COMPUTER SCIENCE LIBRARY

A FORMAL SYNTAX FOR PL/CS

Tim Teitelbaum

TR 76-281

June 1976

Department of Computer Science
Cornell University
Ithaca, New York 14853

A FORMAL SYNTAX FOR PL/CS

Tim Teitelbaum

This document contains a formal syntax for the PL/CS programming language. As is customary, the defining context-free grammar generates a somewhat larger language than PL/CS. That is, only those restrictions conveniently expressed by context-free productions are incorporated in the definition. However, all legal PL/CS programs are contained in the language defined.

With some exceptions, the formal syntax defines the language described in

> Conway, R., "PL/CS - A Highly-Disciplined Subset of PL/C", Dept. of Computer Science, Cornell University, TR 76-273.

The present report supersedes this earlier report as the document defining the syntax of the PL/CS subset.

1. META-SYNTACTIC CONVENTIONS

Values in PL/CS are restricted to one of four types: fixed, float, character or bit. A procedure which does not return a value (i.e. a procedure which is not a function) is said to have "void" type. The main procedure of the program is singled out from the other void procedures and has type "main".

The PL/CS grammar is parameterized according to various syntactic notions. Each notion is associated with a set of types as follows:

notion	main	void	fixed	float	character	bit
$\mathscr{T}$	x	x	x	x	x	x
$\mathscr{E}$		x	x	x	x	x
$\mathscr{P}$	x	x				
$\mathscr{F}$			x	x	x	x
$\mathscr{A}, \mathscr{A}', \mathscr{A}''$			x	x		
$\mathscr{B}$					x	x

A production containing syntactic notions is actually a scheme representing several context-free productions. One obtains a production from a scheme by replacing each notion uniformly by one of the types it represents. Thus, the scheme

$\mathscr{F}$-option $\rightarrow$ RETURNS ($\mathscr{F}$-attribute)

is an abbreviation for the productions

fixed-option → RETURNS(fixed-attribute)
float-option → RETURNS(float-attribute)
character-option → RETURNS(character-attribute)
bit-option → RETURNS(bit-attribute)

In a scheme with several different notions, for example $\mathscr{A}$ and $\mathscr{A}'$, distinct notions can be replaced by distinct types. All occurrences of the same notion must, however, be replaced by the same type.

The grammar employs the following meta-syntactic abbreviations:

$\langle\!\langle \alpha_1 \| \cdots \| \alpha_n \rangle\!\rangle$	α_1 or ... or α_n.
$[\![\alpha]\!]$	optional occurrence of α.
$\lessdot\, \alpha_1 \| \cdots \| \alpha_n \,\gtrdot$	a subset of between 0 and n items selected in any order from $\alpha_1, \ldots, \alpha_n$ without repetition.
$\{\alpha\}$	0 or more occurrences of α.
$\lfloor \alpha \rceil$	either a single occurrence of α or two or more occurrences of α's separated by commas.

In the abbreviations above, syntactic notions may appear within the α's. The replacement of notions by types is assumed to occur prior to any expansion of the meta-syntactic abbreviations. For example, $\lfloor \mathscr{F}$-identifier$\rceil$ denotes a list of identifiers <u>all with the same type</u>.

2. UNDEFINED LEXICAL AND SYNTACTIC OBJECTS

The lexical entities not further specified by this report are:

$\mathcal{T}$-entry-name
loop-name
label
$\mathcal{F}$-identifier
$\mathcal{F}$-array-identifier
$\mathcal{F}$-parameter
$\mathcal{F}$-array-parameter
$\mathcal{F}$-constant
*-plc-card
*-process-card
title-comment

In addition, it is assumed that a program may be laced with macro-definitions, comments and *-option-cards. This grammar defines the language after removal of these items and after the expansion of macro calls.

The syntactic unit $\mathcal{F}$-expression is not formally defined in this report. The $\mathcal{F}$-expressions of PL/CS are precisely the PL/C scalar expressions of type $\mathcal{F}$ where the operands are restricted to be constants, variables or calls. Thus, there are no array or structure expressions.

3. SYNTAX

3A. Values

$\mathscr{F}$-variable →
  $\mathscr{F}$-identifier |
  $\mathscr{F}$-parameter |
  $\mathscr{F}$-array-variable(⌊arithmetic-expression⌉)

character-variable →
  SUBSTR(character-variable,$\mathscr{A}$-expression⟦,$\mathscr{A}$-expression⟧

$\mathscr{F}$-array-variable →
  $\mathscr{F}$-array-identifier |
  $\mathscr{F}$-array-parameter

$\mathscr{F}$-variable-or-array →
  $\mathscr{F}$-variable |
  $\mathscr{F}$-array-variable

$\mathscr{F}$-array-value →
  $\mathscr{F}$-array-variable |
  ($\mathscr{F}$-array-value)

$\mathscr{F}$-expression-or-array →
  $\mathscr{F}$-expression |
  $\mathscr{F}$-array-value

parameter → $\mathscr{F}$-parameter
array-parameter → $\mathscr{F}$-array-parameter
expression-or-array → $\mathscr{F}$-expression-or-array
variable-or-array → $\mathscr{F}$-variable-or-array

$\mathscr{F}$-call → $\mathscr{F}$-entry-name ⟦(argument-list)⟧

argument-list → ⌊expression-or-array⌉

3B. Declarations

<program> →
*-plc-card
main-procedure-definition
{*-process-card
external-procedure-definition}

external-procedure-definition →
$\mathcal{E}$-procedure-definition

$\mathcal{T}$-procedure-definition →
title-comment
$\mathcal{T}$-entry-name:
PROCEDURE $\mathcal{T}$-parameter-list ⟦RECURSIVE⟧ $\mathcal{T}$-option;
{$\mathcal{T}$-parameter-declaration}
{$\mathcal{T}$-declaration}
$\mathcal{T}$-on-condition
{$\mathcal{T}$-statement}
$\mathcal{T}$-return-statement
END $\mathcal{T}$-entry-name;

$\mathcal{E}$-parameter-list → (⌊parameter|array-parameter⌉)
$\mathcal{T}$-parameter-list → ϵ

main-option → OPTIONS(MAIN)
void-option → ϵ
$\mathcal{F}$-option → RETURNS($\mathcal{F}$-attribute)

$\mathcal{P}$-on-condition → ON ENDFILE goto-statement
$\mathcal{T}$-on-condition → ϵ

$\mathscr{E}$-parameter-declaration →

DECLARE(⌊ $\mathscr{F}$-parameter|$\mathscr{F}$-array-parameter(parameter-bounds) ⌉)

$\mathscr{F}$-parameter-attribute

$\mathscr{E}$-protection-attribute;

parameter-bounds → ⌊*⌉

fixed-parameter-attribute → FIXED
float-parameter-attribute → FLOAT
character-parameter-attribute → CHARACTER(*) VARYING
bit-parameter-attribute → BIT(*)

$\mathscr{E}$-protection-attribute → READONLY

void-protection-attribute → ϵ

$\mathcal{T}$-declaration →
DECLARE (⌊ $\mathcal{F}$-identifier | $\mathcal{F}$-array-identifier (array-bounds) ⌉)
$\mathcal{F}$-attribute
⟨ INITIAL ($\mathcal{F}$-initial-value-list) |
$\mathcal{T}$-external-option |
$\mathcal{T}$-static-option ⟩;

array-bounds → ⌊arithmetic expression:arithmetic-expression⌉

fixed-attribute → FIXED
float-attribute → FLOAT
character-attribute → CHARACTER ($\mathcal{A}$-expression) VARYING
bit-attribute → BIT(1)

$\mathcal{P}$-external-option → EXTERNAL
$\mathcal{F}$-external-option → ϵ

$\mathcal{P}$-static-option → STATIC
$\mathcal{F}$-static-option → ϵ

$\mathcal{F}$-initial-value-list → ⌊$\mathcal{F}$-initial-item⌉

$\mathcal{F}$-initial-item →
*|
$\mathcal{F}$-constant-value |
(iteration-factor) ⦅ * | $\mathcal{F}$-constant-value | ($\mathcal{F}$-initial-value-list) ⦆

$\mathcal{A}$-constant-value → $\mathcal{A}'$-constant
$\mathcal{B}$-constant-value → $\mathcal{B}$-constant

iteration-factor → $\mathcal{A}$-expression

3C. Statements

$\mathcal{T}$-statement →
  $\mathcal{T}$-conditional-statement
  $\mathcal{T}$-unconditional-statement|

$\mathcal{T}$-unconditional statement →
  assignment-statement|
  $\mathcal{T}$-compound-statement|
  $\mathcal{T}$-do-while-statement|
  $\mathcal{T}$-do-index-statement|
  $\mathcal{T}$-return-statement |
  goto-statement |
  null-statement |
  assert-statement

$\mathcal{P}$-unconditional-statement →
  get-statement|
  put-statement|
  call-statement

```
𝒯-conditional-statement →
   IF (bit-expression) THEN 𝒯-statement|
   IF (bit-expression)
      THEN 𝒯-closed-statement
      ELSE 𝒯-statement

𝒯-closed-statement →
   𝒯-unconditional-statement|
   IF (bit-expression)
      THEN 𝒯-closed-statement
      ELSE 𝒯-closed-statement


assignment-statement →
   𝒜-variable = 𝒜'-expression;|
   character-variable = character-expression;|
   bit-variable = bit-rhs-expression;|
   𝒜-array-variable = 𝒜'-expression-or-array|
   character-array-variable = character-expression-or-array;|
   bit-array-variable = bit-rhs-expression-or-array;

bit-rhs-expression →
   bit-constant|
   bit-variable|
   (bit-expression)

bit-rhs-expression-or-array →
   bit-rhs-expression|
   bit-array-value
```

$\mathscr{T}$-compound-statement → DO; {$\mathscr{T}$-statement} END;

$\mathscr{T}$-do-while-statement →
loop-name: DO WHILE (bit-expression);
{$\mathscr{T}$-statement} END loop-name;

$\mathscr{T}$-do-index-statement →
loop-name:
DO fixed-identifier = $\mathscr{A}$-expression TO $\mathscr{A}'$-expression BY $\mathscr{A}''$-expression;
{$\mathscr{T}$-statement} END loop-name;

$\mathscr{P}$-return-statement → RETURN;
$\mathscr{A}$-return-statement → RETURN($\mathscr{A}'$-expression);
$\mathscr{B}$-return-statement → RETURN($\mathscr{B}$-expression);

goto-statement → GO TO label;

null-statement → ⟦label:⟧;

assert-statement →
ASSERT (bit-expression) {quantifier};

quantifier →
⟪FOR ALL|FOR SOME⟫ fixed-identifier =
$\mathscr{A}$-expression to $\mathscr{A}'$-expression BY $\mathscr{A}''$-expression

```
get-statement →
   GET LIST   ( L variable-or-array ⅂  );|
   GET EDIT   ( L variable-or-array ⅂  ) format;|
   GET DATA ⟦( L variable-or-array ⅂  )⟧;

put-statement →
   PUT skip-option LIST (argument-list);|
   PUT skip-option EDIT (argument-list);|
   PUT skip-option DATA (argument-list);|
   PUT skip-option;

skip-option → ⟦SKIP ⟦(𝒜-expression)⟧⟧

argument-list → Lexpression-or-array⅂

call-statement →
   CALL 𝒫-entry-name ⟦(argument-list)⟧;



format → ( L specification ⅂ )

specification → ⟦fixed-constant|(𝒜-expression)⟧ ⦅item|format⦆

item → A ⟦(𝒜-expression)⟧|
       B ⟦(𝒜-expression)⟧|
       COLUMN (𝒜-expression)|
       E (𝒜-expression,𝒜'-expression ⟦,𝒜"-expression⟧)|
       F (𝒜-expression)|
       LINE (𝒜-expression)|
       PAGE|
       SKIP ⟦(𝒜-expression)⟧|
       X(𝒜-expression)
```