6. Standard declarations

This section describes the ordered set SD of standard declarations supporting the use of the language. The description takes the form of orderly enumeration of a family of disjoint subsets in SD. The description of each member of this family is headed by a reference numbering of the form (SDi,j,k). The ordering in SD is such that one element under (SDi1,j1,k1) precedes another element under (SDi2,j2,k2) if (i1,j1,k1) precedes (i2,j2,k2) in the lexical order.

Each member description begins either with a '{' (not a '[') or with an '>'.

(a) If a member description begins with a '{', then the matching '}' terminates that description, and the member has a single element, which is the enclosed text.

(b) If a member description begins with an '>', then the part enclosed by the first following '{' and the matching '}', with which the member description is terminated, gives an element of the member when asterisked symbols (T* for example) in that part are consistently replaced by certain objects as prescribed after the leading '>' in terms of common mathematical notations (T*e T for example).

† In the present document, instead of the whole family covering SD, only some illustrating subfamily will be presented. Other members, including those for input/output operations, will be found elsewhere.
6.1. Basic operations

6.1.1. Copying and enproceduing operations

(SD1.1.1) \{let copy operate before all left after copy right\}

(SD1.1.2) \{let enproc operate before all left after all right\}

(SD1.1.3) \(T^*eT\) \{let copy() represent procedure\(\Delta^*T^*\) by

((original) code \(c:original, d:T^*\) \(\Delta^*T^*\) by

\((\text{core}(Q) \text{ let } w(Q)={\langle c:,Q_1>,<d:,Q_2>};\)

\(m(Q_2)+m(Q_1); w(Q_2)+w(Q_1); \Rightarrow Q_2 \text{ end of core})\)})

(SD1.1.4) \(T^*eT\) \{let enproc() represent

procedure\(\Delta^*\)procedure()\(\Delta^*\) by

((body) procedure()\(\Delta^*\) by

((body body))\)

6.1.2. Simple assignment operations

(SD1.2.1) \{let +,- operate before +,- left after all right\}

(SD1.2.2) \(T\{\text{real, bits, reference}\}^+T\) \(\Delta^*\)\(\text{procedure, }\Delta^*\)\(\text{effect}\) by

\{let ()\(\Delta^*()\) represent procedure\(\Delta^*\)\(\text{procedure, }\Delta^*\)\(\text{effect}\) by

((destination, source)

\((\text{core}(Q) \text{ let } w(Q)={\langle d:,Q_1>,<s:,Q_2>};\)

\(w(Q_1)+ p(m(Q_1),w(Q_2)); \Rightarrow Q_0 \text{ end of core})\))

6.1.3. Simple comparison operations

(SD1.3.1) \{let =,# operate before left after +,- right\}

(SD1.3.2) \{let <,,> operate before =,# left

after +,- right\}

(SD1.3.3) \(T^*eT, \Delta^*e=,#; T^*\text{real}, \Delta^*e<,,>\}

\{let ()\(\Delta^*()\) represent procedure\(\Delta^*\)\(\text{procedure, }\Delta^*\)\(\text{bits}\) by

((left,right) code \(l:\text{copy left, r:right, t:1 \) bits by

\((\text{core}(Q) \text{ let } w(Q)={\langle l:,Q_1>,<r:,Q_2>,<t:,Q_3>};\)

\(\text{if } w(Q_1)\Delta^*w(Q_2) \text{ then } \Rightarrow Q_3, \text{ else } \Rightarrow \text{next};\)

\(w(Q_3)+ 0; \Rightarrow Q_3 \text{ end of core})\))

Standard Declarations -2
6.1.4. Conditional operations

(SD1.4.1) \{let if operate before all left\}

(SD1.4.2) \{let then operate\}

(SD1.4.3) \{let else, do operate after all right\}

(SD1.4.4) \{let if() then() else() represent procedure(bits, T*, T*) T* by
((condition, then, else)

\[\text{code } (c:\text{condition}, t:\text{enproc then, e:\text{enproc else}) enproc T* by}

\text{(core(Q) let w(Q)=\{<c:Q_1>, <t:Q_2>, <e:Q_3>\}; \}

\text{if the bit string w(Q_1) contains at least one 1 then \Rightarrow Q_2, else \Rightarrow Q_3, end of core} () )}\}

(SD1.4.5) \{let if() do() represent procedure(bits, effect) effect
by((condition, statement) if condition then statement
else dummy))\}

6.1.5. Basic arithmetic operations

(SD1.5.1) \{let +,- operate before =,\#,<,\le,\ge,> left
after +,-:= right\}

(SD1.5.2) \{let \times,/, operate before =,\#,<,\le,\ge,>,+,- left
after +,-:= right\}

(SD1.5.3) \{let ()\#() represent procedure(real,real)real by ((left,right)

\[\text{code } (a:\text{copy left, b:right) real by}

\text{(core(Q) let w(Q)=\{<a:Q_1>, <b:Q_2>\};}

\text{if the arithmetic operation meant by w(Q_1)\#w(Q_2)
can be performed then \Rightarrow next, else \Rightarrow L0;}

\text{let W_2 be the resulting value of that operation
(possibly with some implementation dependent deviation);}

\text{w(Q_1)\# p(m(Q_1),W_2)); \Rightarrow Q_1 end of core})}\}

(SD1.5.4) \{let -() represent procedure(real)real by
((right) begin (right)b; (copy b)a; a:=0; a-b end)\}

(SD1.5.5) \{let () represent procedure(real)real by
((right) begin (right)b; (copy b)a; a:=1; a/b end)\}

Standard Declarations -3
6.1.6. Some enquiry operations
(SD1.6.1) \{ let mode, length, bd operate before all left
after right \}

(SD1.6.2) \{ let mode() represent procedure(real) structure
(fix:bits, min:real, step:real, max:real) by ((real)
begin (real)x; code (fix:1, min:copy x, step:real,
max:copy x) (mode real) by
(core(Q) let w(Q)={<fix:Q1>,<min:Q2>,
<step:Q3>,<max:Q4>};
let M stand for m(Q2);
w(Q2)= the minimum value in \( W_M \);
w(Q1)= the maximum value in \( W_M \);
if M=real [precision R] with a real number R,
then \( \rightarrow \text{next} \), else \( \rightarrow \text{Kfixed} \);
w(Q3)= p(m(Q3), R); w(Q1)= 0; \( \rightarrow \text{Q} \);
Kfixed: let M=real [R1:R2:R3] with real numbers R1,
R2, R3;
w(Q3)= p(m(Q3), R2); \( \rightarrow \text{Q} \) end of core end)\}

(SD1.6.3) \{ let mode() represent
procedure(T*) structure (exact:bits, length:real) by ((some)
code (s:some, t:(exact:1, length: ) (mode T*) by
(core(Q) let w(Q)={<s:Q1>,<t:Q2>}; \( \rightarrow \text{integer} \)
let w(Q2)={<exact:Q3>,<length:Q4>};
let m(Q1)=T* [I Y I] , where I is an integer and Y
is either exact or varying;
w(Q4)= p(m(Q4), I);
if Y=exact then \( \rightarrow Q2 \), else \( \rightarrow \text{next} \);
w(Q3)= 0; \( \rightarrow \text{Q2} \) end of core)\}

(SD1.6.4) \{ let mode() represent procedure(array T*)
structure (lbd:real, ubd:real) by ((array)
code (a:array, t:(lbd:1, ubd: ) (mode array T*) by
(core(Q) let w(Q)={<a:Q1>,<t:Q2>}; \( \rightarrow \text{integer} \)
let w(Q2)={<lbd:Q3>,<ubd:Q4>};
let m(Q1)=array [I1:I2] T* with integers I1, I2;
w(Q3)= p(m(Q3), I1); w(Q4)= p(m(Q4), I2);
\( \rightarrow \text{Q2} \) end of core)\}

Standard Declarations -4
6.1.6 continued

(SD1.6.5) \( \texttt{fT}*\texttt{e\{bits,string\}} \{ \texttt{let length()} \texttt{represent} \) \\
procedure(T*)\texttt{real by ((string) code \{s:string,t: \} real by} \\
(\texttt{core}(Q) let \( w(Q) \equiv \langle s_1, q_1, \langle t, q_2 \rangle \rangle \); \\
let I be the integral length of \( w(Q_1) \); \\
w(Q_2) \leftarrow p(m(Q_2), I); \) \( \Rightarrow Q_2 \) \texttt{end of core)} \)

(SD1.6.6) \( \texttt{fT}*\texttt{eT} \{ \texttt{let bd()} \texttt{represent} \) procedure(array T*)real \\
by ((array) (mode array)[1bd:] )\}

(SD1.6.7) \( \texttt{fT}*\texttt{eT} \{ \texttt{let ()bd represent} \) procedure(array T*)real \\
by ((array) (mode array)[ubd:] )\}
6.2. Extended operations

6.2.1. Repetitive operations

(SD2.1.1) \{\textbf{let} \textit{while,until operate before left after all right}\}

(SD2.1.2) \{\textbf{let} \textit{succ,step operate before left after +,-,=,\neq,\leq,\geq,+,\text{-,}/ right}\}

(SD2.1.3) \{\textbf{T*}\textit{eT} \{\textbf{let} ()\textit{succ()} \textbf{represent procedure(T*,T*)}\}
\textit{structure (init:enproc T*, succ:enproc T*) by}
\((a,s) \{(\textit{init:enproc a, succ:enproc s})\}}\}

(SD2.1.4) \{\textbf{T*}\textit{eT} \{\textbf{let} ()\textit{while()} \textbf{represent procedure(T*\textit{succT*},bits)}\}
\textit{structure (init:enprocT*, succ:enprocT*, while:enproc bits) by}
\((r,t) \begin{align*}
&(\textit{r:enproc r1, succ:enproc succ, while:enproc while}) \\
&\textit{end})\}

(SD2.1.5) \{\textbf{let} ()\textit{step()} \textbf{represent procedure(real,real)}\}
\textit{structure (init:enproc real, step:enproc real) by}
\((a,b) \{(\textit{init:enproc a, step:enproc b})\}}\}

(SD2.1.6) \{\textbf{let} step() \textbf{represent procedure(real)(real step real)}\}
\textit{by ((b) 1 step b )}\}

(SD2.1.7) \{\textbf{let} ()\textit{until()} \textbf{represent procedure(real step real,real)}\}
\textit{structure (init:enproc real, step:enproc real,}
\textit{until:enproc real) by}
\begin{align*}
&((\textit{progression,limit}) \\
&\begin{align*}
&\textit{begin} \begin{cases}
&\textit{progression} ab; \{(\textit{init:enproc ab[init:]}), \\
&\textit{step:enproc ab[step:}, until:enproc limit\textit{end})
\end{cases}
\end{align*}
\end{align*}\}

(SD2.1.8) \{\textbf{let} ()\textit{until()} \textbf{represent procedure(real,real)}\}
\textit{step real until real by ((a,c) a step 1 until c )}\}

(SD2.1.9) \{\textbf{let} until() \textbf{represent procedure (real)(real until real)}\}
\textit{by ((c) 1 until c )}\}
6.2.1 continued

(SD2.1.10) \{let from operate before all left\}

(SD2.1.11) /\*eT \{let ()from()do\ represent\}
procedure(T\*, T*\text{succ}\*\text{while} bits, effect) effect by
\((\text{cvar}, \text{domain}, \text{statement})\)
  begin (cvar)cv; (domain)dom; cv+ dom[\text{init}:]();
  next: if dom[\text{while}:]() do
    begin statement; cv+ dom[\text{succ}:](); \text{go to next end}
  end
\}

(SD2.1.12) \{let ()from()do() represent\}
procedure(real, until real, effect) effect by
\((\text{cvar}, \text{domain}, \text{statement})\)
  begin (cvar)cv; (copy cv)vl; (copy domain)dom;
  (copy dom[\text{init}:]())al; (copy dom[\text{step}:]())bl;
  (copy dom[\text{until}:]())cl;
  vl from al succ vl+bl while if bl>0 then vl≤cl
  else if bl<0 then vl≥cl else vl≠cl
  do begin cv:=vl; statement end
end\}
6.2.2. Assign-and-hold operations

(SD2.2.1) \{let the operate before all left after right\}

(SD2.2.2) $\forall T^*eT \{let \, the() \, represent$

\begin{align*}
\text{procedure}(T^*) \ & \text{procedure}(\text{effect}) \ T^* \ \text{by} \\
\text{((expression))} \ & \text{procedure}(\text{effect}) \ T^* \ \text{by} \ (\text{dummy expression})
\end{align*}

(SD2.2.3) $\forall T^*eT, \Delta^*e\{+,:=\}\{let \ ()\Delta^*() \, represent$

\begin{align*}
\text{procedure}(\text{the} \ T^*, T^*) \ & \ T^* \ \text{by} \ (\text{the destination, source}) \\
\text{begin} \ & \ (\text{the destination}(\text{dummy}))\text{destination}; \\
\ & \ \text{destination}\Delta^*\text{source}; \ \text{destination} \\
\text{end})
\end{align*}
6.2.3. Reference handling operations

(SD2.3.1) \{\texttt{let } ref, \texttt{has_type}, \texttt{as_type} \texttt{operate before all left after right }\}

(SD2.3.2) \texttt{I^*eT} \{\texttt{let } ref() \texttt{represent procedure(T*)reference} \texttt{by } ((\texttt{referent}) \texttt{code } (q:\texttt{referent}, r:\texttt{reference}) \texttt{reference} \texttt{by } (\texttt{core}(Q) \texttt{let } w(Q)=\{<q_1>,<r_; Q_2>; \texttt{w}(Q_2)+ (Q_1); \rightarrow Q_2 \texttt{end_of_core})\}

(SD2.3.3) \texttt{I^*eT} \{\texttt{let } ()\texttt{has_type() represent procedure(referential, T*)bits by } ((\texttt{ref, type}) \texttt{code } (r:\texttt{ref}, t:\texttt{type}, b:0) \texttt{bits by } \texttt{core}(Q) \texttt{let } w(Q)=\{<r_1>,<t_1>,<b_; Q_3>; \texttt{if } w(Q_1)= \texttt{empty, then } Q_3, \texttt{else } \rightarrow Q_3; \texttt{let } w(Q_1)=(Q_4); \texttt{if } t(Q_2)=t(Q_4) \texttt{then } \rightarrow Q_3, \texttt{else } Q_3; \texttt{w}(Q_3)= 1; \rightarrow Q_3 \texttt{end_of_core})\}

(SD2.3.4) \texttt{I^*eT}, \texttt{T^#reference} \{\texttt{let } ()\texttt{as_type() represent procedure(referential, T*) T* by } ((\texttt{ref, type}) \texttt{code } (r:\texttt{ref}, t:\texttt{type}) \texttt{T* by } \texttt{core}(Q) \texttt{let } w(Q)=\{<r_1>,<t_1>,<t_2>; \texttt{if } w(Q_1)= \texttt{empty, then } Q_3, \texttt{else } \rightarrow Q_2; \texttt{let } w(Q_1)=(Q_4); \texttt{if } t(Q_2)=t(Q_4) \texttt{then } Q_3, \texttt{else } Q_2 \texttt{end_of_core})\}

Standard Declarations -9