数理解析研究所講究録
66巻 1969年 43-63

§4. Operators

4.1 Core Language

To describe the action of operations, the elaboration (e), the generation (g), the refinement (r), etc. we shall use the "core language". This language consists of usual mathematical notations, simple English sentences and ALGOL-like structures. Moreover we shall use several conventions as follows:

Let K be a label in the core language.

1) $\to$ K : "go to K".

2) $\to$ next : "go to next statement".

Let f be an operator, and let A be an operand.

3) f(A) : "operate f on A".

4) f(A) => A' : "let A' stand for the result of the operation f on A".

5) => A' : "the operation is now completed with the result A'".

6) A $\leftarrow$A' : "let A stand for A'".

7) let A = A' : "let A be of the form A'".

When f is the elaboration "e", and E is an <expression>,

8) e(E) if Q $\to$ K, L $\to$ K' : "e(E); let the result be A'; if A'$\in \mathbb{Q}$, then let Q stand for A' and go to K, else if A'$\in \mathbb{L}$, then let L stand for A' and go to K'".

4.2 Generations

4.2.1 g($\mathbb{Q}$) is

<u>core</u>

let Q stand for a new abstract element $\notin \mathbb{Q}$;

$\mathbb{Q} \leftarrow \mathbb{Q} \cup \{Q\}$;

=> Q

<u>end of core</u>

4.2.2 $g(\mathbb{V})$ is

core

let V stand for a new identifier $\notin \mathbb{V} \cup \mathbb{L}$;

$\mathbb{V} \leftarrow \mathbb{V} \cup \{V\}$;

=> V

end of core

4.2.3 $g(\mathbb{L})$ is

core

let L stand for a new identifier $\notin \mathbb{L} \cup \mathbb{V}$;

$\mathbb{L} \leftarrow \mathbb{L} \cup \{L\}$;

=> L

end of core

4.3 Substitutions

Let E be an <expression>; n be an integer (≥ 0); V_i be a <variable> or a <label>, different from each other, for i=1,2,....,n; and E_i be an <expression> or a <label>, for i=1,2,....,n. Then

$$E(\tfrac{E_1}{V_1}\cdots\tfrac{E_n}{V_n})$$

denotes the <expression> obtained from E after substitution of E_1, ...,E_n for V_1,....,V_n respectively and simultaneously, except those V_i, if any, contained in a<string>. (The exact and too long definition, carried out recursively along the course of construction of E is omitted.)

4.4 Refinements

When E is an <expression> the result of its refinement is roughly speaking, the <expression> obtained from E after substitution of the result of

$g(\mathbb{V})$ or $g(\mathbb{L})$

for each <variable> or <label> which is local to E or to a <block> or

<procedure notation> in E. Exactly, it is defined by recursion on the construction of E:

4.4.1 Let E be an <expression> of the form

"$A_0E_1A_1E_2A_2 \ldots A_{n-1}E_nA_n$"

where n is an integer (≥ 0); $E_1, E_2, \ldots, E_n$ are all direct constituents of E; and A_i is a sequence of <basic symbol>'s or empty for $i=0,1,2,\ldots,n$.

Then $r_0(E)$ is

<u>core</u>

$r(E_1)$ => F_1;

$r(E_2)$ => F_2;

....

$r(E_n)$ => F_n;

let F stand for (the <expression>)

"$A_0F_1A_1F_2A_2 \ldots A_{n-1}F_nA_n$";

=> F

<u>end of core</u>

4.4.2 Let E be a <block>, let $V_1, \ldots, V_n$ be all its proper <variable>'s, and let $L_1, \ldots, L_m$ be all its proper <label>'s, where n, m are integers (≥ 0).

r(E) is

<u>core</u>

$r_0(E)$ => E';

$g(\mathbb{V})$ => V_1';

....

$g(\mathbb{V})$ => V_n';

$g(\mathbb{L})$ => L_1';

....

$g(\mathbb{L}) \Rightarrow L_m'$;

let F stand for (the <expression>)

$$F'\left(\begin{matrix} V_1' & \cdots & V_n' & L_1' & \cdots & L_m' \\ V_1 & \cdots & V_n & L_1 & \cdots & L_m \end{matrix}\right);$$

=> F

<u>end of core</u>

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

"<u>procedure</u> ($T_1,\ldots,T_n$)T J"

where n is an integer (≥ 0); T_i is a <typifier> for i=1,2,....,n; T is a <typifier>; J is a <procedure donor>.

1) If J is empty, then r(E) is

<u>core</u>

$r_0(E)$ => F;

=> F

<u>end of core</u>

2) If J is of the form

"<u>by</u> (($V_1,\ldots,V_n$)F)"

where V_i is a <variable> different from each other, for i=1,2,....,n; F is an <expression>.

Then r(E) is

<u>core</u>

r(F) => F';

$g(\mathbb{V}) \Rightarrow V_1'$;

....

$g(\mathbb{V}) \Rightarrow V_n'$;

let F'' stand for (the <expression>)

$$F'\left(\begin{matrix} V_1' & \cdots & V_n' \\ V_1 & \cdots & V_n \end{matrix}\right);$$

let E' stand for (the <expression>)

"T($V_1',\ldots,V_n'$)F''";

=> E'

<u>end of core</u>

4.4.4 Let E be an <expression> other than <block> or <procedure notation>.

Then r(E) is

<u>core</u>

r_0(E) => F;

=> F

<u>end of core</u>

4.5 Elaborations

Let E be an <expression>.

The elaboration of E is defined recursively as follows:

```
e(E) is
    core
    if E=<variable>, then → Kvariable, else → next;
    if E=<go to statement>, then → Kgotostatement, else → next;
    if E=<dummy statement>, then → Kdummystatement, else → next;
    if E=<code call>, then → Kcodecall, else → next;
    if E=<closed expression>, then → Kclosedexpression, else → next;
    if E=<block>, then → Kblock, else → next;
    if E=<array element>, then → Karrayelement, else → next;
    if E=<structure element>, then → Kstructureelement, else → next;
    if E=<procedure call>, then → Kprocedurecall, else → next;
    if E=<effect notation>, then → Keffectnotation, else → next;
    if E=<real notation>, then → Krealnotation, else → next;
    if E=<bits notation>, then → Kbitsnotation, else → next;
    if E=<string notation>, then → Kstringnotation, else → next;
    if E=<reference notation>, then → Kreferencenotation, else → next;
    if E=<array notation>, then → Karraynotation, else → next;
    if E=<structure notation>, then → Kstructurenotation, else → next;
    if E=<procedure notation>, then → Kprocedurenotation,(else => L0);
Kvariable:
    if a(E)=able then → next, else => L0;
    let Q stand for q(E);
    => Q;
Kgotostatement:
    let E be of the form
        "go to L"
    where L is a <label>;
    => L;
```

Kdummystatement:

=> Q0;

Kcodecall:

let E be of the form

"code (S_1E_1,....,S_nE_n)T by (A)"

where n is an integer ($\geq$0),

S_i is a <selecter> for i=1,2,....,n,

E_i is an <expression> for i=1,2,....,n,

T is a <primary typifier>,

A is a <code body>;

let F stand for the express ion

"structure (S_1E_1,....,S_nE_n)";

e(F) if Q → next, L => L;

e(A(Q)) if Q' => Q', L => L;

Kclosedexpression:

let E be of the form

"(F)"

where F is an <expression>;

e(F) if Q => Q, L => L;

Kblock:

let E be of the form

"begin let V_1 be E_1;

....

let V_n be E_n;

L_1^1:....:$L_{i_1}^1$: F_1;

....

L_1^k:....:$L_{i_k}^k$: F_k end"

where n is an integer ($\geq$0),

```
         k is an integer (≥1),
         i_u is an integer (≥0) for u=1,2,....,k,
         V_j is a <variable> for j=1,2,....,n,
         E_j is an <expression> for j=1,2,....,n,
         F_u is an <expression> for u=1,2,....,k,
         L^u_v is a <label> for u=1,2,....,k,
                                v=1,2,....,i_u;
     a(V_j) ← inable for j=1,2,....,n;
     (e(E_j) if Q → next, L => L;
     a(V_j) ← able;
     q(V_j) ← Q;) for j=1,2,....,n
K111:     e(F_1) if Q → next, L → K121;
K112:     e(F_2) if Q → next, L → K121;
....
K11k:     e(F_k) if Q → next, L → K121;
     => Q;
K121:     if i_1=0 then → K122, else → next;
     (if L=L^1_v then → K111, else → next;) for v=1,2,....,i_1
K122:     if i_2=0 then → K123, else → next;
     (if L=L^2_v then → K112, else → next;) for v=1,2,....,i_2
K123:
....
K12k:     if i_k=0 then → K13, else → next;
     (if L=L^k_v then → K11k, else → next;) for v=1,2,....,i_k
K13:      => L;
Karrayelement:
     let E be of the form
          "F[E']"
```

where F is an <expression>,

E' is an <expression>;

if t(F) is a type of array style then → next, else => L0;

if t(E') is <u>real</u> then → next, else => L0;

e(F) if Q → next, L => L;

if w(Q)=∅ then → next, else → K21;

let v stand for the integer 1;

let u stand for the integer 0;

→ K22;

K21: let $w(Q)=\{<v,Q_v>,<v+1,Q_{v+1}>,\ldots\ldots,<u,Q_u>\}$

where v is an integer,

u is an integer (≥v),

Q_j is a quantity for j=v,v+1,....,u;

K22: e(E') if Q' → next, L => L;

let i stand for the integer round(w(Q'));

if i≥v then → next, else => L0;

if i≤u then => Q_i, else => L0;

Kstructureelement:

let E be of the form

"F[S]"

where F is an <expression>,

S is a <selector>;

if t(F) is a type of structure style then → next, else => L0;

e(F) if Q → next, L => L;

let $w(Q)=\{<S_1,Q_1>,\ldots\ldots,<S_n,Q_n>\}$

where n is an integer (≥0),

S_j is a <selector> for j=1,2,.....,n,

Q_j is a quantity for j=1,2,.....,n;

if $S=S_i$ for some i ($1\le i\le n$) then => Q_i, else => L0;

Kprocedurecall:

let E be of the form

"$F(E_1,\ldots.,E_n)$"

where n is an integer (≥ 0),

F is an <expression>,

E_j is an <expression> for j=1,2,....,n;

if t(F) is a type of procedure style, then → next, else => L0;

e(F) if Q → next, L => L;

let t(Q) be of the form

"procedure ($T_1,\ldots.,T_m$)T"

where m is an integer (≥ 0),

T_i is a type for i=1,2,....,m,

T is a type;

let w(Q) be of the form

"($V_1,\ldots.,V_m$)F'"

where V_i is a <variable> for i=1,2,....,m,

F' is an <expression>;

let E' stand for the <block> of the form

"begin let V_1 be T_1;

....

let V_m be T_m;

F' end";

if E' is legal then → next, else => L0;

if n=m then → next, else => L0;

(if $t(E_i)$ is T_i then → next, else => L0;) for i=1,2,....,m

if t(E') is T then → next, else => L0;

(let E_i' stand for the <expression> of the form

```
        "(E_i)";) for i=1,2,....,n
    let F'''=F'(E_1'/v_1 ,...., E_m'/v_m );
    r(F''') => F'';
    e(F'') if Q' => Q', L => L;
Keffectnotation:
    let M stand for the mode
        effect;
    let W stand for the value
        done;
    g(Q) => Q;
    m(Q) ← M;
    w(Q) ← W;
    => Q;
Krealnotation:
    let E be of the form
        "real H J"
    where H is a <real modifier>,
          J is a <real donor>;
    if H is empty then → K31, else → next;
    if H is of the form
        "[E_1:E_2:E_3]"
    where E_i is an <expression> or empty for i=1,2,3,
    then → K32, else → next;
    if H is of the form
        "[precision F]"
    where F is an <expression> or empty,
    then → K33, (else => L0);
K31:      if J is empty then → K331, else → next;
```

if J is an <integer donor> then → K311, else → next;

if J is a <fraction donor> then → K312, (else => LO);

K311: let R_3 stand for the integer w(J);

let R_2 stand for the integer 1;

let R_1 stand for the integer $-R_3$;

→ K326;

K312: let R stand for the real number w(J);

→ K332;

K32: if E_1 is empty then → K321, else → next;

if $t(E_1)$ is real then → next, else => LO;

$e(E_1)$ if Q_1 → next, L => L;

let R_1 stand for the real number $w(Q_1)$;

→ K322;

K321: let R_1 stand for the real number R1 (of implementation dependent);

K322: if E_2 is empty then → K323, else → next;

if $t(E_2)$ is real then → next, else => LO;

$e(E_2)$ if Q_2 → next, L => L;

let R_2 stand for the real number $w(Q_2)$;

→ K324;

K323: let R_2 stand for the integer 1;

K324: if E_3 is empty then → K325, else → next;

if $t(E_3)$ is real then → next, else => LO;

$e(E_3)$ if Q_3 → next, L => L;

let R_3 stand for the real number $w(Q_3)$;

→ K326;

K325: let R_3 stand for the real number R2 (of implementation dependent);

K326: let M stand for the mode

real [R_1:R_2:R_3];

→ K34;

K33: if F is empty then → K331, else → next;

if t(F) is real then → next, else => L0;

e(F) if Q → next, L => L;

let R stand for the real number w(Q);

→ K332;

K331: let R stand for the real number R3 (of implementation dependent);

K332: let M stand for the mode

real [precision R];

K34: let M' stand for the mode d(M);

if J is empty then → K341, else → next;

let W stand for the real number w(J);

→ K35;

K341: let W stand for an arbitrary real number;

K35: if $W_{M'} \neq \emptyset$ then → next, else => L0;

let X stand for the real number p(M',W);

g(Q) => Q';

m(Q') ← M';

w(Q') ← X;

=> Q';

Kbitsnotation:

let E be of the form

"bits H J"

where H is a <bits modifier>,

J is a <bits donor>;

```
     if H is empty then → K41, else → next;
     if H is of the form
          "[exact F]"
     where F is an <expression>,
     then → K411, else → next;
     if H is of the form
          "[exact]"
     then → K412, else → next;
     if H is of the form
          "[varying F]"
     where F is an <expression>,
     then → K414, else → next;
     if H is of the form
          "[varying]"
     then → K415, (else => L0);
K41:      if J is empty then → K412, else → next;
     let I stand for the integer length(w(J));
     → K413;
K411:     if t(F) is real then → next, else => L0;
     e(F) if Q → next, L => L;
     let I stand for the integer round(w(Q));
     → K413;
K412:     let I stand for the integer I1 (of implementation dependent);
K413:     let M stand for the mode
          bits [exact I];
     → K42;
K414:     if t(F) is real then → next, else => L0;
     e(F) if Q → next, L => L;
```

```
     let I stand for the integer round(w(Q));
     → K416;
K415:      let I stand for the integer I2 (of implementation dependent);
K416:      let M stand for the mode
           bits [varying I];
K42:       let M' stand for the mode d(M);
     if J is empty then → K421, else → next;
     let W stand for the bit-string w(J);
     → K43;
K421:      let W stand for an arbitrary bits;
K43:       if W_M'≠Ø, then → next, else => L0;
     let X stand for the bits p(M',W);
     g(Q) => Q';
     m(Q') ← M';
     w(Q') ← X;
     => Q';
Kstringnotation:
     let E be of the form
           "string H J"
     where H is a <string modifier>,
            J is a <string donor>;
     if H is empty then → K51, else → next;
     if H is of the form
           "[exact F]"
     where F is an <expression>,
     then → K511, else → next;
     if H is of the form
           "[exact]"
```

```
     then → K512, else → next;
     if H is of the form
          "[varying F]"
     where F is an <expression>,
     then → K514, else → next;
     if H is of the form
          "[varying]"
     then → K515, (else => L0);
K51:      if J is empty then → K512, else → next;
     let I stand for the integer length(w(J));
     → K513;
K511:     if t(F) is real then → next, else => L0;
     e(F) if Q → next, L => L;
     let I stand for the integer round(w(Q));
     → K513;
K512:     let I stand for the integer I3 (of implementation dependent);
K513:     let M stand for the mode
          string [exact I];
     → K52;
K514:     if t(F) is real then → next, else => L0;
     e(F) if Q → next, L => L;
     let I stand for the integer round(w(Q));
     → K516;
K515:     let I stand for the integer I4 (of implementation dependent);
K516:     let M stand for the mode
          string [varying I];
K52:      let M' stand for the mode d(M);
     if J is empty then → K521, else → next;
```

let W stand for the <string> w(J) (=J);

→ K53;

K521: let W stand for an arbitrary <string>;

K53: if $W_{M'} \neq \emptyset$, then → next, else => L0;

let X stand for the <string> p(M',W);

g($\mathbb{Q}$) => Q';

m(Q') ← M';

w(Q') ← X;

=> Q';

Kreferencenotation:

if E is of the form

"<u>reference</u>"

then → K61, else → next;

if E is of the form

"<u>reference</u> <u>nil</u>"

then → next, (else => L0);

let W stand for the empty set ∅;

→ K62;

K61: let Q stand for an arbitrary quantity in $\mathbb{Q}$;

let W stand for the set {Q};

K62: g($\mathbb{Q}$) => Q';

m(Q') ← <u>reference</u>;

w(Q') ← W;

=> Q';

Karraynotation:

if E is of the form

"<u>array</u> H F"

where H is an <array modifier>,

F is an <expression>,

then → K71, else → next;

if E is of the form

"array (E_1,....,E_n)"

where n is an integer (≥1),

E_i is an <expression> for i=1,2,....,n

then → K73, (else => L0);

K71: if H is empty then → K711, else → next;

if H is of the form

"[E_1:E_2]"

where E_i is an <expression> for i=1,2,

then → next, (else => L0);

if t(E_1) is real then → next, else => L0;

e(E_1) if Q_1 → next, L => L;

let I_1 stand for the integer round(w(Q_1));

if t(E_2) is real then → next, else => L0;

e(E_2) if Q_2 → next, L => L;

let I_2 stand for the integer round(w(Q_2));

→ K712;

K711: let I_1 stand for the integer 1;

let I_2 stand for the integer 0;

K712: let T stand for the type t(F);

let M stand for the mode

array [I_1:I_2]T;

let M' stand for the mode d(M);

let M'=array [v:u]T

where v is an integer,

u is an integer;

```
     if v>u then → K72, else → next;
     (e(F) if Q_j' → next, L => L;) for j=v,v+1,....,u
     let X stand for the set
          {<v,Q_v'>,<v+1,Q_v+1'>,....,<u,Q_u'>}
     → K74;
K72:      let X stand for the empty set Ø;
     → K74;
K73:      let T stand for the type t(E_1);
     let M stand for the mode
          array [1:n]T;
     let M' stand for the mode d(M);
     (if t(E_j) is T then → next, else => L0;
     e(E_j) if Q_j' → next, L => L;) for j=1,2,....,n
     let W stand for the set
          {<1,Q_1'>,<2,Q_2'>,....,<n,Q_n'>};
     let X stand for the set p(M',W);
k74:      g(Q) => Q'';
     m(Q'') ← M';
     w(Q'') ← X;
     => Q'';
Kstructurenotation:
     let E be of the form
          "structure (S_1E_1,....,S_nE_n)"
     where n is an integer (≥0),
           S_i is a <selector> for i=1,2,....,n,
           E_i is an <expression> for i=1,2,....,n;
     if n=0 then → K81, else → next;
     (e(E_j) if Q_j → next, L => L;) for j=1,2,....,n
```

let W stand for the set

$\{\langle S_1,Q_1\rangle,\ldots\ldots,\langle S_n,Q_n\rangle\}$;

(let T_i stand for the type $t(E_i)$;) for i=1,2,....,n

let M stand for the mode

<u>structure</u> <u>(</u>$S_1T_1,\ldots\ldots,S_nT_n$<u>)</u>;

→ K82;

K81: let W stand for the empty set Ø;

let M stand for the mode

<u>structure</u> <u>()</u>;

K82: $g(\mathcal{Q})$ => Q';

m(Q') ← M;

w(Q') ← W;

=> Q';

Kprocedurenotation:

let E be of the form

"<u>procedure</u> $(T_1,\ldots\ldots,T_n)$T J"

where n is an integer (≥0),

T_i is a <typifier> for i=1,2,....,n,

T is a <primary typifier>,

J is a <procedure donor>;

if J is empty, then → K91, else → next;

let J be of the form

"<u>by</u>($(V_1,\ldots\ldots,V_n)$E')"

where V_i is a <variable> for i = 1,2,....,n,

E' is an <expression>;

let W stand for the figure

$(V_1,\ldots\ldots,V_n)$E' ;

→ K92;

K91: let E' be an arbitrary legal <expression> without <mark>

and of the type T and of the form

"<u>begin</u> <u>let</u> V_1 <u>be</u> T_1;

....

<u>let</u> V_n <u>be</u> T_n;

E'' <u>end</u>";

let W stand for the figure

$(V_1, \ldots, V_n)E''$;

K92: let M stand for the mode

<u>procedure</u> $(T_1, \ldots, T_n)T$;

g(Q) => Q;

m(Q) ← M;

w(Q) ← W;

=> Q

<u>end of core</u>