate statements

```
(59)  is-al-aggr =
      is-al-array ∨ is-al-struct ∨ is-al-scalar

(60)  is-al-array =
      (<s-lbd:is-expr ∨ is-*>,
       <s-ubd:is-expr ∨ is-*>,
       <s-elem:is-al-aggr>)

(61)  is-al-struct =
      is-al-succ-list-1

(62)  is-al-succ =
      (<s-aggr:is-al-aggr>)

(63)  is-al-scalar =
      (<s-da:is-al-string ∨ is-al-area ∨ is-0>,
       <s-init:is-init-elem-list ∨ is-call-st>)

(64)  is-al-string =
      (<s-base:is-CHAR ∨ is-BIT>,
       <s-length:is-expr ∨ is-*>)

(65)  is-al-area =
      (<s-size:is-expr ∨ is-*>)
```

5.2.3 PICTURES

```
(66)  is-pic =
      is-dec-pic ∨ is-sterling-pic ∨ is-bin-pic ∨ is-char-pic

(67)  is-dec-pic =
      (<s-mode:is-CPLX ∨ is-0>,
       <s-mt-field:is-dec-spec-list-1>,
       <s-mt-unit:is-intg-val ∨ is-0>,
       <s-scale-f:is-intg-val ∨ is-0>,
       <s-exp-sep:is-E-CHAR ∨ is-0>,
       <s-exp-field:is-dec-spec-list-1 ∨ is-0>)

(68)  is-dec-spec =
      is-9-CHAR ∨ is-Z-CHAR ∨ is-ASTER ∨ is-Y-CHAR ∨ is-T-CHAR ∨ is-I-CHAR ∨
      is-R-CHAR ∨ is-SIGN ∨ is-PLUS ∨ is-MINUS ∨ is-DOLLAR ∨ is-BLANK ∨ is-COMMA ∨
      is-SLASH ∨ is-POINT ∨ is-C-CHAR ∨ is-D-CHAR ∨ is-B-CHAR
```

```

(69)  is-sterling-pic =
      (<s-mt-field:is-sterling-spec-list-1>,
       <s-stat-part-end:is-intg-val>,
       <s-pound-end:is-intg-val>,
       <s-shill-end:is-intg-val>,
       <s-mt-unit:is-intg-val v is-0>)

(70)  is-sterling-spec =
      is-dec-spec v is-6-CHAR v is-7-CHAR v is-8-CHAR v is-S-CHAR

(71)  is-bin-pic =
      (<s-mode:is-CPLX v is-0>,
       <s-mt-field:is-bin-spec-list-1>,
       <s-mt-unit:is-intg-val v is-0>,
       <s-scale-f:is-intg-val v is-0>,
       <s-exp-field:is-bin-spec-list-1 v is-0>)

(72)  is-bin-spec =
      is-1-CHAR v is-2-CHAR v is-3-CHAR v is-SIGN

(73)  is-char-pic =
      (<s-field:is-char-spec-list-1>)

(74)  is-char-spec =
      is-X-CHAR v is-A-CHAR v is-9-CHAR v is-ASTER v is-BLANK v is-COMMA v is-SLASH v
      is-POINT

```

5.2.4 FORMATS

```

(75)  is-format =
      is-format-iter v is-format-item

(76)  is-format-iter =
      (<s-rep:is-expr>,
       <s-format-list:is-format-list-1>)

(77)  is-format-item =
      is-data-format v is-control-format v is-remote-format

(78)  is-data-format =
      is-real-format v is-cplx-format v is-string-format v is-pic-format

```

30 June 1969

TRANSLATION OF PL/I INTO ABSTRACT SYNTAX

```

(79)  is-real-format =
      (<s-type:is-FLT v is-FIX>,
       <s-w:is-expr>,
       <s-d:is-expr>,
       <s-p-s:is-opt-expr>)

(80)  is-cplx-format =
      (<s-type:is-CPLX>,
       <s-real:is-real-format v is-pic-format>,
       <s-imag:is-real-format v is-pic-format>)

(81)  is-string-format =
      (<s-type:is-CHAR v is-BIT v is-BBIT>,
       <s-w:is-opt-expr>)

(82)  is-pic-format =
      (<s-type:is-PIC v is-BPIC>,
       <s-pic:is-pic>)

(83)  is-control-format =
      (<s-type:is-control-format-type>,
       <s-w:is-opt-expr>)

(84)  is-control-format-type =
      is-SPACE v is-BSPACE v is-COL v is-BCOL v is-LINE v is-PAGE v is-SKIP

(85)  is-remote-format =
      (<s-type:is-REMOTE>,
       <s-ref:is-ref>)

```

5.3 STATEMENTS

```

(86)  is-st =
      (<s-cond-part:is-cond-part>,
       <s-label-list:is-id-ref-list>,
       <s-prop-st:is-prop-st>)

(87)  is-prop-st =
      is-block v is-group v is-st-list-1 v is-if-st v is-goto-st v is-call-st v
      is-return-st v is-incorporate-st v is-fetch-st v is-release-st v is-wait-st v
      is-delay-st v is-exit-st v is-stop-st v is-assign-st v is-allocate-st v
      is-free-st v is-on-st v is-revert-st v is-signal-st v is-access-st v
      is-enable-st v is-disable-st v is-open-st v is-close-st v is-get-st v
      is-put-st v is-read-st v is-write-st v is-rewrite-st v is-locate-st v
      is-delete-st v is-unlock-st v is-display-st v is-null-st

```

(88) is-null-st =
 (<s-st:is-NUL>)

5.3.1 BLOCK, GROUP

(89) is-block =
 (<s-decl-part:is-decl-part>,
 <s-body-part:is-body-part>,
 <s-cond-part:is-cond-part>,
 <s-st-list:is-st-list-1>,
 <s-reorder:is-opt>)

(90) is-group =
 is-while-group v is-contr-group

(91) is-while-group =
 (<s-while:is-expr>,
 <s-do-list:is-st-list-1>)

(92) is-contr-group =
 (<s-contr-var:is-ref>,
 <s-spec-list:is-do-spec-list-1>,
 <s-do-list:is-st-list-1>)

(93) is-do-spec =
 (<s-init:is-expr>,
 <s-by:is-opt-expr>,
 <s-to:is-opt-expr>,
 <s-while:is-opt-expr>)

5.3.2 FLOW OF CONTROL STATEMENTS

(94) is-if-st =
 (<s-st:is-IF>,
 <s-expr:is-expr>,
 <s-then:is-st>,
 <s-else:is-st>)

(95) is-goto-st =
 (<s-st:is-GOTO>,
 <s-label:is-ref>)

30 June 1969

TRANSLATION OF PL/I INTO ABSTRACT SYNTAX

(96) is-call-st =
 (<s-st:is-CALL>,
 <s-entry:is-ref>,
 <s-arg-list:is-expr-list>,
 <s-pa-opt:is-pa-opt v is-0>)

(97) is-pa-opt =
 (<s-task:is-ref v is-*>,
 <s-event:is-ref v is-*>,
 <s-pri:is-opt-expr>)

(98) is-return-st =
 (<s-st:is-RETURN>,
 <s-expr:is-opt-expr>)

(99) is-incorporate-st =
 (<s-st:is-INCORPORATE>,
 <s-text:is-incorporate-text>)

(100) is-incorporate-text =

Note: This predicate is implementation defined.

5.3.3 STORAGE MANIPULATING STATEMENTS

(101) is-fetch-st =
 (<s-st:is-FETCH>,
 <s-entry-list:is-ref-list-1>)

(102) is-release-st =
 (<s-st:is-RELEASE>,
 <s-entry-list:is-ref-list-1>)

(103) is-wait-st =
 (<s-st:is-WAIT>,
 <s-event-list:is-ref-list-1>,
 <s-event-number:is-opt-expr>)

(104) is-delay-st =
 (<s-st:is-DELAY>,
 <s-time:is-expr>)

(105) is-exit-st =
 (<s-st:is-EXIT>)

```

(106)  is-stop-st =
       (<s-st:is-STOP>)

(107)  is-assign-st =
       (<s-st:is-ASSIGN>,
        <s-lp:is-ref-list-1>,
        <s-rp:is-expr>,
        <s-bname:is-opt>)

(108)  is-allocate-st =
       (<s-st:is-ALLOCATE>,
        <s-list:is-al-list-1>)

(109)  is-al =
       (<s-id:is-id>,
        <s-aggr:is-al-aggr ∨ is-*>,
        <s-ptr:is-opt-ref>,
        <s-area:is-opt-ref>)

(110)  is-free-st =
       (<s-st:is-FREE>,
        <s-list:is-free-list-1>)

(111)  is-free =
       (<s-ref:is-ref>,
        <s-area:is-opt-ref>)

```

5.3.4 CONDITION AND ATTENTION HANDLING STATEMENTS

```

(112)  is-on-st =
       (<s-st:is-ON>,
        <s-cond:is-cond>,
        <s-snap:is-opt>,
        <s-on-unit:is-st ∨ is-SYSTEM>)

(113)  is-revert-st =
       (<s-st:is-REVERT>,
        <s-cond:is-cond>)

(114)  is-signal-st =
       (<s-st:is-SIGNAL>,
        <s-cond:is-cond>)

(115)  is-cond =
       is-prefix-cond ∨ is-AREA ∨ is-named-io-cond ∨ is-ERROR ∨ is-FINISH ∨
       is-progr-named-cond ∨ is-attn-cond

```

30 June 1969

TRANSLATION OF PL/I INTO ABSTRACT SYNTAX

(116) is-prefix-cond =
 is-CONV v is-TOTL v is-OFL v is-SIZE v is-STRG v is-STRZ v is-SUBRG v is-UFL v
 is-ZDIV v is-check-cond

(117) is-check-cond =
 (<s-ref-list:is-ref-list-1>,
 <s-cond:is-CHECK>)

(118) is-named-io-cond =
 (<s-ref:is-ref>,
 <s-cond:is-io-cond>)

(119) is-io-cond =
 is-BOV v is-EOV v is-ENDP v is-ENDP v is-KEY v is-NAME v is-REC v is-TMT v
 is-UNDF v is-PEND

(120) is-progr-named-cond =
 (<s-id:is-id>,
 <s-cond:is-COND>)

(121) is-attn-cond =
 (<s-attn-list:is-id-ref-list>,
 <s-cond:is-ATTN>)

(122) is-access-st =
 (<s-st:is-ACCESS>,
 <s-cond:is-attn-cond>,
 <s-else:is-st v is-Q>)

(123) is-enable-st =
 (<s-st:is-ENABLE>,
 <s-list:is-enable-list-1>)

(124) is-enable =
 (<s-cond:is-attn-cond>,
 <s-spec:is-ACC v is-ASYN>,
 <s-event:is-opt-ref>)

(125) is-disable-st =
 (<s-st:is-DISABLE>,
 <s-cond:is-attn-cond>)

5.3.5 INPUT AND OUTPUT STATEMENTS

```

(126) is-open-st =
      (<s-st:is-OPEN>,
       <s-list:is-open-list-1>)

(127) is-open =
      (<s-file:is-ref>,
       <s-title:is-opt-expr>,
       <s-lsz:is-opt-expr>,
       <s-blz:is-opt>,
       <s-psz:is-opt-expr>,
       <s-open-attr:is-file-attr-set>,
       <s-env-attr:is-env-attr>,
       <s-volume:is-opt>)

(128) is-close-st =
      (<s-st:is-CLOSE>,
       <s-list:is-close-list-1>)

(129) is-close =
      (<s-file:is-ref>,
       <s-env-attr:is-env-attr>,
       <s-volume:is-opt>)

(130) is-get-st =
      is-file-get-st v is-string-get-st

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

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

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

```

30 June 1969

TRANSLATION OF PL/I INTO ABSTRACT SYNTAX

```

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

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

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

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

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

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

(140)  is-contr-item =
        (<s-contr-var:is-ref>,
         <s-spec-list:is-do-spec-list-1>,
         <s-do-list:is-item-list-1>)

(141)  is-read-st =
        is-into-read-st v is-set-read-st v is-ignore-read-st

(142)  is-into-read-st =
        (<s-st:is-READ>,
         <s-file:is-ref>,
         <s-into:is-ref>,
         <s-ident:is-opt-expr>,
         <s-idto:is-opt-ref>,
         <s-nolock:is-opt>,
         <s-event:is-opt-ref>)

```

TRANSLATION OF PL/I INTO ABSTRACT SYNTAX

30 June 1969

```

(143) is-set-read-st =
      (<s-st:is-READ>,
       <s-file:is-ref>,
       <s-ptr:is-ref>,
       <s-ident:is-opt-expr>,
       <s-idto:is-opt-ref>)

(144) is-ignore-read-st =
      (<s-st:is-READ>,
       <s-file:is-ref>,
       <s-ignore:is-opt-expr>,
       <s-event:is-opt-ref>)

(145) is-write-st =
      (<s-st:is-WRITE>,
       <s-file:is-ref>,
       <s-from:is-ref>,
       <s-ident:is-opt-expr>,
       <s-event:is-opt-ref>)

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

(147) is-locate-st =
      (<s-st:is-LOCATE>,
       <s-file:is-ref>,
       <s-id:is-id>,
       <s-ptr:is-opt-ref>,
       <s-ident:is-opt-expr>)

(148) is-delete-st =
      (<s-st:is-DELETE>,
       <s-file:is-ref>,
       <s-ident:is-opt-expr>,
       <s-event:is-opt-ref>)

(149) is-unlock-st =
      (<s-st:is-UNLOCK>,
       <s-file:is-ref>,
       <s-ident:is-expr>)

(150) is-display-st =
      (<s-st:is-DISPLAY>,
       <s-ident:is-expr>,
       <s-idto:is-opt-ref>,
       <s-event:is-opt-ref>)

```

30 June 1969

TRANSLATION OF PL/I INTO ABSTRACT SYNTAX

5.4 EXPRESSIONS

- (151) is-expr =
 is-infix-expr ∨ is-prefix-expr ∨ is-paren-expr ∨ is-ref ∨ is-const ∨ is-isub
- (152) is-infix-expr =
 (<s-opr:is-infix-opr>,
 <s-op-1:is-expr>,
 <s-op-2:is-expr>)
- (153) is-infix-opr =
 is-OR ∨ is-AND ∨ is-GT ∨ is-GE ∨ is-EQ ∨ is-LB ∨ is-LT ∨ is-NE ∨ is-CAT ∨
 is-ADD ∨ is-SUBTR ∨ is-MULT ∨ is-DIV ∨ is-EXP
- (154) is-prefix-expr =
 (<s-opr:is-prefix-opr>,
 <s-op:is-expr>)
- (155) is-prefix-opr =
 is-NOT ∨ is-PLUS ∨ is-MINUS
- (156) is-paren-expr =
 (<s-op:is-expr>)
- (157) is-ref =
 (<s-id-list:is-id-list-1>,
 <s-ptr:is-opt-ref>,
 <s-sl:is-subscr-expr-list>,
 <s-ap:is-expr-list-list>)
- (158) is-subscr-expr =
 is-expr ∨ is-*
- (159) is-const =
 is-arithm-const ∨ is-string-const
- (160) is-arithm-const =
 (<s-da:is-prop-arithm>,
 <s-v:is-num-val>)
- (161) is-string-const =
 (<s-da:is-fixed-string-edata>,
 <s-v:is-char-val-list ∨ is-bit-val-list>)

TRANSLATION OF PL/I INTO ABSTRACT SYNTAX

30 June 1969

(162) is-fixed-string-eda =
<s-base:is-CHAR v is-BIT>,
<s-length:is-intg-val>

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

(164) is-opt-expr =
is-expr v is-Ω

(165) is-opt-ref =
is-ref v is-Ω

(166) is-intg-val =
Note: cf. /5/

(167) is-num-val =
Note: cf. /5/

(168) is-char-val =
Note: cf. /5/

(169) is-bit-val =
Note: cf. /5/

30 June 1969

TRANSLATION OF PL/I INTO ABSTRACT SYNTAX

APPENDIX I: TRANSCRIPTION OF CONCRETE SYNTAX INTO ABSTRACT REPRESENTATION

The production rules of the concrete syntax as defined in chapter 3 of /3/ using an extended Backus notation form two categories of rules, the so-called higher and lower level production rules. The syntactical structuring of the rules of both levels itself satisfies a meta syntax also given in /3/ which is repeated in section 1.

The transcription of the rules in general depends on the syntax level in order to convey the information of the syntax level also to the transcribed rules, i.e., to the predicate definitions and to the classes of objects characterized by these predicate definitions. The transcription of the higher level production rules is defined in section 2.1; the transcription of the lower level production rules in section 2.2.

1. THE META SYNTAX

- (1) prod-rule ::= not-var :: definition
- (2) definition ::= sequence | sequence | definition
- (3) sequence ::= unit | unit sequence
- (4) unit ::= not-var | not-const | unit*** |
[definition] | [definition] |
[not-const * unit***] | [not-const * unit***]
- (5) not-var ::= sm-letter | sm-letter - not-var | sm-letter not-var
- (6) sm-letter ::= a | b | c | d | e | f | g | h | i | j | k | l | m |
n | o | p | q | r | s | t | u | v | w | x | y | z
- (7) not-const ::= PL/I-symb | PL/I-symb not-const
- (8) PL/I-symb ::= A | B | C | D | E | F | G | H | I | J | K | L | M |
N | O | P | Q | R | S | T | U | V | W | X | Y | Z |
\$ | @ | # | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
blank | - | = | + | - | * | / | (|) | , | . | ; |
: | & | l | ^ | > | < | ? | % | ^

Additional rules for the insertion of spaces:

Spaces (e.g. blanks, new lines, new pages) are optional immediately preceding or succeeding '::=' or ';' or '[' or ']' or '[' or ']' or '==' or '***' and between arbitrary adjacent units. A space is mandatory (to avoid ambiguities) between adjacent not-vars and between adjacent not-consts.

2. TRANSCRIPTION OF PRODUCTION RULES

The transcription of the production rules of the concrete syntax is performed by three functions ar, ar-1, and ar-2. The arguments of these functions are pieces of text out of concrete production rules (i.e. strings), and any such string takes a form which can be generated by the above meta syntax.

The transcription functions map such strings piece by piece into pieces of text of predicate definitions; this is done in a way that the mapping is governed by the meta syntactic category to which the argument belongs.

The transcription of a production rule is complete if the whole text of the production rule has been transcribed. In this case the resulting text is a complete predicate definition which is a member of the predicates defining the abstract representation of concrete programs.

Note: If for a metasyntactical argument no definition formula is found below, the definition is given by the definition for the immediately succeeding syntactical structure of this argument.

2.1 TRANSCRIPTION OF HIGHER LEVEL PRODUCTION RULES

Any production rule of the higher level concrete syntax is transcribed into a predicate definition by the function ar, which is recursively defined as follows:

(9) $\text{ar}(\text{not-var} ::= \text{definition}) =$
 $\text{is-c- not-var} = \text{ar}(\text{definition})$

(10) $\text{ar}(\text{sequence}_1 \mid \dots \mid \text{sequence}_n) =$
 $\text{ar}(\text{sequence}_1) \vee \dots \vee \text{ar}(\text{sequence}_n)$

Note: This rule is valid only for $n \geq 2$

(11) $\text{ar}(\text{unit}_1 \dots \text{unit}_n) =$
 $\langle s(1) : \text{ar}(\text{unit}_1),$
 \dots
 $\langle s(n) : \text{ar}(\text{unit}_n) \rangle$

Note: This rule is valid only for $n \geq 2$

(12) $\text{ar}(\text{not-var}) =$
 is-c- not-var

(13) $\text{ar}(\text{not-const}) =$
 $\text{ar-1}(\text{not-const})$

(14) $\text{ar}([\text{definition}]) =$
 $\text{ar}(\text{definition})$

(15) $\text{ar}([\text{definition}]) =$
 $\text{is-Q} \vee \text{ar}(\text{definition})$

(16) $\text{ar}(\text{unit}^{***}) =$
 $\langle s(1) : \text{ar}(\text{unit}) \rangle, \dots$

(17) $\text{ar}(\{\text{not-const} * \text{unit}^{***}\}) =$
 $\langle s-\text{del} : \text{ar}(\text{not-const}) \rangle,$
 $\langle s(1) : \text{ar}(\text{unit}) \rangle, \dots$

30 June 1969

TRANSLATION OF PL/I INTO ABSTRACT SYNTAX

(18) $\text{ar} ([\text{not-const} * \text{unit}^{**}]) =$
 $\text{is-Q} \vee \text{ar} (\{ \text{not-const} * \text{unit}^{**} \})$

2.2 TRANSCRIPTION OF LOWER LEVEL PRODUCTION RULES

Any production rule of the lower level concrete syntax is transcribed into a predicate definition by the function ar-1 , which is recursively defined as follows:

(19) $\text{ar-1} (\text{not-var} ::= \text{definition}) =$
 $\text{is-c- not-var} = \text{ar} (\text{definition})$

(20) $\text{ar-1} (\text{sequence}_1 | \dots | \text{sequence}_n) =$
 $\text{ar-1} (\text{sequence}_1) \vee \dots \vee \text{ar-1} (\text{sequence}_n)$

Note: This rule is valid only for $n \geq 2$

(21) $\text{ar-1} (\text{unit}_1 \dots \text{unit}_n) =$
 $\langle \text{elem}(1) : \text{ar-1} (\text{unit}_1),$
 \dots
 $\langle \text{elem}(n) : \text{ar-1} (\text{unit}_n) \rangle \rangle$

Note: This rule is valid only for $n \geq 2$

(22) $\text{ar-1} (\text{not-var}) =$
 is-c- not-var

(23) $\text{ar-1} (\text{PL/I-symb}_1, \dots, \text{PL/I-symb}_n) =$
 $\langle \text{elem}(1) : \text{ar-1} (\text{PL/I-symb}_1),$
 \dots
 $\langle \text{elem}(n) : \text{ar-1} (\text{PL/I-symb}_n) \rangle \rangle$

Note: This rule is valid only for $n \geq 2$. It applies e.g. for keywords and composite operators. In order to keep the transcribed predicates short, this transcription rule was not applied in chapter 3 in all instances where the PL/I-symbols were letters.

(24) $\text{ar-1} (\text{PL/I-symb}) =$
 $\text{ar-2} (\text{PL/I-symb})$

(25) $\text{ar-1} (\{ \text{definition} \}) =$
 $\text{ar-1} (\text{definition})$

(26) $\text{ar-1} ([\text{definition}]) =$
 $\text{is-Q} \vee \text{ar-1} (\text{definition})$

(27) ar-1 (unit***) =
 (< elem(1) : ar-1 (unit)>,...)

The function ar-2 is defined as follows:

(28)	PL/I-symb		ar-2 (PL/I-symb)
	A		is-A-CHAR
	•••		•••
	Z		is-Z-CHAR
	\$		is-DOLLAR
	@		is-COMM-AT
	#		is-NUMBER-SIGN
	0		is-0-CHAR
	•••		•••
	9		is-9-CHAR
	-		is-BREAK
	BLANK		is-BLANK
	*		is-APOSTR
	=		is-EQ
	+		is-PLUS
	-		is-MINUS
	*		is-ASTER
	/		is-SLASH
	(is-LEFT-PAR
)		is-RIGHT-PAR
	,		is-COMMA
	.		is-POINT
	:		is-SEMIC
	:		is-COLON
	&		is-AND
			is-OR
	~		is-NOT
	>		is-GT
	<		is-LT
	?		is-QUEST
	%		is-PERC

30 June 1969

TRANSLATION OF PL/I INTO ABSTRACT SYNTAX

APPENDIX II: CROSS-REFERENCE INDEX

This index lists all names used in the document, with the exception of names of selectors or formal parameters, metavariables, abbreviations and those function names, which are defined in chapter 1 of /5/.

Formulas are referenced to by the form XX-YYY(ZZZ), where XX is the number of the main chapter, YYY is the page number within the main chapter and ZZZ is the number of the formula within the main chapter.

The following conventions hold:

- (1) For names defined in chapter 1 only the defining formula is given.
- (2) For all other names all instances of use in a formula are given. The defining formula is indicated by an underlined reference.

APPENDIX II: CROSS-REFERENCE INDEX

A-CHAR 2-8(26), 4-22(80), 4-38(151), 4-49(196), 5-8(74)
ACC 4-74(296), 4-74(297), 5-13(124)
ACCESS 4-74(293), 5-13(122)
ADD 4-55(218), 4-82(321), 5-17(153)
AL 4-27(96), 4-27(97), 4-46(184), 5-4(31), 5-5(47), 5-6(55)
al-sucess-list(p) 4-59(238), 4-59(237)
ALL-DATA 4-79(311), 5-15(137)
ALLOCATE 4-69(274), 5-12(108)
AND 2-4(9), 2-8(24), 2-8(26), 3-20(134), 3-24(162), 5-17(153)
APOSTR 3-23(159), 3-23(161), 3-24(162), 3-24(164), 4-89(347)
app-da(z,bs) 4-88(343), 4-87(342), 4-88(343)
AREA 4-10(33), 4-18(67), 4-25(90), 4-31(116), 4-32(119), 4-32(119), 4-44(179), 4-51(204),
 4-53(214), 4-70(279), 4-73(290), 5-6(56), 5-12(115)
area-da(pd) 4-40(161), 4-31(116), 4-45(181), 4-70(279)
arg-p-list(b,p) 4-86(339), 4-86(338), 4-86(339)
ARITHM 4-31(116), 4-32(117), 4-32(118), 4-32(119), 4-51(204)
arithm-base(pd) 4-33(125), 4-33(123), 4-34(127), 4-34(132)
arithm-da(pd) 4-33(123), 4-31(116)
arithm-mode(pd) 4-33(124), 4-33(123), 4-34(131)
arithm-prec(pd) 4-34(127), 4-33(123)
arithm-scale(pd) 4-33(126), 4-33(123), 4-34(127), 4-34(133), 4-35(134)
arithm-scale-f(pd) 4-35(134), 4-33(123)
ASSIGN 4-69(272), 5-12(107)
ASTER . . 2-4(9), 2-4(11), 2-4(12), 3-3(19), 3-4(28), 3-5(34), 3-5(36), 3-6(45), 3-7(52), 3-21(139),
 3-21(140), 3-22(144), 3-24(162), 3-24(166), 4-21(78), 4-28(104), 4-28(105), 4-54(217),
 4-81(319), 4-82(320), 4-82(321), 4-82(321), 4-85(333), 4-86(340), 5-7(68), 5-8(74)
ASYN 4-74(297), 5-13(124)
ATTN 4-23(81), 4-25(89), 4-26(92), 4-73(292), 5-13(121)
AUTO 4-27(99), 4-28(101), 5-2(11)
B-CHAR 4-22(80), 4-22(80), 4-49(196), 4-87(342), 4-89(347), 5-7(68)
BAC 4-47(188), 5-3(17)
BASED 4-19(72), 4-23(81), 4-24(88), 4-26(92), 4-29(106)
based-ptr(pd) 4-48(192), 4-48(191)

30 June 1969

TRANSLATION OF PL/I INTO ABSTRACT SYNTAX

BBIT	4-49(196), 5-9(81)
BCOL	4-49(196), 5-9(84)
bdp-list(pd)	4-28(102), 4-23(84), 4-30(111), 4-43(172), 4-44(177), 4-48(190), 4-48(191), 4-50(201), 4-51(203), 4-70(276)
BIN	4-33(125), 4-52(209), 4-87(342), 4-88(343), 5-4(33), 5-6(57)
BIT	4-35(140), 4-36(144), 4-49(196), 4-52(209), 4-78(310), 4-87(342), 5-4(34), 5-6(49), 5-6(58), 5-7(64), 5-9(81), 5-14(132), 5-15(135), 5-18(162)
BLANK	2-4(10), 3-24(162), 5-7(68), 5-8(74)
block-defaults(ad-set,b,id)	4-19(71), 4-19(69), 4-19(71)
block-p(p)	4-4(8), 4-7(24), 4-10(31), 4-14(50), 4-14(51), 4-15(56), 4-16(57), 4-19(69), 4-23(81), 4-29(107), 4-54(215), 4-55(219), 4-57(226), 4-57(227), 4-61(241), 4-68(268), 4-68(269), 4-70(276), 4-71(284), 4-73(290), 4-74(298), 4-79(311), 4-81(319), 4-83(323), 4-84(327), 4-86(335), 4-90(351)
BOV	4-73(291), 5-13(119)
BPIC	4-49(196), 5-9(82)
BREAK	2-8(24), 3-22(150)
BSPACE	4-49(196), 5-9(84)
BST	4-47(188), 5-3(17)
BUF	4-47(188), 5-3(17)
BUILTIN	4-10(33), 4-18(67), 4-26(92), 4-26(92), 4-27(94), 5-2(10)
c-base(pd)	4-34(132), 4-34(130)
C-CHAR	2-8(26), 4-22(80), 4-22(80), 4-49(196), 5-7(68)
c-mode(pd)	4-34(131), 4-34(130)
c-scale(pd)	4-34(133), 4-34(130)
CALL	4-66(259), 5-11(96)
CAT	4-82(321), 5-17(153)
CHAR	4-35(140), 4-36(144), 4-49(196), 4-52(209), 4-78(310), 4-87(342), 5-4(34), 5-6(49), 5-6(58), 5-7(64), 5-9(81), 5-14(132), 5-15(135), 5-18(162)
CHECK	4-63(247), 5-13(117)
CLOSE	4-77(305), 5-14(128)
COL	4-49(196), 5-9(84)
collect(set)	1-2(5)
COLON	2-4(9), 2-8(24), 2-8(26), 3-3(19), 3-4(28), 3-10(73), 3-10(77), 3-24(162), 4-21(78)
COMM-AT	2-8(24), 3-22(149), 4-21(79)
COMMA	2-4(9), 2-8(26), 2-9(29), 3-1(3), 3-2(11), 3-2(15), 3-3(19), 3-3(21), 3-3(22), 3-3(28), 3-4(25), 3-4(27), 3-5(32), 3-5(37), 3-6(44), 3-7(51), 3-7(55), 3-8(61), 3-9(66), 3-9(67), 3-10(73), 3-11(84), 3-13(94), 3-13(95), 3-13(96), 3-14(100), 3-14(101), 3-15(104), 3-15(109), 3-15(110), 3-16(114), 3-16(115), 3-16(116), 3-17(118), 3-17(120), 3-18(128), 3-22(144), 3-24(162), 3-24(166), 5-7(68), 5-8(74)

TRANSLATION OF PL/I INTO ABSTRACT SYNTAX

30 June 1969

30 June 1969

TRANSLATION OF PL/I INTO ABSTRACT SYNTAX

descr-p-list(p)	4-56 (224), 4-56 (222), 4-56 (223), 4-58 (232)
descr-p-list-1(p)	4-56 (223), 4-56 (222)
descr-string-da(pd)	4-45 (180), 4-44 (179)
descr-succ-p-list(p)	4-31 (115), 4-30 (112)
descriptor-defaults(b)	4-57 (228), 4-57 (227)
dim-p(pd)	4-28 (103), 4-28 (102)
dimensionality(ad-set)	4-86 (336), 4-86 (335)
DIR	4-47 (188), 5-3 (17)
DISABLE	4-74 (299), 5-13 (125)
DISPLAY	4-80 (316), 5-16 (150)
DIV	4-82 (321), 5-17 (153)
DOLLAR	3-22 (149), 3-24 (166), 4-21 (79), 5-7 (68)
E-CHAR	2-8 (26), 4-22 (80), 4-22 (80), 4-39 (158), 4-39 (158), 4-49 (196), 4-50 (197), 5-7 (67)
elem-set(list)	4-38 (152), 4-38 (151), 4-38 (153), 4-38 (154), 4-39 (157), 4-56 (223)
empty-cond-pt	4-60 (240), 4-60 (239), 4-62 (242), 4-66 (257)
ENABLE	4-74 (294), 5-13 (123)
end-cl-p(x)	2-7 (19), 2-6 (16)
ENDF	4-73 (291), 5-13 (119)
ENDP	4-73 (291), 5-13 (119)
ENTRY	4-10 (33), 4-18 (67), 4-25 (91), 4-27 (94), 4-31 (116), 4-32 (119), 4-46 (184), 4-51 (204), 5-6 (56)
entry-attr-p(pd)	4-42 (169), 4-42 (168), 4-42 (171)
ENTRY-CONST	4-3 (4), 4-23 (81), 4-25 (89), 4-25 (91)
entry-da(pd)	4-32 (120), 4-31 (116)
entry-descr-p-list(p-list,p)	4-56 (225), 4-56 (224), 4-56 (225)
entry-p(pd)	4-43 (173), 4-42 (171), 4-45 (182)
env-attr-p(pd)	4-47 (189), 4-47 (187)
env-trans-arg-list(b,p)	4-86 (338), 4-83 (324)
env-trans-bdp(pd,p)	4-28 (104), 4-28 (102)
env-trans-call-st(b,p)	4-66 (259), 4-54 (216), 4-61 (241)
env-trans-expr(b,p)	4-82 (320), 4-29 (106), 4-54 (217), 4-81 (319), 4-82 (320), 4-85 (332), 4-86 (340)
env-trans-id(b,p)	4-90 (352), 4-90 (351)
env-trans-init(b,p)	4-54 (217), 4-54 (216), 4-54 (217)
env-trans-init-const(b,p)	4-55 (218), 4-54 (217)

TRANSLATION OF PL/I INTO ABSTRACT SYNTAX

30 June 1969

env-trans-init-spec(b,p)	4-54 (216), 4-54 (215)
env-trans-lbd(pd,p)	4-28 (105), 4-28 (104)
env-trans-ref(b,p)	4-83 (324), 4-33 (121), 4-66 (260), 4-67 (261), 4-68 (268), 4-68 (269), 4-72 (285), 4-73 (290), 4-74 (298), 4-82 (320), 4-83 (323), 4-83 (324)
env-trans-subscr-list(b,p)	4-85 (332), 4-83 (324)
EOV	4-73 (291), 5-13 (119)
EQ	2-4 (9), 2-8 (26), 3-11 (84), 3-14 (100), 3-21 (136), 3-24 (162), 4-82 (321), 5-17 (153)
ERROR	4-73 (290), 5-12 (115)
EVENT	4-10 (33), 4-18 (67), 4-25 (90), 4-32 (119), 4-51 (204), 5-4 (32), 5-5 (48), 5-6 (56)
EXC	4-47 (188), 5-3 (17)
EXIT	4-62 (241), 5-11 (105)
EXP	4-82 (321), 5-17 (153)
expand-pic-spec(pic)	4-38 (149), 4-37 (148), 4-38 (149)
expl-decl-ref(b,id1)	4-84 (327), 4-14 (51), 4-16 (57), 4-84 (326), 4-84 (327)
expl-defaults(b,id)	4-20 (75), 4-19 (71)
EXT	4-3 (3), 4-3 (4), 4-26 (93), 4-27 (94), 4-27 (99), 4-42 (167), 4-46 (185), 5-2 (11), 5-2 (14), 5-2 (16)
ext-entry-set(t)	4-3 (4), 4-2 (1), 4-3 (6)
ext-proc-p(p)	4-4 (13), 4-10 (31)
EXTERNAL	4-45 (182)
F-CHAR	4-22 (80), 4-22 (80), 4-49 (196)
FETCH	4-68 (268), 5-11 (101)
FILE	4-10 (33), 4-18 (67), 4-25 (91), 4-26 (92), 4-32 (119), 4-51 (204), 5-4 (32), 5-5 (48), 5-6 (56)
FILE-CONST	4-23 (81), 4-25 (89), 4-25 (91), 4-26 (92), 4-26 (93)
FINISH	4-73 (290), 5-12 (115)
first-letter(id)	4-35 (137), 4-35 (136)
FIX	4-30 (109), 4-33 (126), 4-49 (196), 4-52 (209), 4-88 (343), 5-4 (33), 5-6 (57), 5-9 (79)
FLT	4-33 (126), 4-34 (133), 4-35 (134), 4-46 (184), 4-49 (196), 4-52 (209), 4-88 (343), 5-4 (33), 5-6 (57), 5-9 (79)
FOFL	4-63 (246), 5-13 (116)
FORMAT	4-10 (33), 4-18 (67), 4-23 (81), 4-25 (89), 4-26 (92), 4-26 (92), 4-27 (94)
format-attr-p(pd)	4-48 (194), 4-48 (193)
FREE	4-71 (283), 5-12 (110)
G-CHAR	2-8 (26), 4-22 (80), 4-22 (80), 4-38 (154)
GE	4-82 (321), 5-17 (153)

30 June 1969

TRANSLATION OF PL/I INTO ABSTRACT SYNTAX

gen-base(ad-set)	4-52 (207), 4-51 (205)
gen-base-1(pd)	4-53 (213), 4-52 (212)
gen-mode(ad-set)	4-52 (206), 4-51 (205)
gen-prec(prec)	4-52 (210), 4-51 (205)
gen-scale(ad-set)	4-52 (208), 4-51 (205)
gen-scale-f(prec)	4-52 (211), 4-51 (205)
generate(t)	2-3121, 2-2 (1), 2-6 (14)
generate-equiv(t)	2-6 (14)
generate-1(x)	2-3 (3), 2-3 (2), 2-3 (3), 2-8 (24), 4-21 (79), 4-90 (353)
generate-2(x)	2-3 (4), 2-3 (2), 2-3 (4), 2-8 (24), 2-9 (28), 4-21 (79), 4-90 (353)
generate-48(t)	2-8 (24)
GENERIC	4-23 (81), 4-25 (89), 4-26 (92)
generic-attr-p(pd)	4-50 (199), 4-50 (198)
GET	4-77 (305), 5-14 (131), 5-14 (132)
GOTO	4-66 (258), 5-10 (95)
GT	2-4 (9), 2-8 (24), 2-8 (26), 2-8 (26), 3-21 (136), 3-21 (142), 3-24 (162), 4-82 (321), 5-17 (153)	
H-CHAR	4-22 (80), 4-22 (80), 4-40 (160)
I-CHAR	4-22 (80), 4-22 (80), 4-35 (138), 5-7 (68)
id-p-list(p)	4-85 (331), 4-85 (330)
IF	4-66 (256), 5-10 (94)
incorporate(program,id-set,library)	4-3 (5), 4-2 (1), 4-3 (6)
INCORPORATE	4-67 (266), 5-11 (99)
inf-operator(x)	4-82 (321), 4-82 (320)
INP	4-47 (188), 5-3 (17)
insert-aster(pa-opt)	4-67 (263), 4-67 (262)
insert-aster-1(x)	4-67 (264), 4-67 (263)
insert-space(x)	2-3 (6), 2-3 (2)
insert-space-1(x)	2-3 (7), 2-3 (6), 2-3 (7)
insert-space-48(x)	2-9 (27), 2-8 (24)
insert-space-48-1(x)	2-9 (28), 2-9 (27), 2-9 (28)
INT	4-26 (93), 4-27 (94), 4-27 (99), 5-2 (11), 5-2 (14), 5-2 (16)	
intg-val(y,bs)	4-89 (348), 4-89 (347), 4-89 (348)
is-access-st	5-13 (122), 5-9 (87)
is-al	5-12 (109), 5-12 (108)

TRANSLATION OF PL/I INTO ABSTRACT SYNTAX

30 June 1969

is-al-aggr	5-7(59), 5-7(60), 5-7(62), 5-12(109)
is-al-area	5-7(65), 5-7(63)
is-al-array	5-7(60), 5-7(59)
is-al-attr-of(p,ad)	4-59(236), 4-57(226)
is-al-scalar	5-7(63), 5-7(59)
is-al-string	5-7(64), 5-7(63)
is-al-struct	5-7(61), 5-7(59)
is-al-succ	5-7(62), 5-7(61)
is-allocate-st	5-12(108), 5-9(87)
is-area-cont(p)	4-17(64), 4-18(67)
is-arithm-const	5-17(160), 5-17(159)
is-assign-st	5-12(107), 5-9(87)
is-attn	5-3(25), 5-2(10)
is-attn-cond	5-13(121), 5-12(115), 5-13(122), 5-13(124), 5-13(125)
is-attr-descr(ad)	4-10(33), 4-10(32)
is-attr-of(p,ad)	4-18(67), 4-10(31), 4-14(49), 4-14(50), 4-18(68)
is-based	5-2(13), 5-2(10)
is-bin-pic	5-8(71), 5-7(66)
is-bin-spec	5-8(72), 5-8(71)
is-bit-val	5-18(169), 5-17(161)
is-block	5-10(89), 5-9(87)
is-block-p(b)	4-3(7), 4-10(32)
is-body	5-1(3), 5-1(2)
is-body-part	5-1(2), 5-1(1), 5-1(3), 5-10(89)
is-builtin-cont(p)	4-16(58), 4-18(67)
is-c-[abbr-name]	2-4(13)
is-c-[name]	3-25(172), 2-4(13)
is-c-A	3-9(68), 3-22(149), 3-24(166)
is-c-abbr-[name]	2-4(13)
is-c-ACCESS	3-16(115), 3-16(116), 4-74(296), 4-74(297)
is-c-access-statement	3-16(115), 3-10(78), 4-6(20), 4-62(241), 4-74(293), 4-90(355)
is-c-ALIGNED	3-4(31), 4-18(68), 4-27(97), 4-51(204), 4-58(231)
is-c-ALL	3-2(14)
is-c-ALLOCATE	3-14(101)

30 June 1969

TRANSLATION OF PL/I INTO ABSTRACT SYNTAX

TRANSLATION OF PL/I INTO ABSTRACT SYNTAX

30 June 1969

is-c-BIT	3-5 (34), 4-20 (74), 4-20 (77), 4-35 (140), 4-36 (144), 4-52 (209)
is-c-bit-string	<u>3-23 (159)</u> , 3-23 (158), 4-87 (342), 4-89 (347)
is-c-BITSTREAM	3-7 (54), 4-47 (188)
is-c-BITSTRING	3-18 (123), 4-76 (302), 4-78 (310)
is-BLANK	3-24 (162)
is-c-BLINESIZE	3-17 (119), 4-76 (302), 4-78 (307)
is-c-block	<u>4-4 (9)</u> , 4-3 (7), 4-4 (8), 4-4 (11), 4-4 (12), 4-7 (21), 4-90 (352)
is-c-bound-pair	<u>3-4 (28)</u> , 3-4 (27), 4-28 (104), 4-28 (105)
is-c-BP	3-9 (69), 4-49 (196)
is-c-BUFFERED	3-7 (54), 4-20 (74), 4-47 (188)
is-c-BUILTIN	3-7 (49), 4-26 (92), 4-27 (94)
is-c-BX	3-9 (70), 4-49 (196)
is-c-BY	3-11 (85), 3-14 (100), 4-65 (254)
is-c-C	3-9 (67), 3-22 (149), 3-24 (166)
is-c-CALL	3-6 (43), 3-12 (90), 4-20 (77)
is-c-call-optionslist	<u>3-12 (91)</u> , 3-12 (90), 4-67 (262), 4-75 (300)
is-c-call-statement	<u>3-12 (90)</u> , 3-10 (78), 4-16 (58), 4-61 (241), 4-66 (259)
is-c-CAT	2-8 (24)
is-c-CHARACTER	3-5 (34), 4-20 (74), 4-20 (77), 4-35 (140), 4-36 (144), 4-52 (209)
is-c-character-string	<u>3-23 (161)</u> , 3-23 (158), 4-89 (347)
is-c-CHECK	3-15 (109)
is-c-check-condition	<u>3-15 (109)</u> , 3-10 (74), 3-15 (108), 4-62 (242), 4-64 (249), 4-73 (290)
is-c-CLOSE	3-17 (120), 4-77 (305)
is-c-close-optionslist	<u>3-17 (121)</u> , 3-17 (120), 4-75 (300)
is-c-close-statement	<u>3-17 (120)</u> , 3-10 (78), 4-62 (241), 4-78 (308)
is-c-COLUMN	3-9 (70), 4-49 (196)
is-c-comment	<u>2-4 (11)</u> , 2-4 (10)
is-c-comment-symbol(ch)	<u>2-4 (12)</u> , 2-4 (11)
is-c-comparison-operator	<u>3-21 (136)</u> , 3-20 (135)
is-c-COMPLEX	3-5 (32), 4-20 (74), 4-20 (77), 4-32 (117), 4-32 (119), 4-33 (124), 4-34 (130), 4-34 (131), 4-37 (147), 4-52 (206), 4-52 (209)
is-c-complex-format	<u>3-9 (67)</u> , 3-9 (65), 4-50 (197)
is-c-compound(x)	<u>2-6 (17)</u> , 2-6 (16), 2-7 (18), 2-7 (19), 2-7 (20), 2-7 (22)
is-c-condition	<u>3-15 (108)</u> , 3-15 (105), 3-15 (106), 3-15 (107), 4-73 (290)

30 June 1969

TRANSLATION OF PL/I INTO ABSTRACT SYNTAX

is-c-CONDITION	3-16 (113), 4-90 (355)
is-c-CONNECTED	3-4 (31), 4-26 (92), 4-27 (95)
is-c-constant	<u>3-22 (146)</u> , 3-21 (141), 4-82 (320), 4-87 (342), 4-89 (347)
is-c-control-format	<u>3-9 (70)</u> , 3-8 (64), 4-50 (197)
is-c-CONTROLLED	3-5 (39), 4-20 (74), 4-28 (101)
is-c-controlled-allocate-item	<u>3-14 (103)</u> , 3-14 (101), 4-57 (226), 4-58 (232), 4-58 (233), 4-58 (234), 4-59 (236), 4-70 (276)
is-c-CONVERSION	3-10 (75), 4-63 (246)
is-c-COPY	3-18 (123), 4-76 (302), 4-77 (303)
is-c-D	3-22 (149), 3-24 (166)
is-c-DATA	3-18 (125), 4-79 (311)
is-c-data-attribute	<u>3-4 (31)</u> , 3-4 (26), 3-4 (30), 4-25 (90)
is-c-data-directed	<u>3-18 (125)</u> , 3-18 (124)
is-c-data-format	<u>3-9 (65)</u> , 3-8 (64)
is-c-data-specification	<u>3-18 (124)</u> , 3-18 (123), 4-76 (302), 4-76 (303), 4-79 (311)
is-c-datalist	<u>3-18 (128)</u> , 3-18 (125), 3-18 (126), 3-18 (127), 3-19 (129), 4-79 (312), 4-80 (314)
is-c-datalist-element	<u>3-19 (129)</u> , 3-18 (128), 4-80 (314)
is-c-DECIMAL	3-5 (32), 4-20 (74), 4-20 (77), 4-33 (125), 4-34 (130), 4-34 (132), 4-52 (207), 4-52 (209)
is-c-declaration	<u>3-2 (12)</u> , 3-2 (11), 4-13 (46), 4-18 (67)
is-c-declaration-sentence	<u>3-2 (91)</u> , 3-1 (7), 4-6 (20), 4-11 (38), 4-12 (44), 4-60 (239)
is-c-declarationlist	<u>3-2 (11)</u> , 3-2 (10), 3-2 (12)
is-c-DECLARE	3-2 (10)
is-c-declare-sentence	<u>3-2 (10)</u> , 3-2 (9), 4-11 (36)
is-c-DEFAULT	3-2 (14), 3-2 (15)
is-c-default-option-1	<u>3-2 (14)</u> , 3-2 (13), 4-20 (75), 4-57 (228)
is-c-default-option-2	<u>3-2 (15)</u> , 3-2 (13)
is-c-default-sentence	<u>3-2 (13)</u> , 3-2 (9)
is-c-default-spec	<u>3-3 (16)</u> , 3-2 (15), 3-3 (22), 4-20 (75), 4-57 (228)
is-c-DEFINED	3-6 (40), 4-20 (74), 4-48 (190)
is-c-defined-attribute	<u>3-6 (40)</u> , 3-4 (31), 4-26 (92), 4-27 (94)
is-c-DELAY	3-13 (97)
is-c-delay-statement	<u>3-13 (97)</u> , 3-10 (78), 4-62 (241), 4-68 (271)

TRANSLATION OF PL/I INTO ABSTRACT SYNTAX

30 June 1969

is-c-DELETE 3-19 (130), 4-77 (305)
is-c-delimiter(x) <u>2-4 (9)</u> , 2-3 (7), 2-9 (29)
is-c-delimiter-48(x) <u>2-9 (29)</u> , 2-9 (28)
is-c-descriptor	<u>3-7 (52)</u> , 3-7 (51), 4-57 (226), 4-57 (229), 4-58 (230), 4-58 (231), 4-58 (232), 4-58 (233), 4-58 (234)
is-c-descriptorlist <u>3-7 (51)</u> , 3-7 (50), 3-7 (51), 3-8 (56), 4-56 (225)
is-c-DESCRIPTORS 3-3 (18), 4-57 (228)
is-c-digit <u>3-22 (151)</u> , 3-22 (150), 3-23 (153), 4-89 (348)
is-c-dimension-attribute	<u>3-4 (27)</u> , 3-2 (12), 3-3 (20), 3-7 (52), 3-14 (103), 4-10 (33), 4-18 (67), 4-20 (73), 4-20 (74), 4-20 (76), 4-20 (77), 4-25 (90), 4-28 (103), 4-51 (204), 4-58 (230), 4-59 (236), 4-86 (336)
is-c-DIRECT 3-7 (54), 4-20 (74), 4-47 (188)
is-c-DISABLE 3-16 (117)
is-c-disable-statement <u>3-16 (117)</u> , 3-10 (78), 4-62 (241), 4-74 (299)
is-c-DISPLAY 3-20 (132)
is-c-display-statement <u>3-20 (132)</u> , 3-10 (78), 4-62 (241), 4-80 (316)
is-c-DO 3-11 (82), 3-11 (83), 3-19 (129)
is-c-do-specification <u>3-11 (84)</u> , 3-11 (83), 3-19 (129)
is-c-E 3-9 (66), 3-22 (149), 3-23 (156), 3-24 (166)
is-c-EDIT 3-18 (126), 4-79 (311)
is-c-edit-directed <u>3-18 (126)</u> , 3-18 (124)
is-c-ELSE 3-12 (86), 3-12 (88), 3-16 (115)
is-c-ENABLE 3-16 (116)
is-c-enable-statement <u>3-16 (116)</u> , 3-10 (78), 4-62 (241), 4-74 (294)
is-c-END 2-7 (23), 3-1 (6), 4-62 (241)
is-c-end-clause <u>3-1 (6)</u> , 3-1 (5), 4-7 (21), 4-11 (38), 4-60 (239)
is-c-ENDFILE 3-16 (112), 4-73 (291)
is-c-ENDPAGE 3-16 (112), 4-73 (291)
is-c-ENDVOLUME 3-16 (112), 4-73 (291)
is-c-entry	<u>3-2 (8)</u> , 3-1 (7), 4-4 (11), 4-5 (17), 4-6 (18), 4-7 (23), 4-11 (37), 4-11 (40), 4-43 (174)
is-c-ENTRY 3-2 (8), 3-7 (50), 4-42 (168), 4-42 (169), 4-42 (171), 4-56 (224)
is-c-entry-name-attribute	<u>3-7 (50)</u> , 3-4 (26), 3-4 (30), 4-20 (74), 4-25 (91), 4-32 (119), 4-56 (222), 4-56 (223), 4-57 (226)
is-c-env-option 3-7 (53), 3-8 (59), 3-17 (119), 3-17 (121)

30 June 1969

TRANSLATION OF PL/I INTO ABSTRACT SYNTAX

TRANSLATION OF PL/I INTO ABSTRACT SYNTAX

30 June 1969

is-c-format-item	3-8(64), 3-8(62), 3-8(63), 4-48(195), 4-49(196), 4-50(197)
is-c-format-iteration	3-8(63), 3-8(62), 4-49(196)
is-c-format-sentence	3-8(60), 3-1(7), 4-7(21), 4-11(39), 4-48(194)
is-c-formatlist	3-8(61), 3-8(60), 3-8(63), 3-18(126), 4-48(195)
is-c-FREE	3-15(104)
is-c-free-statement	3-15(104), 3-10(78), 4-62(241), 4-71(283)
is-c-FROM	3-19(131), 4-76(302)
is-c-G	3-22(149), 3-24(166)
is-c-GE	2-8(24)
is-c-GENERIC	3-7(55)
is-c-generic-attribute	3-7(55), 3-7(49), 4-26(92), 4-27(94), 4-50(199), 4-57(226)
is-c-generic-element	3-8(56), 3-7(55), 4-56(222), 4-56(223)
is-c-GET	3-17(122), 4-16(60), 4-77(305)
is-c-GO	3-12(89)
is-c-GOTO	3-12(89)
is-c-goto-statement	3-12(89), 3-10(78), 4-61(241), 4-66(258)
is-c-group	3-11(81), 2-6(17), 3-10(78), 4-7(21), 4-61(241), 4-65(252), 4-72(287)
is-c-GT	2-8(24)
is-c-H	3-22(149), 3-24(166)
is-c-I	3-22(149), 3-23(157), 3-24(166)
is-c-identifier	3-22(148), 2-7(18), 2-8(24), 3-1(3), 3-2(12), 3-3(19), 3-5(37), 3-14(102), 3-14(103), 3-16(113), 3-16(114), 3-16(115), 3-19(130), 3-22(143), 3-22(145), 4-11(36), 4-11(37), 4-21(78), 4-43(172), 4-76(303), 4-85(330), 4-85(331), 4-90(351), 4-90(352), 4-90(353), 4-90(355)
is-c-identifier-range-spec	3-3(19), 3-3(18)
is-c-IF	3-12(87)
is-c-if-clause	3-12(87), 3-12(86), 3-12(88), 4-6(20), 4-61(241)
is-c-if-statement	3-12(86), 2-7(21), 3-10(72), 4-66(256)
is-c-IGNORE	3-19(131), 4-76(302)
is-c-imaginary-constant	3-23(157), 3-7(48), 3-22(146), 4-87(342), 4-89(347)
is-c-IN	3-14(102), 3-15(104), 4-17(64), 4-71(282)
is-c-INCORPORATE	3-13(93)
is-c-incorporate-specification	3-13(93)
is-c-incorporate-statement	3-13(93), 3-10(78), 4-61(241), 4-67(266)
is-c-INITIAL	3-6(42), 4-20(77)

30 June 1969

TRANSLATION OF PL/I INTO ABSTRACT SYNTAX

is-c-initial-attribute	3-6(42), 3-4(31), 3-14(103), 4-54(215)
is-c-initial-call	3-6(43), 3-6(42), 4-16(58), 4-54(216), 4-66(259)
is-c-initial-constant	3-6(47), 3-6(45), 3-6(46), 4-54(217)
is-c-initial-item	3-6(45), 3-6(44)
is-c-initial-itemlist	3-6(44), 3-6(42), 3-6(46), 4-54(216), 4-54(217)
is-c-initial-iteration	3-6(46), 3-6(45), 4-54(217)
is-c-INPUT	3-7(54), 4-20(74), 4-47(188)
is-c-integer . . .	3-23(153), 3-2(12), 3-5(32), 3-5(33), 3-7(52), 3-8(63), 3-14(103), 3-22(147), 3-23(152), 3-23(155), 3-23(156), 3-24(163), 3-24(164), 3-24(165), 4-13(46), 4-49(196), 4-87(342), 4-89(347), 4-89(348)
is-c-INTERNAL	3-8(57), 4-20(74), 4-27(94)
is-c-INTO	3-19(131), 4-76(302)
is-c-io-condition	3-16(112), 3-16(111)
is-c-IRREDUCIBLE	3-7(50), 4-46(186)
is-c-isub	3-23(152), 3-21(141), 4-82(320)
is-c-iterated-group	3-11(83), 2-7(22), 3-11(81), 4-5(17)
is-c-J	3-22(149)
is-c-K	3-22(149), 3-24(166)
is-c-KEY	3-16(112), 3-19(131), 4-73(291), 4-76(302)
is-c-KEYED	3-7(54), 4-47(188)
is-c-KEYFROM	3-19(131), 4-76(302)
is-c-KEYTO	3-19(131), 4-76(302)
is-c-L	3-22(149), 3-24(163)
is-c-LABEL	3-5(37), 4-20(77)
is-c-label-attribute	3-5(37), 3-4(31), 4-32(119), 4-41(165)
is-c-labellist	3-10(77), 3-1(2), 3-1(6), 3-2(8), 3-2(9), 3-8(60), 3-10(72), 3-12(88)
is-c-LE	2-8(24)
is-c-letter	3-22(149), 3-3(19), 3-22(148), 3-22(150)
is-c-LIKE	3-8(58)
is-c-like-attribute	3-8(58), 3-4(30), 4-14(49), 4-14(50), 4-20(77), 4-56(222)
is-c-LINE	3-9(70), 3-18(123), 4-49(196), 4-76(302)
is-c-LINESIZE	3-17(119), 4-76(302)
is-c-LIST	3-18(127), 4-79(311)
is-c-list-directed	3-18(127), 3-18(124)
is-c-LOCATE	3-19(130), 4-77(305), 4-80(315), 4-90(355)

TRANSLATION OF PL/I INTO ABSTRACT SYNTAX

30 June 1969

is-c-LT	2-8 (24)
is-c-M	3-22 (149), 3-24 (166)
is-c-N	3-22 (149)
is-c-NAME	3-14 (100), 3-16 (112), 4-73 (291)
is-c-named-io-condition	<u>3-16 (111)</u> , 3-15 (108), 4-16 (59), 4-73 (290)
is-c-NE	2-8 (24)
is-c-NG	2-8 (24)
is-c-NL	2-8 (24)
is-c-no-check-condition	<u>3-15 (110)</u> , 3-10 (74), 4-62 (242), 4-63 (245)
is-c-no-prefix	<u>3-10 (76)</u> , 3-10 (74), 4-63 (245), 4-63 (246), 4-64 (249)
is-c-NOCHECK	3-15 (110)
is-c-NOCONVERSION	3-10 (76), 4-63 (246)
is-c-NOFIXEDOVERFLOW	3-10 (76), 4-63 (246)
is-c-NOLOCK	3-19 (131), 4-76 (302), 4-77 (303)
is-c-non-data-attribute	<u>3-7 (49)</u> , 3-4 (30)
is-c-NOOVERFLOW	3-10 (76), 4-63 (246)
is-c-NOSIZE	3-10 (76), 4-63 (246)
is-c-NOSTRINGRANGE	3-10 (76), 4-63 (246)
is-c-NOSTRINGSIZE	3-10 (76), 4-63 (246)
is-c-NOSUBSCRIPTRANGE	3-10 (76), 4-63 (246)
is-c-NOT	2-8 (24)
is-c-NUnderflow	3-10 (76), 4-63 (246)
is-c-NOZERODIVIDE	3-10 (76), 4-63 (246)
is-c-null-statement	<u>3-10 (79)</u> , 3-10 (78), 4-62 (241)
is-c-0	3-22 (149)
is-c-OFFSET	3-5 (38)
is-c-offset-attribute	<u>3-5 (38)</u> , 3-4 (31), 4-32 (119), 4-33 (122)
is-c-ON	3-15 (105)
is-c-on-statement	<u>3-15 (105)</u> , 3-10 (78), 4-62 (241), 4-72 (286), 4-72 (287)
is-c-OPEN	3-17 (118), 4-77 (305)
is-c-open-optionslist	<u>3-17 (119)</u> , 3-17 (118), 4-75 (300)
is-c-open-statement	<u>3-17 (118)</u> , 3-10 (78), 4-62 (241), 4-77 (306), 4-78 (307)

30 June 1969

TRANSLATION OF PL/I INTO ABSTRACT SYNTAX

is-c-OPTIONS	3-4 (25)
is-c-options-attribute	<u>3-4 (25)</u> , 3-1 (4), 3-11 (80)
is-c-OR	2-8 (24)
is-c-ORDER	3-1 (4), 3-11 (80), 4-7 (24)
is-c-OUTPUT	3-7 (54), 4-20 (74), 4-47 (188)
is-c-OVERFLOW	3-10 (75), 4-63 (246)
is-c-P	3-9 (69), 3-22 (149), 3-24 (166)
is-c-PAGE	3-9 (70), 3-18 (123), 4-49 (196), 4-50 (197), 4-76 (302), 4-77 (303)
is-c-PAGESIZE	3-17 (119), 4-76 (302)
is-c-parameterlist	<u>3-1 (3)</u> , 3-1 (2), 3-2 (8)
is-c-PENDING	3-16 (112), 4-73 (291)
is-c-PICTURE	3-5 (35)
is-c-picture-attribute	<u>3-5 (35)</u> , 3-4 (31), 4-32 (119), 4-37 (146)
is-c-picture-character	<u>3-24 (166)</u> , 3-24 (165)
is-c-picture-format	<u>3-9 (69)</u> , 3-9 (65), 3-9 (67), 4-50 (197)
is-c-picture-specification	<u>3-24 (164)</u> , 3-5 (35), 3-9 (69)
is-c-picture-string	<u>3-24 (165)</u> , 3-24 (164), 3-24 (165)
is-c-POINTER	3-4 (31), 4-32 (119)
is-c-POSITION	3-6 (40), 4-20 (74), 4-48 (190)
is-c-precision-spec	<u>3-3 (24)</u> , 3-3 (23), 4-34 (129), 4-34 (130)
is-c-prefix	<u>3-10 (75)</u> , 3-10 (74), 3-15 (108), 4-63 (245), 4-63 (246), 4-64 (249), 4-73 (290)
is-c-prefix-element	<u>3-10 (74)</u> , 3-10 (73), 4-63 (245), 4-64 (249)
is-c-prefixlist	<u>3-10 (73)</u> , 3-1 (2), 3-1 (6), 3-8 (60), 3-10 (72), 3-12 (88), 4-62 (242), 4-62 (243), 4-62 (244), 4-63 (248)
is-c-primitive-expression	<u>3-21 (141)</u> , 3-21 (140)
is-c-PRINT	3-7 (54), 4-47 (188)
is-c-PRIORITY	3-12 (91), 4-76 (302)
is-c-procedure	<u>3-1 (2)</u> , 2-6 (17), 2-7 (22), 3-1 (1), 3-1 (7), 4-4 (9), 4-4 (11), 4-4 (12), 4-5 (15), 4-5 (17), 4-6 (18), 4-7 (23), 4-11 (37), 4-11 (40), 4-43 (174)
is-c-PROCEDURE	3-1 (2)
is-c-procedure-optionslist	<u>3-1 (4)</u> , 3-1 (2)
is-c-program	<u>3-1 (1)</u> , 2-2 (1), 2-6 (15), 4-3 (6), 4-4 (9), 4-4 (11)
is-c-programmer-named-condition	<u>3-16 (113)</u> , 3-15 (108), 4-17 (66), 4-73 (290)
is-c-PT	2-8 (24)

TRANSLATION OF PL/I INTO ABSTRACT SYNTAX

30 JUNE 1969

is-c-PUT	3-17 (122), 4-16 (61), 4-77 (305)
is-c-Q	3-22 (149)
is-c-R	3-9 (71), 3-22 (149), 3-24 (166)
is-c-RANGE	3-3 (19)
is-c-range-spec	<u>3-3 (18)</u> , 3-3 (17), 4-21 (78)
is-c-READ	3-19 (130), 4-77 (305)
is-c-REAL	3-5 (32), 4-20 (74), 4-20 (77), 4-33 (124), 4-34 (130), 4-34 (131), 4-52 (206), 4-52 (209)
is-c-real-constant	<u>3-23 (154)</u> , 3-7 (48), 3-22 (146), 3-23 (157), 4-87 (342)
is-c-real-format	<u>3-9 (66)</u> , 3-9 (65), 3-9 (67), 4-50 (197)
is-c-RECORD	3-7 (54), 3-16 (112), 4-20 (74), 4-47 (188), 4-73 (291)
is-c-record-io-statement	<u>3-19 (130)</u> , 3-10 (78), 4-62 (241), 4-80 (315)
is-c-record-optionslist	<u>3-19 (131)</u> , 3-19 (130), 4-75 (300)
is-c-RECURSIVE	3-1 (4), 4-7 (25), 4-7 (26)
is-c-REDUCIBLE	3-7 (50), 4-7 (25), 4-46 (186)
is-c-REFER	3-4 (29)
is-c-refer-expression	<u>3-4 (29)</u> , 3-4 (28), 3-5 (34), 3-5 (36), 4-29 (106)
is-c-reference	<u>3-21 (142)</u> , 3-5 (38), 3-6 (41), 3-6 (43), 3-6 (45), 3-6 (46), 3-8 (56), 3-9 (71), 3-11 (84), 3-12 (89), 3-12 (90), 3-12 (91), 3-13 (94), 3-13 (95), 3-13 (96), 3-14 (100), 3-14 (102), 3-15 (104), 3-16 (111), 3-16 (116), 3-17 (119), 3-17 (121), 3-18 (123), 3-19 (131), 3-20 (132), 3-21 (141), 3-21 (142), 4-17 (65), 4-44 (179), 4-56 (224), 4-66 (260), 4-67 (261), 4-71 (284), 4-72 (285), 4-74 (298), 4-82 (320), 4-83 (323), 4-83 (324)
is-c-RELEASE	3-13 (95)
is-c-release-statement	<u>3-13 (95)</u> , 3-10 (78), 4-61 (241), 4-68 (269)
is-c-remote-format	<u>3-9 (71)</u> , 3-8 (64), 4-50 (197)
is-c-REORDER	3-1 (4), 3-11 (80), 4-7 (24), 4-7 (25)
is-c-replicated-string-constant	<u>3-22 (147)</u> , 3-6 (47), 3-22 (146), 4-87 (342), 4-89 (347)
is-c-REPLY	3-20 (132), 4-80 (317), 4-81 (318)
is-c-RETURN	3-13 (92)
is-c-return-statement	<u>3-13 (92)</u> , 3-10 (78), 4-61 (241), 4-67 (265), 4-72 (287)
is-c-RETURNS	3-4 (26), 4-45 (182), 4-45 (183)
is-c-returns-attribute	<u>3-4 (26)</u> , 3-1 (4), 3-2 (8), 3-7 (50), 4-6 (18), 4-10 (33), 4-46 (184)
is-c-REVERT	3-15 (106)
is-c-revert-statement	<u>3-15 (106)</u> , 3-10 (78), 4-62 (241), 4-72 (288)
is-c-REWRITE	3-19 (130), 4-77 (305)
is-c-S	3-22 (149), 3-24 (166)
is-c-scope-attribute	<u>3-8 (57)</u> , 3-4 (30)

30 June 1969

TRANSLATION OF PL/I INTO ABSTRACT SYNTAX

is-c-SECONDARY	3-4 (31), 4-26 (92)
is-c-sentence	3-1 (7), 2-7 (21), 3-1 (5), 4-7 (21)
is-c-sentencelist	3-1 (5), 3-1 (2), 3-11 (80), 3-11 (82), 3-11 (83)
is-c-SEQUENTIAL	3-7 (54), 4-20 (74), 4-47 (188)
is-c-SET	3-14 (102), 3-19 (131), 4-17 (65), 4-71 (281), 4-76 (302)
is-c-SIGNAL	3-15 (107)
is-c-signal-statement	3-15 (107), 3-10 (78), 4-62 (241), 4-72 (289)
is-c-signed-integer	3-5 (33), 3-5 (32), 4-81 (319), 4-87 (342), 4-89 (347)
is-c-simple-default-spec	3-3 (17), 3-3 (16)
is-c-simple-group	3-11 (82), 2-7 (22), 3-11 (81), 4-65 (252)
is-c-simple-string-constant . .	3-23 (158), 3-6 (45), 3-22 (146), 3-22 (147), 4-54 (217), 4-87 (342)
is-c-SIZE	4-63 (246)
is-c-SKIP	3-9 (70), 3-18 (123), 4-49 (196), 4-76 (302), 4-77 (303)
is-c-SNAP	3-15 (105)
is-c-space	2-4 (10), 2-3 (6), 2-3 (7), 2-9 (27), 2-9 (28)
is-c-specification	3-11 (85), 3-11 (84), 4-65 (253)
is-c-statement . .	3-10 (72), 3-1 (7), 3-12 (86), 3-16 (115), 4-6 (20), 4-11 (38), 4-60 (239), 4-66 (257)
is-c-STATIC	3-5 (39), 4-20 (74), 4-28 (101)
is-c-sterling-constant	3-24 (163), 3-6 (47), 3-22 (146), 4-87 (342), 4-89 (347)
is-c-STOP	3-13 (99)
is-c-stop-statement	3-13 (99), 3-10 (78), 4-62 (241)
is-c-storage-class-attribute	3-5 (39), 3-4 (31), 4-26 (92)
is-c-STREAM	3-7 (54), 4-20 (74), 4-47 (188)
is-c-stream-io-statement	3-17 (122), 3-10 (78), 4-62 (241), 4-78 (309), 4-78 (310)
is-c-stream-optionslist	3-18 (123), 3-17 (122), 4-75 (300)
is-c-STRING	3-18 (123), 4-16 (60), 4-16 (61), 4-76 (302)
is-c-string-attribute . .	3-5 (34), 3-3 (23), 3-4 (31), 3-14 (103), 4-27 (96), 4-32 (119), 4-35 (141), 4-36 (143), 4-36 (144)
is-c-string-character	3-24 (162), 3-23 (161)
is-c-string-format	3-9 (68), 3-9 (65), 4-50 (197)
is-c-STRINGRANGE	3-10 (75), 4-63 (246)
is-c-STRINGSIZE	3-10 (75), 4-63 (246)
is-c-SUB	3-23 (152)

TRANSLATION OF PL/I INTO ABSTRACT SYNTAX

30 June 1969

is-c-subscriptlist	3-22 (144), 3-22 (143)
is-c-SUBSCRIPTRANGE	3-10 (75), 4-63 (246)
is-c-SYSIN	4-85 (330)
is-c-SYSPRINT	4-85 (330)
is-c-SYSTEM	3-3 (20), 3-15 (105), 4-19 (69), 4-19 (70), 4-19 (71), 4-57 (227), 4-72 (287)	
is-c-T	3-22 (149), 3-24 (166)
is-c-TASK	3-4 (31), 3-12 (91), 4-16 (62), 4-32 (119), 4-76 (302), 4-77 (303)	
is-c-THEN	3-12 (87)
is-c-TITLE	3-17 (119), 4-76 (302)
is-c-TO	3-11 (85), 3-12 (89), 4-65 (255)
is-c-TRANSIENT	3-7 (54), 4-47 (188)
is-c-TRANSMIT	3-16 (112), 4-73 (291)
is-c-U	3-22 (149)
is-c-UNALIGNED	3-4 (31), 4-18 (68), 4-27 (97), 4-51 (204), 4-58 (231)
is-c-UNBUFFERED	3-7 (54), 4-20 (74), 4-47 (188)
is-c-unconditional-statement	3-10 (78), 3-10 (72), 3-12 (88), 3-15 (105)
is-c-UNDEFINEDFILE	3-16 (112), 4-73 (291)
is-c-UNDERFLOW	3-10 (75), 4-63 (246)
is-c-UNLOCK	3-19 (130), 4-77 (305)
is-c-unsubscripted-reference . . .	3-22 (145), 3-4 (29), 3-8 (58), 3-15 (109), 3-15 (110), 4-63 (247) 4-83 (323), 4-85 (331), 4-85 (333)	
is-c-UPDATE	3-7 (54), 4-20 (74), 4-47 (188)
is-c-V	3-22 (149), 3-24 (166)
is-c-VALUE	3-3 (21)
is-c-value-clause	3-3 (21), 3-3 (20), 4-10 (33), 4-20 (73), 4-20 (74), 4-20 (76), 4-20 (77), 4-34 (129), 4-36 (143), 4-40 (163)	
is-c-value-spec	3-3 (23), 3-3 (21)
is-c-VARIABLE	3-4 (31)
is-c-VARYING	3-4 (31), 4-32 (119), 4-36 (145)
is-c-volume	4-76 (302)
is-c-VOLUME	3-17 (119), 3-17 (121)
is-c-W	3-22 (149)
is-c-WAIT	3-13 (96)
is-c-wait-statement	3-13 (96), 3-10 (78), 4-17 (63), 4-62 (241), 4-68 (270)

30 June 1969

TRANSLATION OF PL/I INTO ABSTRACT SYNTAX

is-c-WHEN	3-8 (56)
is-c- WHILE	3-11 (83), 3-11 (85), 4-65 (252)
is-c-WRITE	3-19 (130), 4-77 (305)
is-c-X	3-9 (70), 3-22 (149), 3-24 (166)
is-c-Y	3-22 (149), 3-24 (166)
is-c-Z	3-22 (149), 3-24 (166)
is-c-ZERODIVIDE	3-10 (75), 4-63 (246)
is-call-st	<u>5-11 (96)</u> , 5-5 (39), 5-7 (63), 5-9 (87)
is-char-pic	<u>5-8 (73)</u> , 5-7 (66)
is-char-spec	<u>5-8 (74)</u> , 5-8 (73)
is-char-val	<u>2-3 (5)</u> , <u>5-18 (168)</u> , 2-3 (4), 2-4 (12), 5-17 (161)
is-check-cond	<u>5-13 (117)</u> , 5-13 (116)
is-close	<u>5-14 (129)</u> , 5-14 (128)
is-close-st	<u>5-14 (128)</u> , 5-9 (87)
is-compat(ad-1,ad-2)	<u>4-19 (72)</u> , 4-19 (70)
is-compl(x,y)	<u>4-20 (74)</u> , 4-20 (73)
is-cond	<u>5-12 (115)</u> , 5-12 (112), 5-12 (113), 5-12 (114)
is-cond-cont(p)	<u>4-17 (66)</u> , 4-18 (67)
is-cond-part	<u>5-1 (7)</u> , 5-1 (3), 5-3 (21), 5-9 (86), 5-10 (89)
is-const	<u>5-17 (159)</u> , 5-17 (151)
is-contained-in(b,p)	<u>4-4 (11)</u> , 4-4 (10), 4-4 (13)
is-contr-group	<u>5-10 (92)</u> , 5-10 (90)
is-contr-item	<u>5-15 (140)</u> , 5-15 (139)
is-control-format	<u>5-9 (83)</u> , 5-8 (77)
is-control-format-type	<u>5-9 (84)</u> , 5-9 (83)
is-cplx-format	<u>5-9 (80)</u> , 5-8 (78)
is-data-attr-descr(ad)	<u>4-25 (90)</u> , 4-24 (87), 4-25 (91)
is-data-format	<u>5-8 (78)</u> , 5-8 (77)
is-data-matching(p,pd)	<u>4-34 (130)</u> , 4-34 (129)
is-data-spec	<u>5-15 (136)</u> , 5-14 (131), 5-14 (132), 5-15 (134), 5-15 (135)
is-data-type	<u>4-24 (88)</u> , 4-24 (87), 4-42 (167)
is-dec-flt-default(pd)	<u>4-35 (136)</u> , 4-33 (125), 4-33 (126)
is-dec-pic	<u>5-7 (67)</u> , 5-7 (66)
is-dec-spec	<u>5-7 (68)</u> , 5-7 (67), 5-8 (70)

TRANSLATION OF PL/I INTO ABSTRACT SYNTAX

30 June 1969

is-decl	5-2(10), 5-2(9)
is-decl-area-attr(ad, pd)	4-40(162), 4-40(161)
is-decl-cont(p)	4-11(36), 4-3(4), 4-9(29), 4-11(35), 4-12(42), 4-12(43), 4-12(44), 4-13(46), 4-13(47), 4-14(48), 4-14(49), 4-14(50), 4-15(53), 4-15(54), 4-15(55), 4-18(67), 4-31(113)
is-decl-part	5-2(9), 5-1(1), 5-1(3), 5-10(89)
is-def-area-size(p, pd)	4-40(163), 4-40(161)
is-def-prec(p, pd)	4-34(129), 4-34(127), 4-35(134)
is-def-string-length(p, pd)	4-36(143), 4-36(142)
is-default-attr(p)	4-20(76), 4-20(75), 4-57(228)
is-defined	5-2(12), 5-2(10)
is-delay-st	5-11(104), 5-9(87)
is-delete-st	5-16(148), 5-9(87)
is-dens-attr(attr, pd)	4-27(97), 4-27(96), 4-51(203)
is-descr	5-2(15), 5-2(14), 5-4(36)
is-descr-aggr	5-5(43), 5-2(15), 5-5(44), 5-5(46)
is-descr-area	5-6(50), 5-5(48)
is-descr-array	5-5(44), 5-5(43)
is-descr-attr-of(p, ad)	4-58(230), 4-57(229), 4-58(231)
is-descr-da	5-5(48), 5-5(47)
is-descr-dens(p, ad)	4-58(231), 4-57(229), 4-58(231)
is-descr-scalar	5-5(47), 5-1(5), 5-2(14), 5-4(36), 5-5(43)
is-descr-string	5-6(49), 5-5(48)
is-descr-struct	5-5(45), 5-5(43)
is-descr-subel-of(p, q, p-list)	4-58(233), 4-58(232)
is-descr-succ	5-5(46), 5-5(45)
is-descr-succ-of(p, q)	4-58(232), 4-31(115), 4-58(230), 4-58(231), 4-58(235), 4-59(236), 4-59(238)
is-disable-st	5-13(125), 5-9(87)
is-display-st	5-16(150), 5-9(87)
is-do-spec	5-10(93), 5-10(92), 5-15(140)
is-edit-dir	5-15(138), 5-15(136)
is-enable	5-13(124), 5-13(123)
is-enable-st	5-13(123), 5-9(87)
is-entry-const	5-2(14), 5-2(10)

30 June 1969

TRANSLATION OF PL/I INTO ABSTRACT SYNTAX

TRANSLATION OF PL/I INTO ABSTRACT SYNTAX

30 June 1969

is-generic-array	5-6(52), 5-6(51)
is-generic-da	5-6(56), 5-6(55)
is-generic-descr	5-3(24), 5-3(23)
is-generic-member	5-3(23), 5-3(22)
is-generic-scalar	5-6(55), 5-6(51)
is-generic-string	5-6(58), 5-6(56)
is-generic-struct	5-6(53), 5-6(51)
is-generic-succ	5-6(54), 5-6(53)
is-get-st	5-14(130), 5-9(87)
is-goto-st	5-10(95), 5-9(87)
is-group	5-10(90), 5-9(87)
is-i-to-n(ch)	4-35(138), 4-35(136)
is-id-list-match(idl-1,idl-2)	4-84(329), 4-14(51), 4-84(327), 4-84(329)
is-id-ref	5-1(6), 5-1(5), 5-4(37), 5-9(86), 5-13(121)
is-if-st	5-10(94), 5-9(87)
is-ignore-read-st	5-16(144), 5-15(141)
is-in-letter-range-of(obj,id)	4-21(79), 4-21(78), 4-21(79)
is-in-range-of(p,id)	4-21(78), 4-20(75)
is-incompat(x,y)	4-20(73), 4-19(72)
is-incorporate-st	5-11(99), 5-9(87)
is-incorporate-text	5-11(100), 5-11(99)
is-index	5-3(20), 5-1(5), 5-3(19)
is-infix-expr	5-17(152), 5-17(151)
is-infix-opr	5-17(153), 5-17(152)
is-init	5-5(39), 5-4(31)
is-init-elem	5-5(40), 5-5(39), 5-5(41), 5-7(63)
is-init-iter	5-5(41), 5-5(40)
is-init-label-cont(p)	4-55(221), 4-54(215), 4-55(219)
is-iatg-const	4-30(109), 4-29(108)
is-intg-val	5-18(166), 4-30(109), 5-3(20), 5-4(33), 5-5(42), 5-5(44), 5-6(49), 5-6(50), 5-6(57), 5-7(67), 5-8(69), 5-8(71), 5-18(162), 5-18(163)	
is-into-read-st	5-15(142), 5-15(141)
is-invalid-descr-ln(p)	4-58(235), 4-56(222)
is-invalid-like(p)	4-15(54), 4-9(29)

30 June 1969

TRANSLATION OF PL/I INTO ABSTRACT SYNTAX

is-invalid-ln(p)	4-13(47), 4-9(29)
is-io-cond	5-13(119), 5-13(118)
is-isub	5-18(163), 5-17(151)
is-item	5-15(139), 5-15(137), 5-15(138), 5-15(140)
is-label-attr-p(ad, pd)	4-41(165), 4-41(164)
is-label-const	5-3(19), 5-2(10)
is-label-cont(p)	4-11(38), 4-11(35), 4-18(67), 4-55(221)
is-label-da	5-4(37), 5-4(32)
is-labeled(x, c-id)	2-7(18), 2-6(16)
is-like(p)	4-14(49), 4-13(48), 4-15(54)
is-like-succ-of(p, q)	4-13(48), 4-12(42), 4-15(54)
is-list-data-dir	5-15(137), 5-15(136)
is-ln-epd(b, pd)	4-14(52), 4-14(51)
is-ln-of(p, q)	4-13(46), 4-13(45)
is-ln-subel-of(p, q)	4-12(44), 4-12(43), 4-14(52), 4-15(54)
is-ln-succ-of(p, q)	4-12(43), 4-12(42), 4-13(47), 4-14(48), 4-15(54)
is-local-sentence(p, q)	4-7(21), 4-6(20), 4-7(23)
is-local-to(b, p)	4-4(10), 4-4(8), 4-5(15), 4-9(29), 4-20(75), 4-57(228)
is-locate-st	5-16(147), 5-9(87)
is-mult-decl(seq-1, seq-2)	4-15(55), 4-9(29)
is-named-io-cond	5-13(118), 5-12(115)
is-non-data-type	4-25(89), 4-24(87)
is-non-expl-decl(p)	4-16(57), 4-9(29), 4-9(30), 4-10(31), 4-16(58), 4-16(59), 4-16(62), 4-17(63), 4-17(64), 4-17(65), 4-84(326)
is-null-st	5-10(88), 5-9(87)
is-num-val	5-18(167), 5-17(160)
is-offset	5-5(38), 5-4(32), 5-5(48)
is-on-st	5-12(112), 5-9(87)
is-open	5-14(127), 5-14(126)
is-open-st	5-14(126), 5-9(87)
is-opt	5-1(8), 5-1(3), 5-2(11), 5-2(14), 5-2(15), 5-4(36), 5-10(89), 5-12(107), 5-12(112), 5-14(127), 5-14(129), 5-14(131), 5-15(134), 5-15(142)
is-opt-expr	5-18(164), 5-2(12), 5-9(79), 5-9(81), 5-9(83), 5-10(93), 5-11(97), 5-11(98), 5-11(103), 5-14(127), 5-14(131), 5-15(134), 5-15(142), 5-16(143), 5-16(144), 5-16(145), 5-16(146), 5-16(147), 5-16(148)
is-opt-keyw(q, p)	4-75(301), 4-75(300)

TRANSLATION OF PL/I INTO ABSTRACT SYNTAX

30 June 1969

is-opt-ref	5-18 (165), 5-2 (13), 5-12 (109), 5-12 (111), 5-13 (124), 5-15 (142), 5-16 (143), 5-16 (144), 5-16 (145), 5-16 (146), 5-16 (147), 5-16 (148), 5-16 (150), 5-17 (157)
is-pa-opt	5-11 (97), 4-67 (262), 5-11 (96)
is-param-cont(p)	4-11 (37), 4-11 (35), 4-15 (55), 4-18 (67)
is-paren-expr	5-17 (156), 5-17 (151)
is-pic	5-7 (66), 4-37 (148), 5-4 (32), 5-5 (48), 5-6 (56), 5-9 (82)
is-pic-format	5-9 (82), 5-8 (78), 5-9 (80)
is-predec-dens(seq,ad)	4-18 (68), 4-10 (31), 4-18 (68)
is-prefix-cond	5-13 (116), 5-1 (7), 5-12 (115)
is-prefix-expr	5-17 (154), 5-17 (151)
is-prefix-opr	5-17 (155), 5-17 (154)
is-prel-decl	4-10 (32), 4-29 (107), 4-30 (112), 4-79 (311)
is-progr-named-cond	5-13 (120), 5-12 (115)
is-prop-aggr	5-3 (26), 5-2 (11), 5-2 (12), 5-2 (13), 5-3 (27), 5-4 (30)
is-prop-area	5-4 (35), 5-4 (32)
is-prop-arithm	5-4 (33), 5-4 (32), 5-5 (48), 5-17 (160)
is-prop-array	5-3 (27), 5-3 (26)
is-prop-da	5-4 (32), 5-4 (31)
is-prop-scalar	5-4 (31), 5-3 (26)
is-prop-st	5-9 (87), 4-77 (304), 5-9 (86)
is-prop-string	5-4 (34), 5-4 (32)
is-prop-struct	5-4 (29), 5-3 (26)
is-prop-succ	5-4 (30), 5-4 (29)
is-prop-var	5-2 (11), 5-2 (10)
is-proper-program	5-1 (1), 4-3 (5), 4-3 (6)
is-ptr-cont(p)	4-17 (65), 4-18 (67)
is-put-st	5-14 (133), 5-9 (87)
is-read-st	5-15 (141), 5-9 (87)
is-real-format	5-9 (79), 5-8 (78), 5-9 (80)
is-ref	5-17 (157), 5-2 (12), 5-3 (23), 5-5 (38), 5-9 (85), 5-10 (92), 5-10 (95), 5-11 (96), 5-11 (97), 5-11 (101), 5-11 (102), 5-11 (103), 5-12 (107), 5-12 (111), 5-13 (117), 5-13 (118), 5-14 (127), 5-14 (129), 5-14 (131), 5-15 (134), 5-15 (135), 5-15 (140), 5-15 (142), 5-16 (143), 5-16 (144), 5-16 (145), 5-16 (146), 5-16 (147), 5-16 (148), 5-16 (149), 5-17 (151), 5-18 (165)
is-ref-cont(p)	4-83 (325), 4-16 (57), 4-83 (323), 4-83 (324), 4-85 (330), 4-85 (332), 4-85 (333), 4-85 (334), 4-86 (335), 4-86 (338)
is-refer	5-4 (28), 5-3 (27), 5-4 (34), 5-4 (35)

30 June 1969

TRANSLATION OF PL/I INTO ABSTRACT SYNTAX

TRANSLATION OF PL/I INTO ABSTRACT SYNTAX

30 June 1969

30 June 1969

TRANSLATION OF PL/I INTO ABSTRACT SYNTAX

TRANSLATION OF PL/I INTO ABSTRACT SYNTAX

30 June 1969

mk-env-attr(p)	4-23(82), 4-23(81), 4-23(83), 4-47(187), 4-77(303)
mk-event(p)	4-81(318), 4-80(316)
mk-event-ref(p)	4-74(298), 4-74(295)
mk-file-decl(pd)	4-47(187), 4-23(81)
mk-format-decl(pd)	4-48(193), 4-23(81)
mk-free-ref(p,b)	4-72(285), 4-71(284)
mk-gen-aggr(pd,bdpl)	4-51(202), 4-50(201), 4-51(202), 4-51(203)
mk-gen-area(pd)	4-53(214), 4-51(204)
mk-gen-arithm(pd)	4-51(205), 4-51(204)
mk-gen-da(pd)	4-51(204), 4-51(203)
mk-gen-descr(pd)	4-50(201), 4-50(200)
mk-gen-member(p)	4-50(200), 4-50(198)
mk-gen-non-array(pd)	4-51(203), 4-51(202)
mk-gen-string(pd)	4-52(212), 4-51(204)
mk-generic-decl(pd)	4-50(198), 4-23(81)
mk-id(chl)	4-90(354), 4-10(34), 4-19(71), 4-20(75), 4-21(78), 4-21(79), 4-35(137), 4-85(330), 4-90(353)
mk-id-ref(p)	4-6(19), 4-6(18), 4-41(164), 4-64(250), 4-73(292)
mk-id-1(p)	4-90(353), 4-3(4), 4-5(15), 4-5(17), 4-6(19), 4-10(31), 4-21(78), 4-29(106), 4-45(182), 4-46(185), 4-48(193), 4-55(220), 4-59(237), 4-84(327), 4-85(330), 4-90(352)
mk-indexlist(p,q)	4-6(20), 4-6(18), 4-6(20), 4-23(81)
mk-init(pd)	4-54(215), 4-24(86), 4-24(87), 4-32(117), 4-48(190), 4-70(279)
mk-list(a,b,list)	2-4(8), 2-3(7), 2-4(8), 2-9(28)
mk-no-pref-list(p,n)	4-63(248), 4-62(243), 4-63(248)
mk-no-pref-list-1(p,n)	4-64(249), 4-63(248), 4-64(249)
mk-on-pref-list(p,n)	4-62(244), 4-62(243), 4-62(244)
mk-on-pref-list-1(p,n)	4-63(245), 4-62(244), 4-63(245)
mk-opt(x)	4-69(273), 4-69(272), 4-72(286)
mk-opt-1(p)	4-7(25), 4-7(24), 4-7(26)
mk-par-p(p)	4-43(174), 4-42(171)
mk-pref(pref)	4-63(246), 4-63(245), 4-64(249), 4-73(290)
mk-prop-var-decl(pd)	4-23(84), 4-23(81)
mk-ptr-ref(p)	4-71(281), 4-70(276)
mk-recursive(b)	4-7(26), 4-5(16)
mk-red(pd)	4-46(186), 4-32(120), 4-41(166)

30 June 1969

TRANSLATION OF PL/I INTO ABSTRACT SYNTAX

mk-refer-expr(pd,p)	4-29(106), 4-28(104), 4-28(105), 4-36(142), 4-40(161)
mk-reorder(b)	4-7(24), 4-5(16), 4-7(24), 4-64(251)
mk-reply(p)	4-80(317), 4-80(316)
mk-ret-type(pd)	4-45(182), 4-41(166)
mk-ret-type-1(pd)	4-45(183), 4-6(18), 4-32(120), 4-45(182)
mk-ret-type-2(pd)	4-46(184), 4-45(182), 4-45(183)
mk-st-list(p)	4-7(22), 4-5(16), 4-64(251), 4-65(252)
mk-succ(pd)	4-30(111), 4-30(110)
mk-to-expr(p)	4-65(255), 4-65(253)
MULT	4-82(321), 5-17(153)
N-CHAR	2-8(26), 4-22(80), 4-22(80), 4-35(138)
NAME	4-73(291), 5-13(119)
NE	4-82(321), 5-17(153)
non-array-aggr(pd)	4-24(86), 4-23(85)
NOT	2-4(9), 2-8(24), 2-8(26), 2-8(26), 3-21(136), 3-21(140), 3-24(162), 4-82(321), 5-17(155)
NULL	4-60(239), 4-62(241), 4-66(257), 5-10(88)
NUMBER-SIGN	2-8(24), 3-22(149), 4-21(79)
O-CHAR	2-8(26), 4-22(80), 4-22(80)
OFFSET	4-31(116), 4-32(119), 4-33(121), 4-44(179), 4-44(179), 4-51(204), 5-5(38), 5-6(56)
offset-attr-p(pd)	4-33(122), 4-33(121), 4-44(179)
offset-da(pd)	4-33(121), 4-31(116)
OFL	4-63(246), 5-13(116)
ON	4-72(286), 5-12(112)
OPEN	4-77(305), 5-14(126)
opt-sel(opt)	4-76(302), 4-75(300), 4-75(301)
OR	2-4(9), 2-8(24), 2-8(26), 2-8(26), 3-20(133), 3-21(137), 3-24(162), 4-82(321), 5-17(153)
order-set(set,obj)	4-79(313), 4-79(311)
OUT	4-47(188), 5-3(17)
P-CHAR	2-8(26), 4-22(80), 4-22(80), 4-40(160), 4-49(196)
PAGE	4-49(196), 5-9(84)
PARAM	4-10(33), 4-18(67), 4-26(92), 4-26(93), 4-27(94), 4-27(95), 4-27(98), 4-27(99), 4-29(106), 4-32(117), 4-46(184), 4-58(230), 4-79(311), 5-2(11)
parse(text)	2-2(1)
pd-block-p(pd)	4-29(107), 4-29(106), 4-36(142), 4-40(161), 4-54(215)

TRANSLATION OF PL/I INTO ABSTRACT SYNTAX

30 June 1969

pd-seq(pd)	4-31(113), 4-23(81), 4-29(106), 4-30(112), 4-48(194), 4-86(335)
PEND	4-73(291), 5-13(119)
PERC	2-8(24), 3-24(162)
PIC	4-31(116), 4-32(117), 4-32(119), 4-49(196), 4-51(204), 5-9(82)
pic-da(pd)	4-37(146), 4-31(116), 4-51(204)
pic-mode(pd)	4-37(147), 4-37(146)
PLUS	2-4(9), 3-5(33), 3-7(48), 3-21(138), 3-21(140), 3-23(156), 3-24(162), 3-24(164), 3-24(166), 4-29(108), 4-82(321), 5-7(68), 5-17(155)
POINT	2-4(9), 2-8(26), 2-9(28), 2-9(29), 3-22(143), 3-22(145), 3-23(155), 3-24(162), 3-24(163), 3-24(166), 4-10(34), 5-7(68), 5-8(74)
pos-el(el,list)	4-38(153), 4-39(156), 4-39(158)
pos-v(pcl,n)	4-39(156), 4-39(155), 4-39(158)
prec-p(pd)	4-34(128), 4-33(126), 4-34(127), 4-35(134), 4-37(147), 4-51(205)
prel-decl(x)	4-9(30), 4-3(4), 4-6(18), 4-14(52), 4-15(53), 4-29(106), 4-30(112), 4-31(113), 4-43(172), 4-43(173), 4-46(185), 4-84(326), 4-84(328), 4-86(335)
prel-decl-set(b)	4-9(28), 4-8(27), 4-14(52), 4-84(328)
prel-decl-set-1(b)	4-9(29), 4-9(28)
prel-decl-1(x)	4-10(31), 4-9(29), 4-9(30)
prel-descr(p)	4-57(226), 4-30(112), 4-56(222)
prel-descr-list(p)	4-56(222), 4-42(168), 4-42(171), 4-50(200), 4-69(275)
proc-p(p)	4-4(12), 4-5(17), 4-6(18), 4-46(185), 4-84(327)
progr-decl-part(b)	4-3(3), 4-2(2), 4-3(4)
prop-arg-p-list(b,p)	4-86(340), 4-86(339), 4-86(340)
PROP-VAR	4-19(72), 4-23(81), 4-24(88), 4-25(91), 4-26(92), 4-45(182), 4-79(311)
PRT	4-47(188), 5-3(17)
PTR	4-10(33), 4-18(67), 4-25(90), 4-32(119), 4-32(119), 4-51(204), 5-4(32), 5-5(48), 5-6(56)
PUT	4-77(305), 5-15(134), 5-15(135)
Q-CHAR	4-22(80), 4-22(80)
QUEST	2-8(24), 3-24(162)
R-CHAR	2-8(26), 4-22(80), 4-22(80), 4-49(196), 5-7(68)
READ	4-77(305), 5-15(142), 5-16(143), 5-16(144)
REAL	4-30(109), 4-33(124), 4-34(131), 4-46(184), 4-52(209), 4-88(343), 5-4(33), 5-6(57)
REC	4-47(188), 4-73(291), 5-3(17), 5-13(119)
ref-id-list(p)	4-84(330), 4-10(31), 4-14(50), 4-16(57), 4-78(310), 4-84(326)
RELEASE	4-68(269), 5-11(102)

30 June 1969

TRANSLATION OF PL/I INTO ABSTRACT SYNTAX

REMOTE 4-49(196), 5-9(85)
replace-48(x) <u>2-8(25)</u> , 2-8(24)
replace-48-1(x) <u>2-8(26)</u> , 2-8(25)
rest-da-1(da,v) <u>4-88(344)</u> , 4-87(342)
RETURN 4-67(265), 5-11(98)
REVERT 4-72(288), 5-12(113)
REWRITE 4-77(305), 5-16(146)
RIGHT-PAR	2-4(9), 3-1(3), 3-2(12), 3-3(19), 3-3(21), 3-3(22), 3-3(24), 3-4(25), 3-4(26), 3-4(27), 3-4(29), 3-5(32), 3-5(34), 3-5(36), 3-5(37), 3-5(38), 3-6(40), 3-6(41), 3-6(44), 3-6(45), 3-6(46), 3-7(50), 3-7(53), 3-7(55), 3-8(56), 3-8(59), 3-8(61), 3-8(63), 3-9(66), 3-9(67), 3-9(68), 3-9(70), 3-9(71), 3-10(73), 3-11(83), 3-11(85), 3-12(91), 3-13(92), 3-13(93), 3-13(96), 3-13(97), 3-14(102), 3-15(104), 3-15(109), 3-15(110), 3-16(111), 3-16(113), 3-16(114), 3-16(115), 3-16(116), 3-17(119), 3-17(121), 3-18(123), 3-18(125), 3-18(126), 3-18(127), 3-19(129), 3-19(131), 3-20(132), 3-21(141), 3-22(144), 3-22(147), 3-24(162), 3-24(164), 3-24(165)
s(n,x) <u>1-2(4)</u>
S-CHAR 4-22(80), 4-22(80), 4-40(160), 4-40(160), 5-8(70)
scale-f(scf) <u>4-38(150)</u> , 4-37(148)
scope-attr(pd)	<u>4-26(93)</u> , 4-3(4), 4-23(84), 4-24(87), 4-27(98), 4-27(99), 4-29(106), 4-32(117), 4-41(166), 4-42(167), 4-45(182), 4-46(185), 4-47(187)
scope-attr-1(ad) <u>4-27(94)</u> , 4-26(93)
SEMIC	2-4(9), 2-6(16), 2-7(23), 2-8(24), 2-8(26), 3-1(2), 3-1(6), 3-2(8), 3-2(10), 3-2(14), 3-2(15), 3-8(60), 3-10(79), 3-11(80), 3-11(82), 3-11(83), 3-12(89), 3-12(90), 3-13(92), 3-13(96), 3-13(97), 3-13(98), 3-13(99), 3-14(100), 3-14(101), 3-15(104), 3-15(105), 3-15(106), 3-15(107), 3-16(115), 3-17(118), 3-17(120), 3-17(122), 3-19(130), 3-20(132), 3-24(162), 4-74(293)
sent-list-p(x) <u>2-7(22)</u> , 2-7(19), 2-7(20)
SEQ 4-47(188), 5-3(17)
sgn(u) <u>4-89(350)</u> , 4-38(150), 4-43(176), 4-89(347)
SIGN 4-40(160), 5-7(68), 5-8(72)
SIGNAL 4-72(289), 5-12(114)
SIZE 4-63(246), 5-13(116)
SKIP 4-49(196), 5-9(84)
SLASH	2-4(9), 2-4(11), 2-4(12), 3-21(139), 3-24(162), 3-24(166), 4-82(321), 5-7(68), 5-8(74)
slength(x) <u>1-1(3)</u>
SPACE 4-49(196), 5-9(84)
st-opt-test(type,stmt) <u>4-77(304)</u> , 4-77(306), 4-78(308), 4-78(309), 4-80(315)
st-p-list(p) <u>4-7(23)</u> , 4-7(22)
STATIC 4-27(99), 4-28(101), 5-2(11)
stg-cl-attr-1(pd) <u>4-27(99)</u> , 4-27(95), 4-27(98)

TRANSLATION OF PL/I INTO ABSTRACT SYNTAX

30 June 1969

stg-cl-attr-2(ad)	4-28(101), 4-27(100)
stg-class-attr(pd)	4-27(98), 4-23(84), 4-29(106), 4-32(117), 4-43(172), 4-43(176), 4-50(201)
STOP	4-62(241), 5-12(106)
STRG	4-63(246), 5-13(116)
STRING	4-31(116), 4-32(119), 4-44(179), 4-51(204), 4-70(279)
string-base(pd)	4-35(140), 4-35(139), 4-36(144)
string-da(pd)	4-35(139), 4-31(116), 4-45(180), 4-70(279)
string-length(pd)	4-36(142), 4-35(139)
STRUCT	4-10(33), 4-18(67), 4-24(86), 4-25(90), 4-27(95), 4-32(117), 4-32(119), 4-32(119), 4-44(177), 4-51(203), 4-58(230), 4-59(236), 4-70(276), 4-70(278), 4-79(311)
struct-aggr(pd)	4-30(110), 4-24(86)
STRZ	4-63(246), 5-13(116)
SUBRG	4-63(246), 5-13(116)
subscr-nb(p,n)	4-86(337), 4-86(335), 4-86(337)
subscr-p-list(p)	4-85(333), 4-85(332)
SUBTR	4-82(321), 5-17(153)
succ(char)	4-22(80), 4-21(79)
SUCC	4-10(33), 4-18(67), 4-24(88), 4-25(90), 4-26(92), 4-26(92), 4-27(94), 4-27(95), 4-30(111), 4-58(230), 4-59(236)
succ-list(pd)	4-30(112), 4-30(110), 4-44(177), 4-51(203), 4-70(278), 4-71(280)
succ-p-list(p)	4-31(114), 4-30(112)
SYSTEM	4-72(287), 5-12(112)
T-CHAR	2-8(26), 4-22(80), 4-22(80), 5-7(68)
TASK	4-10(33), 4-18(67), 4-25(90), 4-32(119), 4-32(119), 4-51(204), 5-4(32), 5-5(48), 5-6(56)
TMT	4-73(291), 5-13(119)
TRA	4-47(188), 5-3(17)
trans-access-st(p)	4-74(293), 4-62(241)
trans-allocate-list(p)	4-69(275), 4-69(274)
trans-allocate-st(p)	4-69(274), 4-62(241)
trans-assign-st(p)	4-69(272), 4-62(241)
trans-attr(attr)	4-52(209), 4-52(206), 4-52(207), 4-52(208), 4-53(213)
trans-block(b)	4-64(251), 4-61(241)
trans-body(b)	4-5(16), 4-5(15)
trans-close-st(p)	4-78(308), 4-62(241)
trans-cond(p)	4-73(290), 4-72(286), 4-72(288), 4-72(289)

30 June 1969

TRANSLATION OF PL/I INTO ABSTRACT SYNTAX

trans-cond-part(p)	4-62(243), 4-5(16), 4-62(242), 4-64(251)
trans-data-spec(p)	4-79(311), 4-76(303)
trans-delay-st(p)	4-68(271), 4-62(241)
trans-disable-st(p)	4-74(299), 4-62(241)
trans-display-st(p)	4-80(316), 4-62(241)
trans-else-st(p)	4-66(257), 4-66(256), 4-74(293)
trans-enable-st(p)	4-74(294), 4-62(241)
trans-expr(p) . . .	4-81(319), 4-49(196), 4-50(197), 4-65(252), 4-65(253), 4-65(254), 4-65(255), 4-66(256), 4-68(271), 4-69(272), 4-77(303), 4-80(314), 4-80(316), 4-82(322)
trans-fetch-st(p)	4-68(268), 4-61(241)
trans-file-attr(c-fa)	4-47(188), 4-47(187), 4-78(307)
trans-format(p)	4-49(196), 4-48(195), 4-50(197)
trans-format-item(p)	4-50(197), 4-49(196)
trans-format-list(p)	4-48(195), 4-48(193), 4-49(196), 4-79(311)
trans-free(p)	4-71(284), 4-71(283)
trans-free-st(p)	4-71(283), 4-62(241)
trans-goto-st(p)	4-66(258), 4-61(241)
trans-group(p)	4-65(252), 4-61(241)
trans-id(p)	4-90(351), 4-70(276), 4-73(290), 4-77(303), 4-80(315)
trans-if-st(p)	4-66(256), 4-61(241)
trans-incorporate-spec(x)	4-67(267), 4-67(266)
trans-incorporate-st(p)	4-67(266), 4-61(241)
trans-init-label(p)	4-55(220), 4-55(219)
trans-io-cond(x)	4-73(291), 4-73(290)
trans-io-st-type(type)	4-77(305), 4-77(304)
trans-item(p)	4-80(314), 4-79(312)
trans-item-list(p)	4-79(312), 4-79(311), 4-80(314)
trans-label-list(p)	4-64(250), 4-60(239)
trans-num-pic(pcl,sf)	4-39(158), 4-38(154)
trans-on-st(p)	4-72(286), 4-62(241)
trans-on-unit(p)	4-72(287), 4-72(286)
trans-open-options(p)	4-78(307), 4-77(306)
trans-open-st(p)	4-77(306), 4-62(241)
trans-opt-expr(p) . . .	4-82(322), 4-48(190), 4-50(197), 4-65(253), 4-65(254), 4-65(255), 4-67(265), 4-68(270), 4-71(284), 4-80(317), 4-81(318)

TRANSLATION OF PL/I INTO ABSTRACT SYNTAX

30 June 1969

trans-option(p)	4-76(303), 4-75(300)
trans-optionslist(p)	4-75(300), 4-67(262), 4-78(307), 4-78(308), 4-78(309), 4-80(315)
trans-pa-opt(p)	4-67(262), 4-66(259)
trans-pcl(pcl)	4-39(159), 4-39(155), 4-39(158)
trans-pic(c-pic-spec, mode)	4-37(148), 4-37(146), 4-50(197)
trans-pic-el(el)	4-40(160), 4-39(159)
trans-pic-1(pcl, sf, mode)	4-38(151), 4-37(148)
trans-pic-2(pcl, sf)	4-38(154), 4-38(151)
trans-prop-st(p)	4-61(241), 4-60(239)
trans-record-io-st(p)	4-80(315), 4-62(241)
trans-ref(p)	4-83(323), 4-16(58), 4-48(190), 4-48(192), 4-50(197), 4-50(200), 4-55(220), 4-55(221), 4-63(247), 4-65(252), 4-66(258), 4-68(270), 4-69(272), 4-71(281), 4-71(282), 4-80(314), 4-80(317), 4-90(353)
trans-release-st(p)	4-68(269), 4-62(241)
trans-return-st(p)	4-67(265), 4-61(241)
trans-revert-st(p)	4-72(288), 4-62(241)
trans-signal-st(p)	4-72(289), 4-62(241)
trans-spec(p)	4-65(253), 4-74(297), 4-65(252), 4-74(296), 4-80(314)
trans-spec-init-list(pd)	4-55(219), 4-54(215)
trans-st(p)	4-60(239), 4-7(22), 4-66(256), 4-66(257), 4-72(287)
trans-st-cond-part(p)	4-62(242), 4-48(193), 4-60(239)
trans-sterling-pic(pcl)	4-39(155), 4-38(154)
trans-stream-io-st(p)	4-78(309), 4-62(241)
trans-stream-options(p,stmt)	4-78(310), 4-78(309)
trans-wait-st(p)	4-68(270), 4-62(241)
translate(t)	4-2(1)
translate-1(t)	4-2(2), 4-2(1), 4-3(6)
type-attr(pd)	4-24(87), 4-3(4), 4-23(81), 4-29(106), 4-30(111), 4-42(167), 4-45(182), 4-79(311)
type-attr-1(pd)	4-25(91), 4-24(87), 4-26(93)
type-attr-2(ad)	4-26(92), 4-19(72), 4-25(91)
U-CHAR	4-22(80), 4-22(80)
UFL	4-63(246), 5-13(116)
UNAL	4-27(96), 4-27(97), 5-4(31), 5-5(47), 5-6(55)
UNB	4-47(188), 5-3(17)

30 June 1969

TRANSLATION OF PL/I INTO ABSTRACT SYNTAX

UNDF	4-73 (291), 5-13 (119)
UNLOCK	4-77 (305), 5-16 (149)
UPD	4-47 (188), 5-3 (17)
V-CHAR	4-22 (80), 4-22 (80), 4-39 (156), 4-40 (160)
VAR	4-36 (145), 5-4 (34), 5-6 (49), 5-6 (58)
varying-attr(pd)	4-36 (145), 4-35 (139), 4-52 (212)
W-CHAR	4-22 (80), 4-22 (80)
WAIT	4-68 (270), 5-11 (103)
WRITE	4-77 (305), 5-16 (145)
X-CHAR	4-22 (80), 4-22 (80), 4-38 (151), 4-49 (196), 5-8 (74)
Y-CHAR	4-22 (80), 4-22 (80), 5-7 (68)
Z-CHAR	4-21 (79), 4-22 (80), 5-7 (68)
ZDIV	4-63 (246), 5-13 (116)
0-BIT	4-89 (347)
0-CHAR	3-22 (151), 3-23 (160), 4-50 (197), 4-89 (347), 4-89 (348)
1-BIT	4-89 (347)
1-CHAR	3-22 (151), 3-23 (160), 3-24 (166), 4-28 (105), 4-36 (142), 4-77 (303), 4-89 (348), 5-8 (72)
2-CHAR	3-22 (151), 3-24 (166), 4-89 (348), 5-8 (72)
3-CHAR	3-22 (151), 3-24 (166), 4-89 (348), 5-8 (72)
4-CHAR	3-22 (151), 3-24 (166), 4-89 (348)
5-CHAR	3-22 (151), 3-24 (166), 4-89 (348)
6-CHAR	3-22 (151), 3-24 (166), 4-89 (348), 5-8 (70)
7-CHAR	3-22 (151), 3-24 (166), 4-89 (348), 5-8 (70)
8-CHAR	3-22 (151), 3-24 (166), 4-89 (348), 5-8 (70)
9-CHAR	3-22 (151), 3-24 (166), 4-89 (348), 5-7 (68), 5-8 (74)

