§2. Static Structures of Programs

2.1 Syllables

We consider a program as an <expression> which is a figure consisting of a finite sequence of <basic symbol>'s. A program is divided into several number of subsequences called "syllables". A syllable is of the form <identifier>, <number>, <bits>, <string>, <code body> or <delimiter>. In a program two syllables other than <delimiter>'s must be separated by one or more <delimiter>'s. Under these conditions, the division of a program is unique. In the following, a sequence of syllables of the form α is called "to be α in the program" or simply "to be α", where α is a meta-variable. Each <identifier> in a program is used either as a <variable> or as a <label> or as a part of a <selector>.

2.2 Block-Structures and Declarations

For each <block> and <procedure notation>, we define its proper <variable>'s, proper <form>'s, proper <mark>'s, proper <label>'s and proper interior:

2.2.1 Let E be a <block> of the form

\[
\text{begin } D_1; \\
\ldots \\
D_n'; \\
L_1^1; \ldots ; L_1^k : E_1; \\
\ldots \\
L_k^1; \ldots ; L_k^k : E_k \text{ end}
\]

with <declaration>'s \( D_1, \ldots , D_n \), <label>'s \( L_1^1, \ldots , L_1^k \), \ldots , \( L_k^k \), <expression>'s \( E_1, \ldots , E_k \), where \( n \) is an integer (\( \geq 0 \)), \( k \) is an integer (\( \geq 1 \)), \( i_1, \ldots , i_k \) are integers (\( \geq 0 \)). Let \( i \) be an integer, and \( 1 \leq i \).
1) If $D_1$ is a <variable declaration> of the form
   \[ \text{"let } v_1 \text{ be } P_1 \text{"} \]
   with a <variable> $v_1$ and an <expression> $P_1$, then $v_1$ is a proper
   <variable> of $E$, and we say that "($<\text{declaration}>$ in the program)
   $D_1$ is $<\text{declaration}>$ for $v_1$".

2) If $D_1$ is a <form declaration> of the form
   \[ \text{"let } G_1 \text{ represent } P_1 \text{"} \]
   with a <form> $G_1$ and an <expression> $P_1$, then $G_1$ is a proper
   <form> of $E$, and we say that "$D_1$ is a $<\text{declaration}>$ for $G_1$".

3) If $D_1$ is a <mark declaration> of the form
   \[ \text{"let } P_1 \text{ operate } Z_1 \text{ for } P'_1 \text{"} \]
   with a <mark> $P_1$, a <left priority> $Z_1$ and a <right priority> $Z'_1$,
   then $P_1$ is a proper <mark> of $E$, and we say that "$D_1$ is a $<\text{declaration}>$ for $P_1$".

Furthermore,

3.1) If $Z_1$ is of the form
   \[ \text{"before } P_1', \ldots, P_m' \text{ left"} \],
   then we say that "$D_1$ is a reverse $<\text{declaration}>$ for the ordered
   pair $<P_j', P_1'>$ for $j=1,2,\ldots,m$.

3.2) If $Z_1$ is of the form
   \[ \text{"before all left"} \],
   then "$D_1$ is a reverse $<\text{declaration}>$ for the pair $<P', P_1'>$ for
   each <mark> $P'$.

3.3) If $Z_1$ is of the form
   \[ \text{"after } P_1'', \ldots, P_m'' \text{ right"} \],
   then "$D_1$ is a reverse $<\text{declaration}>$ for the pair $<P_1'', P_j''>$ for
   $j=1,2,\ldots,m$.

3.4) If $Z_1$ is of the form

Static Structures-2
"after all right",
then "D_i is a reverse <declaration> for the pair <P_i, P_i'>" for each <mark> P_i'.

4) L_1, ..., L_k are proper <label>'s of E.

2.2.2 Let E be a <procedure notation> of the form

"procedure (T_1, ..., T_n) T J"

with <typifier>'s T_1, ..., T_n, T and <procedure donor> J, where n is an integer (≥0).

If J is empty, then E has no proper <variable>'s.

If J is of the form

"by ((V_1, ..., V_n) F)"

with <variable>'s V_1, ..., V_n and an <expression> F, then V_1, ..., V_n are proper <variable> of E.

A <procedure notation> has neither proper <form>'s, nor proper <mark>'s, nor proper <label>'s.

2.2.3 Let E be a <block> or a <procedure notation> in a program, and let F be an <expression>, or a <label>, or a <declaration> in that program. If E is a <block> and F is a subsequence of E, then we say that "F is in the interior of E". If E is a <procedure notation> of the form

"procedure (T_1, ..., T_n) T J"

with <typifier>'s T_1, ..., T_n and T and a <procedure donor> J, and F is a subsequence of J, then we say that "F is in the interior of E". If F is in the interior of E and there is no <block> or <procedure notation> E', such that E' is in the interior of E, and F is in the interior of E', then we say that "F is in the proper interior of E".

Static Structures-3
2.2.4 Let A be a <variable> (or a <form> or a <mark>), and E be a <block> or a <procedure notation> (in a program). And let \( P \) be the minimum <block> or <procedure notation> (in the program), which contains E in its interior and A is its proper <variable> (or <form> or <mark>), and D be a <declaration> (in the program) which is a <declaration> of A and is in the proper interior of \( E \). Then we say that "D is a <declaration> of A in the interior of E", or "A in the proper interior of E is declared on \( P \)". Let L be a <label>, and E be a <block> or a <procedure notation> (in a program). And let \( P \) be the minimum <block> or <procedure notation> (in the program), which contains E in its interior and L is its proper <label>. Then we say that "L in the proper interior of E is declared on \( P \)".

Let \( P, P' \) be <mark>'s, and E be a <block> or a <procedure notation> (in a program). And let \( P \) be the minimum <block> or <procedure notation> (in the program), on which \( P \) or \( P' \) in the proper interior of E is declared. Furthermore D be a <declaration> of either \( P \) or \( P' \) and is in the proper interior of \( P \). Then we say that "D is a <declaration> of the pair \( <P,P'> \) in the proper interior of E". If there is a reverse <declaration> of \( <P,P'> \), which is a <declaration> of \( <P,P'> \) in the proper interior of E, then we say that "\( <P,P'> \) is reverse in the interior of E", and in other cases "\( <P,P'> \) is natural in the proper interior of E".
2.3 Parsing of Expressions

The parsing of an <expression> is syntactically unique except for constructions of <form call>'s. To obtain the complete uniqueness, we restrict <mark declaration>'s and the construction of <form call>'s as follows:

(R1) In the proper interior of each <block>, there must be at most one <declaration> for each <mark>.

(R2) For each <mark> P in a program, there must be a <declaration> (in the program) by which P is declared or P is declared by a standard <declaration>. Therefore each <mark> P in a program has just one (standard or not standard) <declaration> D by which P is declared.

Let D be of the form

"let P operate ZZ"

where Z is a <left priority> and Z' is a <right priority>. If both Z and Z' are not empty, then P is called "\(\text{z} \text{z}\) independent". If Z is not empty and Z' is empty, then P is called "\(\text{z} \text{z}\) initial". If Z is empty and Z' is not empty, then P is called "\(\text{z} \text{z}\) terminal". If both Z and Z' are empty, then P is called "\(\text{z} \text{z}\) connecting".

Let an <expression> in a program be a <form call> of the form

"E_1 E_2 E_3 \ldots E_{n-1} E_n P E"

where \(n\) is an integer \((\geq 1)\), \(E_i\) is a <mark> for \(i = 1, 2, \ldots, n\), and \(E_i\) is empty or an <expression> for \(i = 0, 1, \ldots, n\).

(R3.1) If \(n = 1\) then \(P_1\) must be independent.

(R3.2) If \(n > 1\) then \(P_1\) must be initial, and \(P_n\) must be terminal, and \(P_i\) must be connecting for \(i = 2, 3, \ldots, n-1\).

(R4.1) If \(E_o\) is a <form call> of the form

"E_0 E_1 \ldots E_{n-1} E_n P E_1 E_2 \ldots E_n E_1 E_2 \ldots E_n P E_1 E_2 \ldots E_n"

where \(n'\) is an integer \((\geq 1)\),

\(P_i\) is a <mark> for \(i = 1, 2, \ldots, n'\),

\(E_i\) is empty or an <expression> for \(i = 0, 1, \ldots, n'\),

then <\(P_n\)'), \(P_1\) must be natural.

Static Structures-5
(R4.2) If $E_n$ is a $\langle$form call$\rangle$ of the form

$$E_0^"p_1"E_1^"p_2"E_2"\ldots E_{n-1}^"p_n"E_n"$$

where $n'$ is an integer ($\geq 1$),

$P_i$' is a $\langle$mark$\rangle$ for $i = 1, 2, \ldots, n'$,

$E_i$' is empty or an $\langle$expressin$\rangle$ for $i = 0, 1, \ldots, n'$,

then $\langle P_n, P_1' \rangle$ must be reverse.

Those restrictions (R3.1), (R3.2), (R4.1) and (R4.2) are also applied for every $\langle$typifier$\rangle$'s and $\langle$form$\rangle$'s as $\langle$expression$\rangle$'s.

2.4 Direct Constituents of Expressions

Let $E$ and $E'$ be $\langle$expression$\rangle$'s.

$E$ is said to embrace $E'$ if and only if $E$ is of the form

$"A\varepsilon'B"$

where $A$ and $B$ are figures and at least one of them is non-empty. $E'$ is called a direct constituent of $E$ if and only if the following three conditions are satisfied.

1) $E$ embraces $E'$;

2) $E$ embraces no $\langle$expression$\rangle$ which embraces $E'$;

3) $E'$ is used neither as a $\langle$typifier$\rangle$ nor as a $\langle$primary typifier$\rangle$ in the construction of $E$. 

Static Structures-6
2.5 Types

2.5.1 Types are defined recursively as follows:

1) **effect** is a type.
2) **real** is a type.
3) **bits** is a type.
4) **string** is a type.
5) **reference** is a type.

6) Let \( T \) be a type, then **array** \( T \) is a type, and called array style.

7) Let \( n \) be an integer \( (\geq 0) \); \( S_i \) be a **selector** different from each other, for \( i = 1, 2, \ldots, n \), and \( T_i \) be a type for \( i = 1, 2, \ldots, n \); then **structure** \( (S_1, T_1, \ldots, S_n, T_n) \) is a type, and called structure style.

8) Let \( n \) be an integer \( (\geq 0) \);

\( T_n \) be a type for \( i = 1, 2, \ldots, n \);

and \( T \) be a type; then

**procedure** (\( T_1, \ldots, T_n \)) \( T \) is a type, and called procedure style.

We shall use the following notations:

- \( T_{array} \) : The set \( \{ T \mid T \text{ is a type of array style} \} \).

- \( T_{structure} \) : The set \( \{ T \mid T \text{ is a type of structure style} \} \).

- \( T_{procedure} \) : The set \( \{ T \mid T \text{ is a type of procedure style} \} \).

Let \( T \) stand for an arbitrary type, then \( T \) is of the form **typifier**.

We shall denote this **typifier** simply by \( T \). In a legal program, each **expression** and each **form** has its type. And some semantical notion (quantity, value and mode) has its type.

Let \( A \) be an **expression**, **form**, quantity, value or mode, then we shall denote its **type** by \( t(A) \).

Types-1
2.5.2 To define the type's of expression's, we introduce some restrictions:

(R5.1) In the proper interior of each block in a program, there must be at most one declaration for each variable.

(R5.2) For each procedure donor in a program of the form

"by \((V_1, ..., V_n)^E\)"

with variable's \(V_1, ..., V_n\), and an expression \(E, V_1, ..., V_n\) must be different from each other.

(R6) Each variable in a program, must be declared by a declaration in the program, or by a standard declaration.

Further restrictions on types are introduced recursively with the definition of the types of expression's.

The type of an expression \(E\) in a program is abstracted by the form of \(E\) and types of expression's contained in \(E\). Those types of sub(expression)'s are abstracted from left to right in the contextual order.

(R7) By this process, the type of each expression must be able to be defined.

1) In the beginning of the type abstraction of a block (in a program) \(E\), each variable declaration and form declaration are processed from left to right.

1.1) Let a variable declaration be of the form

"let \(V\) be \(F\)"

with a variable \(V\) and an expression \(F\).

Then the type of \(F\) (\(t(F)\)) is abstracted, and \(t(F)\) is represent the type of a variable (in the program) of the form \(V\) and declared on \(E\).

1.2) Let a form declaration \(D\) be of the form

"let \(G\) represent \(F\)"

with a form \(G\) and an expression \(F\), and let \(G\) be of the form

Types-2
where \( n \) is an integer \((\geq 1)\),

- \( P_i \) is a \( \langle \text{mark} \rangle \) for \( i = 1, 2, \ldots, n \),
- \( E_i \) is empty or an \( \langle \text{expression} \rangle \) for \( i = 0, 1, \ldots, n \).

Let \( T_i \) stand for \( t(E_i) \) if \( E_i \) is an \( \langle \text{expression} \rangle \),
- empty if \( E_i \) is empty, for \( i = 0, 1, \ldots, n \).

Then the figure

\[
( T_0 P_1 T_1 P_2 T_2 \ldots T_{n-1} P T_n )
\]

is called the operator form of \( G \), and we say that "\( D \) is a \( \langle \text{declaration} \rangle \) for this operator form". And the figure, which is made from the operator form of \( G \) eliminating all \( \langle \text{mark} \rangle \)'s and insert a comma "," between each succession of two types, is called the argument-types of \( G \).

(R8) In this case, \( t(F) \) must be procedure style, and if \( t(F) \) is of the form

\[
\text{procedure } (T_1, \ldots, T_n)T
\]

with types \( T_1, \ldots, T_n, T \), then the argument-types of \( G \) must be \( (T_1, \ldots, T_n) \).

And we say that in this \( \langle \text{declaration} \rangle \) the result type of \( G \) is \( T \).

2) In the case of a \( \langle \text{procedure notation} \rangle \) (in a program) of the form

\[
( T_1, \ldots, T_n ) T \text{ by } ((V_1, \ldots, V_m) E)
\]

with \( \langle \text{expression} \rangle \)'s \( T_1 ; \ldots, T_n, T, E \) and \( \langle \text{variable} \rangle \)'s \( V_1, \ldots, V_m \).

(R9) In this case, \( m \) must be \( n \), and \( t(E) \) must be \( t(T) \).

\( t(T_i) \) represents the type of a \( \langle \text{variable} \rangle \) (in the program) of the form \( V_i \) and declared on \( E \), for \( i = 1, 2, \ldots, n \).

2.5.3 Let \( E \) be a \( \langle \text{block} \rangle \) or \( \langle \text{procedure notation} \rangle \) (in a program) and \( \varnothing \) be an operator form.

If \( F \) is the minimum \( \langle \text{block} \rangle \) (in a program) which contains \( E \) (or is \( E \)), and
is declared by a $\langle$declaration$\rangle$ $D$ in its proper interior, then we say that "$\bar{\omega}$ in the proper interior of $E$ is declared on $F$" or "$\bar{\omega}$ in the proper interior of $E$ is declared by $D$".

(R10) In the proper interior of each $\langle$block$\rangle$ (in a program) there must be at most one $\langle$declaration$\rangle$ for each operator form.

2.5.4 1) Let $E$ be a $\langle$variable$\rangle$ $V$ in a program. The type of $V$ ($t(V)$) is defined as above.

2) Let $E$ be a $\langle$go to statement$\rangle$ or $\langle$dummy statement$\rangle$. Then $t(E)$ is effect.

3) Let $E$ be a $\langle$code call$\rangle$ of the form

$$\langle\text{code} \ (S_1 \ E_1, \ldots, S_n \ E_n) \rangle_T \text{ by (A)}$$

with $\langle$selector$\rangle$'s $S_1$, $\ldots$, $S_n$, $\langle$expression$\rangle$'s $E_1$, $\ldots$, $E_n$, $T$, and $\langle$code body$\rangle$.

Then, $t(E)$ is $t(T)$.

(R11) In this case, $S_1$, $\ldots$, $S_n$ must be different from each other.

4) Let $E$ be a $\langle$closed expression$\rangle$ of the form

$$\langle F \rangle$$

with an $\langle$expression$\rangle$ $F$. Then $t(E)$ is $t(F)$.

5) Let $E$ be a $\langle$block$\rangle$ of the form

$$\langle\text{begin} \ D_1 : \ldots ; D_n : \$$

$$L_1 : \ldots ; L_{i_1} : E_1 ; \ldots ; L_{i_k} : \ldots ; L_{i_k} : E_k \ \text{end} \rangle$$

with $\langle$declaration$\rangle$'s $D_1$, $\ldots$, $D_n$,

$\langle$label$\rangle$'s $L_1$, $\ldots$, $L_{i_1}$, $\ldots$, $L_{i_k}$, $\ldots$, $L_{i_k}$,

$\langle$expression$\rangle$'s $E_1$, $\ldots$, $E_k$.

Then $t(E)$ is $t(E_k)$.

6) Let $E$ be an $\langle$array element$\rangle$ of the form

$$\langle F \ E' \rangle$$

with $\langle$expression$\rangle$'s $F$ and $E'$.
(R12) In this case, \( t(F) \) must be array style, and \( t(E') \) must be real.

Let \( t(F) \) be of the form

\[
\text{array } T
\]

with a type \( T \). Then \( t(E) \) is \( T \).

7) Let \( E \) be a \(<\text{structure element}>\) of the form

\[
" F[ S ] "
\]

with an \(<\text{expression}>F\) and a \(<\text{selector}>S\).

(R13) In this case, \( t(F) \) must be structure style, and when \( t(F) \) is of the form

\[
\text{structure } (S_1 T_1, \ldots, S_n T_n)
\]

with \(<\text{selector}>'s S_1, \ldots, S_n \) and types \( T_1, \ldots, T_n \), \( n \) must be \( \geq 1 \),

and \( S \) must be one of \( S_1, \ldots, S_n \).

If \( S \) is \( S_i \) \((1 \leq i \leq n)\), then \( t(E) \) is \( T_i \).

8) Let \( E \) be a \(<\text{procedure call}>\) of the form

\[
" F(E_1, \ldots, E_n) "
\]

with \(<\text{expression}>'s F, E_1, \ldots, E_n \).

(R14) In this case, \( t(F) \) must be procedure style, and when \( t(F) \) is of the form

\[
\text{procedure } (T_1, \ldots, T_m) T
\]

with types \( T_1, \ldots, T_m, T \), and \( m \) must be \( n \), and \( t(E_i) \) must be \( T_i \) for \( i = 1, 2, \ldots, m \).

Then \( t(E) \) is \( T \).

9) Let \( E \) be a \(<\text{form call}>\) of the form

\[
" E P E P E \ldots E P E "
\]

where \( n \) is an integer \((\mathbb{Z} \geq 1)\),

\( P_i \) is a \(<\text{mark}>\) for \( i = 1, 2, \ldots, n \),

\( E_i \) is empty or an \(<\text{expression}>\) for \( i = 0, 1, \ldots, n \).

and be in the proper interior of a \(<\text{block}>\) or \(<\text{procedure notation}>E'\).

Types=5
Let \( T_i \) stand for \( t(E_i) \) if \( E_i \) is an expression, empty if \( E_i \) is empty, for \( i = 1, 2, \ldots, n \), and let \( \overline{\varnothing} \) stand for the operator form

\[
(T_0 \overline{\varnothing} T_1 \overline{\varnothing} T_2 \cdots T_{n-1} \overline{\varnothing} T_n)
\]

(R15) In this case, \( \overline{\varnothing} \) in the proper interior of \( E' \) must be declared by a declaration in the program or by a standard declaration.

Let \( D \) be the declaration for \( \overline{\varnothing} \) in the proper interior of \( E' \), of the form

"let \( G \) represent \( F \)"

with a form \( G \) and an expression \( F \), and let \( T \) be the result type of \( G \).

Then \( t(E) \) is \( T \).

10) The type of a effect notation is effect.
11) The type of a real notation is real.

(R16) In a real modifier of the form

"[ \( E_1 : E_2 \) : \( E_3 \) ]" or "[ precision \( E_4 \) ]",

if \( E_i \) is an expression, then \( t(E_i) \) must be real for \( i = 1, 2, 3, 4 \).

12) The type of a bits notation is bits.

(R17) In a bits modifier of the form

"[ exact \( E_1 \) ]" or "[ varying \( E_1 \) ]",

if \( E_1 \) is an expression then \( t(E_1) \) must be real.

13) The type of a string notation is string.

(R18) In a string modifier of the form

"[ exact \( E_1 \) ]" or "[ varying \( E_1 \) ]",

if \( E_1 \) is an expression then \( t(E_1) \) must be real.

14) The type of a reference notation is reference.
15) Let \( E \) be a array notation of the form

"array HJ"

Types-6
with an \textit{array modifier} H and an \textit{expression} J.

Then \( t(E) \) is \textit{array} \( t(J) \).

(R19) In a \langle array modifier \rangle of the form

\[
\begin{array}{c}
\begin{array}{c}
E_1 : E_2
\end{array}
\end{array}
\]  

with \langle expression \rangle's, \( t(E_1) \) and \( t(E_2) \) must be real.

16) Let \( E \) be a \langle array notation \rangle of the form

\[
\begin{array}{c}
\begin{array}{c}
\text{array} ( E_1, ..., E_n )
\end{array}
\end{array}
\]  

with \langle expression \rangle's \( E_1, ..., E_n \).

(R20) In this case, \( t(E_1), t(E_2), ..., t(E_n) \) must be equal.

Then \( t(E) \) is \textit{array} \( t(E_1) \).

17) Let \( E \) be a \langle structure notation \rangle of the form

\[
\begin{array}{c}
\begin{array}{c}
\text{structure} ( S_1E_1, ..., S_mE_n )
\end{array}
\end{array}
\]  

with \langle selector \rangle's \( S_1, ..., S_m \), and \langle expression \rangle's \( E_1, ..., E_n \).

(R21) In this case, \( S_1, ..., S_m \) must be different from each other.

Then \( t(E) \) is \textit{structure} \( ( S_1 t(E_1), ..., S_m t(E_n) ) \).

18) Let \( E \) be a \langle procedure notation \rangle of the form

\[
\begin{array}{c}
\begin{array}{c}
\text{procedure} ( T_1, ..., T_n ) T_J
\end{array}
\end{array}
\]  

with \langle typifier \rangle's \( T_1, ..., T_n, T \), and \langle procedure donor \rangle \( J \).

Then \( t(E) \) is \textit{procedure} \( ( t(T_1), ..., t(T_n) ) t(T) \).

2.6 Legal Programs

A program is called legal, if it suffices the restrictions (R1) - (R21) and the following (R22) and (R23).

(R22) For each \langle block \rangle in the program of the form

\[
\begin{array}{c}
\begin{array}{c}
\text{begin} D_1 : ... : D_n ;
L_1^1 : ... : L_1^1 : E_1 ; ... ; L_1^k : ... : L_k^k \ E_k \text{ end}
\end{array}
\end{array}
\]  

with \langle declaration \rangle's \( D_1, ..., D_n \);

\langle label \rangle's \( L_1^1, ..., L_1^1, ..., L_1^k, ..., L_k^k \);

and \langle expression \rangle's \( E_1, ..., E_k \);

\( L_1^1, ..., L_1^1, ..., L_1^k, ..., L_k^k \) must be different from each other.

Types-7
(R23) For each block or procedure notation E in the program, and for each label L in the proper interior of E, there must be a block or procedure notation in the program, which contains E and on which L declared.