

〔研究〕

General Tracer の研究(IV) --Tracer の skelton-- A Study of General Tracer (IV)--A Skelton of Tracer--

大谷木 重 夫
S. OHYAGI

Assume a semantical description of a Pascal-like language through G.H.C. has been already given. In this paper we give a tracer for that language using this description. The characteristic feature of this tracer are a trace for each syntactic object, abstract execution and backward trace e.t.c.

§ 1 はじめに

プログラムの指示意味論という分野も確立してから 20 年以上たつわけであるが、応用面ではいま一つ盛り上がりには欠けるところがある。本シリーズの研究の一つの目標は、プログラム意味論が役に立つことを示すことにある。当初の目的は、Prologのトレーサを使っていてbackができない不便さをどうにかならないかということであった。そしてPascal系言語やAssemblaのDebugでもbackが出来ないことがわかった。次にPascal系言語の semantics を λ -CalculusでなくてPrologで書いたらPrologのtracerが利用できて構文毎のtrace及び抽象実行ができるのではないかと考えた。更にその場合、local stack optimization を取りやめれば Prolog の interpreter は過去の変数の値を保存しているので、トレースの際、前進だけでなくbackもできる可能性があることに気づいた。こうして本研究がスタートしたわけである。

実際にPrologでPascal系言語の semantics を書く際Prologではなくてそのsubsetの G.H.C. で十分であることが判明した。G.H.C. は、そのinterpreterの制御構造がごく簡単なので、本シリーズでは全面的に採用した。

§ 2 準 備

問題とするPascal系言語には、B. N. F.により syntax が G.H.C. により semantics が与えられているものとする。例えば前論文で示した場合がそうである。そこでとりあげた例を再掲する。

```
program longestupsequence;  
  var x, n, m;
```

```
array a, b;  
procedure readarray(a, n)  
  label start:  
    begin x := 1 end;  
  label next:  
    read(x);  
    a[n] := x;  
    if eof then  
      goto next  
    else fi;  
  begin goto start;  
  goto next  
end;  
procedure solution(a, n:b, m)  
  var k, l, i  
  array c, d, e, e1  
  begin  
    if n=1 then  
      b[1] := a[1];  
      m := 1  
    else fi;  
    for k:=1 to n-1  
      do  
        c[k] := a[k+1]  
      od;  
    for k := 1 to n  
      do  
        if a[k]  $\geq$  a[1]  
          then a[k] := a[k]  
        else a[k] := -1
```

```

    fi
  od;
l := 1;
for k := 1 to n
  do
    if a[k] ≠ -1 then
      d[l] := a[k];
      l := l+1
    else fi
  od;
solution(c, n-1, e, s);
solution(d, l, e1, s1);
if s ≥ s1 then
  begin
    for i := 1 to s
      do
        b[i] := e[i]
      od
    end
  else
    begin
      for i := 1 to s1
        do
          b[i] := e1[i]
        od
      end
    fi
  end
procedure writesolution(b, m)
  var i
  begin
    for i := 1 to m
      do
        write(b[i])
      od
    end
  begin
    readarray(a, n);
    solution(a, n:b, m);
    writesolution(b, m)
  end
.

```

このプログラムのsyntax treeは

```

<program> ⇒ program <identifier>1 ;
  <variable block>2.
<identifier>1 ⇒ * longestupsequence

```

```

<variable block>2 ⇒
  var <identifier list>3;
  array <identifier list>4 ;
  <label block>5
<identifier list>3 ⇒ * x,n,m
<identifier list>4 ⇒ * a,b
<label block>5 ⇒ <procedure block>6
<procedure block>6 ⇒ <procedure or
  function block>7 <procedure block>8
<procedure or function block>7 ⇒
  <procedure or function heading>9
  <variable block>10
<procedure or function heading>9
  ⇒ procedure <identifier>11
  (<identifier list>12 :<value list>13)
<identifier>11 ⇒ * readarray
<identifier list>12 ⇒ nil
<identifier list>13 ⇒ * a, n
<variable block>10 ⇒ <label block>14
<label block>14 ⇒ label <identifier>15 :
  <statement>16 ; <label block>22
<identifier>15 ⇒ * start
<statement>16 ⇒ begin <statement>17 end
<statement>17 ⇒ <variable>18 := <expression>19
<variable>18 ⇒ <identifier>20
<identifier>20 ⇒ * x
<expression>19 ⇒ <constant>21
<constant>21 ⇒ 1
<label block>22 ⇒ label <identifier>23 :
  <statement>24 ; <label block>25
<identifier>23 ⇒ next
<statement>24 ⇒ <statement>25 ; <statement>26
<statement>25 ⇒ read(<variable>27)
<variable>27 ⇒ <identifier>28
<identifier>28 ⇒ * x
<statement>26 ⇒ <statement>29 ; <statement>30
<statement>29 ⇒ <variable>31 := <expression>32
<variable>31 ⇒ <identifier>33 [<expression>34]
<identifier>33 ⇒ a
<expression>34 ⇒ <variable>35
<variable>35 ⇒ <identifier>36
<identifier>36 ⇒ n
<expression>32 ⇒ <variable>37
<variable>37 ⇒ <identifier>38
<identifier>38 ⇒ x
<statement>30 ⇒ if <expression>39 then

```

$\langle \text{statement} \rangle^{40} \Rightarrow \underline{\text{else}} \langle \text{statement} \rangle^{41} \underline{\text{fi}}$
 $\langle \text{expression} \rangle^{39} \Rightarrow \langle \text{unary operator} \rangle^{42}$
 $\langle \text{expression} \rangle^{43}$
 $\langle \text{unary operator} \rangle^{43} \Rightarrow \neg$
 $\langle \text{expression} \rangle^{44} \Rightarrow \underline{\text{eof}}$
 $\langle \text{statement} \rangle^{40} \Rightarrow \underline{\text{goto}} \langle \text{identifier} \rangle^{45}$
 $\langle \text{identifier} \rangle^{45} \Rightarrow \text{next}$
 $\langle \text{statement} \rangle^{41} \Rightarrow \text{null}$
 $\langle \text{label block} \rangle^{25} \Rightarrow \langle \text{procedure block} \rangle^{46}$
 $\langle \text{procedure block} \rangle^{46} \Rightarrow \underline{\text{begin}}$
 $\langle \text{statement} \rangle^{47} \underline{\text{end}}$
 $\langle \text{statement} \rangle^{47} \Rightarrow \langle \text{statement} \rangle^{48} ;$
 $\langle \text{statement} \rangle^{49}$
 $\langle \text{statement} \rangle^{48} \Rightarrow \underline{\text{goto}} \langle \text{identifier} \rangle^{50}$
 $\langle \text{identifier} \rangle^{50} \Rightarrow \text{start}$
 $\langle \text{statement} \rangle^{49} \Rightarrow \underline{\text{goto}} \langle \text{identifier} \rangle^{52}$
 $\langle \text{identifier} \rangle^{52} \Rightarrow \text{next}$
 $\langle \text{procedure block} \rangle^8 \Rightarrow \langle \text{procedure or function block} \rangle^{65} \langle \text{procedure block} \rangle^{66}$
 $\langle \text{procedure or function block} \rangle^{65} \Rightarrow$
 $\langle \text{procedure or function heading} \rangle^{53}$
 $\langle \text{variable block} \rangle^{54}$
 $\langle \text{procedure or function heading} \rangle^{53} \Rightarrow$
 $\underline{\text{procedure}} \langle \text{identifier} \rangle^{55}$
 $(\langle \text{identifier list} \rangle^{56} :$
 $\langle \text{value list} \rangle^{57})$
 $\langle \text{identifier list} \rangle^{56} \Rightarrow \langle \text{identifier} \rangle^{58} ,$
 $\langle \text{identifier list} \rangle^{59}$
 $\langle \text{identifier} \rangle^{55} \Rightarrow \text{solution}$
 $\langle \text{identifier} \rangle^{58} \Rightarrow \text{a}$
 $\langle \text{identifier list} \rangle^{59} \Rightarrow \langle \text{identifier} \rangle^{60}$
 $\langle \text{identifier} \rangle^{60} \Rightarrow \text{n}$
 $\langle \text{value list} \rangle^{57} \Rightarrow \langle \text{value} \rangle^{61} , \langle \text{value list} \rangle^{62}$
 $\langle \text{value} \rangle^{61} \Rightarrow \langle \text{identifier} \rangle^{62}$
 $\langle \text{identifier} \rangle^{62} \Rightarrow \text{b}$
 $\langle \text{value list} \rangle^{63} \Rightarrow \langle \text{value} \rangle^{64}$
 $\langle \text{value} \rangle^{64} \Rightarrow \text{m}$
 $\langle \text{variable block} \rangle^{54} \Rightarrow$
 $\underline{\text{var}} \langle \text{identifier list} \rangle^{65}$
 $\underline{\text{array}} \langle \text{identifier list} \rangle^{66}$
 $\langle \text{label block} \rangle^{67}$
 $\langle \text{identifier list} \rangle^{65} \Rightarrow \langle \text{identifier} \rangle^{68}$
 $, \langle \text{identifier list} \rangle^{69}$
 $\langle \text{identifier} \rangle^{68} \Rightarrow \text{k}$
 $\langle \text{identifier list} \rangle^{69} \Rightarrow \langle \text{identifier} \rangle^{70}$
 $, \langle \text{identifier list} \rangle^{71}$

$\langle \text{identifier} \rangle^{70} \Rightarrow \text{l}$
 $\langle \text{identifier list} \rangle^{71} \Rightarrow \langle \text{identifier} \rangle^{72}$
 $\langle \text{identifier} \rangle^{72} \Rightarrow \text{i}$
 $\langle \text{identifier list} \rangle^{66} \Rightarrow \langle \text{identifier} \rangle^{73}$
 $, \langle \text{identifier list} \rangle^{74}$
 $\langle \text{identifier} \rangle^{73} \Rightarrow \text{c}$
 $\langle \text{identifier list} \rangle^{74} \Rightarrow \langle \text{identifier} \rangle^{75}$
 $, \langle \text{identifier list} \rangle^{76}$
 $\langle \text{identifier} \rangle^{75} \Rightarrow \text{d}$
 $\langle \text{identifier list} \rangle^{76} \Rightarrow \langle \text{identifier} \rangle^{77}$
 $, \langle \text{identifier list} \rangle^{78}$
 $\langle \text{identifier} \rangle^{77} \Rightarrow \text{e}$
 $\langle \text{identifier list} \rangle^{78} \Rightarrow \langle \text{identifier} \rangle^{79}$
 $\langle \text{identifier} \rangle^{79} \Rightarrow e_1$

等等、このプログラムのsemanticsは

$m(p) \llbracket \langle \text{program} \rangle \rrbracket (v^*, y^*)$
 $= m(p) \llbracket \underline{\text{program}} \langle \text{identifier} \rangle^1 :$
 $\langle \text{variable block} \rangle^2. \rrbracket (v^*, y^*)$
 $= m(p) \llbracket \underline{\text{program}} \text{longestupsequence} :$
 $\langle \text{variable block} \rangle^2 \rrbracket (v^*, y^*)$
 $:- m(s) \llbracket \underline{\text{procedure}} \text{longestupsequence}(:) :$
 $\langle \text{variable block} \rangle^2 ;$
 $\underline{\text{begin}} \text{longestupsequence} \underline{\text{end}} \rrbracket$
 $(e\text{-init}, s \leftarrow T, \text{sin}(v^*), \text{sout}(\text{nil}),$
 $e', s', \text{sin}', \text{sout}(y^*)), !.$

$\langle \text{procedure block} \rangle^0$
 $\Rightarrow \langle \text{procedure or function block} \rangle^{-1}$
 $\langle \text{procedure block} \rangle^{-2}$
 $\Rightarrow \underline{\text{procedure}} \text{ or function heading} \rangle^{-3}$
 $\langle \text{variable block} \rangle^2$
 $\langle \text{procedure block} \rangle^{-2}$
 $\Rightarrow^* \text{procedure} \langle \text{identifier} \rangle^4 ($
 $\langle \text{identifier list} \rangle^5 : \langle \text{value list} \rangle^6$
 $\langle \text{variable block} \rangle^2$
 $\langle \text{procedure block} \rangle^{-2}$
 $\Rightarrow^* \underline{\text{procedure}} \text{longestupsequence}(:) :$
 $\langle \text{variable block} \rangle^2$
 $\langle \text{procedure block} \rangle^{-2}$
 $\langle \text{procedure block} \rangle^{-2}$
 $\Rightarrow \underline{\text{begin}} \langle \text{statement} \rangle^{-7} \underline{\text{end}}$
 $\Rightarrow \underline{\text{begin}} \langle \text{identifier} \rangle^{-8} ($
 $\langle \text{identifier list} \rangle^{-9} :$
 $\langle \text{value list} \rangle^{-10}) \underline{\text{end}}$
 $\Rightarrow^* \underline{\text{begin}} \text{longestupsequence}(:) \underline{\text{end}}$

より

$m(p) \llbracket \langle \text{program} \rangle \rrbracket (v^*, y^*)$

```

:- m(s) [ [ <procedure block>° ] ] (e-init
, s<- T, sin(v*), sout(nil), e', s', sin', sout(y*)),!.
m(s) [ [ <procedure block>° ] ] (e, s, sin, sout, e', s', sin'
, sout')
= m(s) [ [ procedure longestupsequence(:):
<variable block>²
<procedure block>² ] ] (e, s, sin,
sout, e', s', sin', sout')
:- m(s) [ [ <procedure block>² ] ] (e
(longestupsequence)<-
p [ [ procedure longestupsequence
(:):<variable block>² ] ],
s(longestupsequence)<- T, sin,
sout, e', s', sin', sout'),!.
m(s) [ [ <procedure block>² ] ] (e, s, sin, sout,
e', s', sin', sout')
= m(s) [ [ begin longestupsequence end ] ]
(e, s, sin, sout, e', s', sin', sout')

```

より

```

m(s) [ [ <procedure block>² ] ] (e, s, sin, sout, e', s',
sin', sout')
:- m(s) [ [ longestupsequence ] ]
(e, s, sin, sout, e', s', sin', sout'),!.

```

また

```

m(s) [ [ longestupsequence ] ] (
e(longestupsequence)=p [ [ procedure
longestupsequence variable_block ] ],
s, sin, sout, e', s', sin', sout')
:- p [ [ procedure longestupsequence
variable_block ] ]
(e, s, sin, sout, e', s', sin', sout'),!.
p [ [ procedure longestupsequence
<variable_block>² ] ]
(e, s, sin, sout, e', s', sin', sout')
:- m(s) [ [ <variable_block>² ] ]
(e(longestupsequence)<-
p [ [ procedure longestupsequence
<variable_block>² ] ],
s(longestupsequence)<- T,
sin, sout, e', s', sin', sout'),!.

```

<variable_block>² \Rightarrow

```

var x, n, m
array a, b
<label block>⁵

```

より

```

m(s) [ [ <variable_block>² ] ]

```

```

(e, s, sin, sout, e', s', sin', sout')
= m(s) [ [ var x, n, m array a, b:
<label block>⁵ ] ]
(e, s, sin, sout, e', s', sin', sout')
:- m(s) [ [ <label block>⁵ ] ]
(e(x)<- T, e(n)<- T, e(m)<- T,
e(a)<- T, e(b)<- T,
s(x)<- , s(n)<- , s(m)<-
s(a)<- , s(b)<- ,
sin, sout, e', s', sin', sout'),!.
<label block>⁵
 $\Rightarrow$  procedure readarray(:a, n)
<variable_block>¹⁰
<procedure block>⁸

```

より

```

m(s) [ [ <label block>⁵ ] ]
(e, s, sin, sout, e', s', sin', sout')
= m(s) [ [ procedure readarray(:a, n)
<variable_block>¹⁰
<procedure block>⁸ ] ]
(e, s, sin, sout, e', s', sin', sout')
:- m(s) [ [ <procedure block>⁸ ] ]
(e(readarray)<-p [ [ procedure readarray(:a, n)
<variable_block>¹⁰ ] ] (:a, n),
s(readarray)<- T, sin, sout, e', s', sin', sout'),!.

```

以下同様。

§ 3 トレーサのスケルトン

前節で見たように一般にプログラムの生成木が与えられ、その意味が各非終端記号毎に意味関数mによるG. H. C. プログラムで与えられているものとする。このときこのプログラムをトレースするトレーサを assert, retract文を含むG.H.C.で書こう。

トレーサの機能を概説する。まず各構文に意味記述が

$$m(u) [[A]] (f(x)) :- B_1, \dots, B_n$$

.

.

.

と与えられているものとする。そして構文A のトレースをしたいものとする。始めに

$$\text{trace}(A, g(x)) ?$$

と質問する。m(u) [[A]] に対する G.H.C. の interpreter の初期設定をし、

$$A \xrightarrow{*} y \quad \text{但し } y \text{ は終端記号列}$$

なるyをdisplayする。

そして次にexit, skip, step, backの内どれを選択するか質問する。exitならとまる。skipなら $m(u) \llbracket A \rrbracket (g(x))$ を評価して終わる。step ならば

$$m(u) \llbracket A \rrbracket (f(x)) :- B_1, \dots, B_n$$

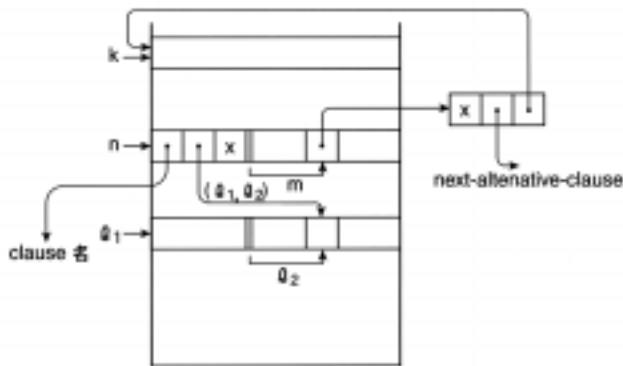
の B_1, \dots, B_n 側を評価しスタックを積み続け最初にぶつかった項

$$m(u') \llbracket B \rrbracket (g'(x))$$

の直前で止まり, $B \xrightarrow{*} z$ (z は終端記号列)となる z をdisplayする。例えば

$$m(u) \llbracket A \rrbracket (f(x)) :- m(u') \llbracket B \rrbracket (g(x)), \dots$$

だとするとこのあとskipすると現在のstackの位置をpreviousにしまい $m(u') \llbracket B \rrbracket (g(l(x)))$ を評価する。その後backの場合にはpreviousの値を知ってそれまでのstackの値を開放して、次のコマンドをまつ。ここで使うstackの一般的構造について述べる。



現在注目しているtermがスタックの n 番目の m 個目のtermとする。 n 番目のスタックの最初のelementはclause名を差す。第二elementはcallerへのポインター(l_1, l_2)である。第三elementは変数の初期値, 第四番目は全体としてこのclauseのtermに関する情報が入り, m 番目のtermの第一elementは変数の値, 次がnext-alternative-clauseになっており, 最後がこのtermを展開した内容を差すpointerこの場合はstack topの k を表す。

$trace(A, g(X)) :-$

```
asserta(Stack(1, :-m(u) \llbracket A \rrbracket (g(x))', (0, 0),
x, << , first_clause(m(u) \llbracket A \rrbracket ), 2>>),
asserta(curposition(1, 1)), asserta(top(1)),
asserta(previous(1)), out(A),
body, !.
```

トレースをかけるとまずstackを初期化する。即ちスタックの一番目に $:-m(u) \llbracket A \rrbracket (g(x))$ をしまい, 次にcallerはないから $(0, 0)$, 変数 x は何も代入されてないからそのまましまい, 最後に $clause :-m(u) \llbracket A \rrbracket (g(x))$ のAnd partについての情報をしまう。それは変数の値は決まっていから, 次に $m(u) \llbracket A \rrbracket$ のfirst clause名, 最後に

$m(u) \llbracket A \rrbracket$ の展開に対する情報を差すstack pointerここでは2がしまわれる。現在注目しているtermの位置は $(1, 1)$ であるから $curposition(1, 1)$, stack topの位置は1であるから $top(1)$, backしてきたときに使う情報としてstackの位置1をpreviousにしまう。そして $A \xrightarrow{*} x$ (x は終端記号列)となる x を出力するために $out(A)$ をよぶ。そして本体bodyに入る。

```
body :- read(x), branch_command(x), !.
```

```
branch_command(exit) :- !.
```

```
branch_command(skip) :-
```

```
curposition(n, m), top(k),
asserta(previous(k)), Stack(n, c, p, x, Y),
son_process(n, m, k, c, Y), body, !.
```

```
branch_command(step) :-
```

```
curposition(n, m), top(k), asserta(previous(k)),
Stack(n, c, p, x, Y),
find_non_terminal(n, m, k, c, p, x, Y), body, !.
```

```
branch_command(back) :-
```

```
previous(k), top(n), Stack(k, c, (p1, p2), x, Y),
retract(curposition(u, v)), retract(top(n)),
asserta(curposition(p1, p2)), asserta(top(k)),
Pop(n-k+1), cur_term_head(c, p2, Q), out(Q),
body, !.
```

プログラムの本体即ちbodyはまずオペレータからの入力を待ち, 入力exit, skip, step, backのいずれかがあるとそれに対応する行動即ちbranch_commandをとる。

入力がexitであれば何もしないで終わる。skipであれば現在注目しているtermの情報 (n, m) , スタックトップの位置 k , 及び第 n 番目のstackの情報を読み取り, 表示されたプログラムを実行して前に進むルーチン $son_process$ を行う。その際将来のbackコマンドに備えてpreviousにスタックトップの位置 k をしまう。入力がステップの場合, 同様に現在の情報を集めて現在の構文からderivateされる最初の構文を捜すルーチン $find_non_terminal$ を行う。最後に入力コマンドがbackの場合であるが, この場合はまずpreviousの内容 k を手に入れて, そのときのStackの内容 $Stack(k, c, (p_1, p_2), x, Y)$ から k より始まるprocessを起動するtermの位置 (p_1, p_2) を得る。そして $curposition$ を (p_1, p_2) に top を k に書き換えてStackを $(n-k+1)$ Popする。 c と p_2 よりprocess名 Q を得て, Q に対応するプログラムを out を用いて表示する。

以上が終わると再びbodyを呼ぶのである。

```
son_process(n, m, c, k, Y) :-
```

```
term(c, m, m(u) \llbracket A \rrbracket (f(y))),
element(Y, m-1, <x, c', z>),
```

```
branch_son_process(n, m, c, k, Y,
```

```
m(u) [ [ A ] ] (f(x)),!.
```

まずson_processの方から解説する。節cのm番目のtermがm(u) [[A]] (f(y)) という形をしていることを確認する(term)。m番目のtermについての情報<x, c', z>をYからとってきてbindingの状態xを知る。そしてm(u) [[A]] (f(x))を実行するプログラムbranch_son_processを起動する。

```
branch_son_process(n, m, c, k, Y, m(u) [ [ A ] ] (f(x))) :-
  m(u) [ [ A ] ] (f(x)),!, tran(Y, m, x, Y'),
  retract(Stack(n, c, p, x', Y)),
  assert(Stack(n, c, p, x', Y')), display_id, advance,
  next.
```

branch_son_process はAで指定されたプログラムm(u) [[A]] (f(x)) を実行し、実行の結果得られた変数のbinding情報xを用いてYのm欄のbinding情報を書き換え(tranと続くretractとassert)、変数の全ての値を表示し(display_id)、次の実行文m(u') [[B]] (g(y))を捜しに行く(advanceとnext)。

```
branch_son_process(0, 0, c, k, Y, B) :-
  write_end,!.
```

```
branch_son_process(n, m, c, k, Y, B) :-
  Stack(n, c, (t, u), x, Y),
  branch_after_cut(n, m, c, (t, u), x, Y),!.
```

注目しているtermが無い場合すなわちbranch_son_processの(n,m)欄が(0,0)のときはendを出力して終わる。そうでない場合は、現在注目しているtermの位置がcutの前か後かを判別し、backtrackする手続きbranch_after_cutを行う。

```
branch_after_cut(0, 0, c, (t, u), x, Y) :- !.
branch_after_cut(n, m, c, (t, u), x, Y) :-
  (m ≥ m!(c)),!,
  Stack(t, c', (t', u'), x', Y'), top(k), Pop(k-n),
  retract(curposition(n, m)),
  asserta(curposition(t, u)),
  retract(top(k)), asserta(top(n)),
  branch_after_cut(t, u, c', (t', u'), x', Y').
```

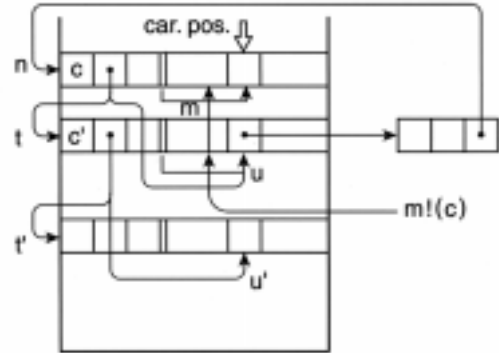
```
branch_after_cut(n, m, c, (t, u), x, Y) :-
  (m < m!(c)),!,
  Stack(t, c', (t', u'), x', Y'),
  element(Y', u, < x'', c'', w>),
  blanch_last_clause(t, u, c'', n).
```

branch_after_cutはmがm!(c)即ちcのカットの位置より右にあるときはt番目のスタックからcallerへのポインタ(t', u'), 変数のbinding情報x', And項の情報Y'をうる。そしてスタックをnの位置迄Popして現在注目している項の位置curpositionを(t, u)にsetし、スタックトップの位置をnと

して、(t, u)で失敗が起こったとして、更にbranch_after_cutを呼ぶ。mがm!(c)より左にある時は、Stackのt番目を開きそのAnd_terms Y'とuから(t, u)の差す項のnext_clause c"を得る。そしてbranch_last_clauseにより(t, u)をnext_clauseであるc"から展開していく。

```
branch_last_clause(t, u, nil, n) :-
  Stack(t, c', (t', u'), x', Y'),
  branch_after_cut(t, u, c', (t', u'), x', Y'),!.
```

項(t, u)のnext alternative欄がnilであった場合



branch_last_clause(t, u, nil, n) が呼ばれるわけであるが、これは項(t, u)が失敗した場合と同じであるのでbranch_after_cutを呼べばよい。

```
branch_last_clause(t, u, c'', w) :-
  Stack(t, c', (t', u'), x, Y), top(k),
  Pop(k-w+1), retract(Stack(t, c', (t', u'), x, Y')),
  tran2(Y'', u, < , next(c''), w >, Y'),
  asserta(Stack(t, c', (t', u'), x, Y')),
  retract(top(k)), asserta(top(w-1)),
  retract(curposition(n, m)),
  asserta(curposition(t, u)), next,!.
```

項(t, u)のnext_alternative欄がnilでない場合。t番目のStackを開いてclause名c', caller(t', u'),最初のbinding情報x及びAnd_part Yを得る。次にスタックをelement(Y, u, <z, c'', w>)からw-1の位置迄Popする。次にtran2でYのu番目を以降を

```
< , next(c''), w>, < , first(term(u+1, c')), >,
..., < , first(term(length(c'), c')), >
```

で置き換えたものY'を得る。asserta, retract文によりStackのt番目の値をYのかわりにY'と書き換える。topをkからw, curpositionを(n,m)から(t, u)へ変更してprocess(t, u)を展開して出会う最初のm(u) [[A]] 迄進むルーチンnextを行う。

```
advance :-
  curposition(n, m), top(k),
  Stack(n, c, (i, j), x, Y),
```

```

branch_length (n, m, i, j, Y),!.
branch_length (n, m, i, j, Y) :-
    (length (Y) > m),!, retract (curposition (n, m)),
    asserta (curposition (n, m+1)).
branch_length(n, m, i, j, Y) :-
    (length (Y) = m),!,
    retract (curposition (n, m)),
    asserta (curposition (i, j)),
    RTN (n, m, i, j), advance.

```

advanceはcurpositionを(n, m), topをkとするとき, n番目のStackの内容 (n, c, (i, j), x, Y)を得る。そしてmがYの最後の要素を差すとき以外はcurpositionを一步進めて (n, m+1) とする。mがYの最後の要素を差す場合は, RTNにより変数の値をcallerである(i, j)にもどしcurpositionを(i, j) とする。

```

next :-
    curposition (n, m), top (k),
    Stack (n, c, p, x, Y),
    branch_next (n, m, k, c, p, x, Y),!.
branch_next(n, m, k, c, p, x, Y) :-
    term (c, m, m (u) [ A ] (f(x))),!, out ( A).
branch_next (n, m, k, c, p, x, Y) :-
    branch_next1 (n, m, k, c, p, x, Y).
branch_next1(n, m, k, c, p, x, Y) :-
    term (c, m, B), cur_clause (Y, m, β),
    initialize (β, Y'), (length (β) ≥ 1),!,
    head (β, B), element (Y, m-1, < xm-1, c', w >),
    y is B-1 (B(xm-1) B' (y')),
    asserta (Stack (k+1, β, (n, m), y, Y')),
    retract(curposition (n, m)),
    asserta (curposition (k+1, 1)), retract (top(k)),
    asserta (top(k+1)), next.
branch_next1 (n, m, k, c, p, x, Y) :-
    term (c, m, B), curclause (Y, m, β), initial (β, Y'),
    (length (β)=0),!,head (β, B'),
    element (Y, m-1, < xm-1, c', w>),
    y is B-1 (B'(y') B(xm-1)),
    retract (Stack(n, c, p, x, Y)),
    tran2 (Y, m, < y, next(β), k+1 >, Y''),
    asserta (Stack(n, c, p, x, Y''),
    retract (top (k)), asserta (top (k+1)), advance,
    next.

```

nextは現在位置, Stack topの位置, 現在位置でのスタックの内容を調べて, branch_next へとぶ。
branch_next は, 現在位置のtermが何らかの構文の意味になっている場合はその構文に対応するプログラムコード

をoutにより表示してとまる。

そうでない場合。現在のtermに適用されているclause c のm番目の項をB, このtermのnext_clause部をβ, βの長さが1 以上の場合βのAnd部を初期化したものをY', βの頭をB', c のAnd部Yの m-1 番目を<x_{m-1}, c', w> とする。B(x_{m-1})とB'をunifyしてB'⁻¹することによりB'の変数のbinding yを得る。こうしてcurtermに対応するStackの内容(k+1, β,(n, m), y,Y')を得る。これをStackにpushし, curpositionを (n, m)から(k+1, 1) へ移す。この時, 同時にtopをkから k+1へ変える。βの長さが0の場合, B(x_{m-1})とB'をunifyしてB'⁻¹することによりBの変数のbinding yを得, それをtran2 によりYのm番目にもどし, next(β)をβのかわりにしまい, stackの位置k+1としてYからY"を得る。こうしてStackのn番目を (c, p, x, Y") に書き換えてStack topの位置をk+1にsetする。その上で一つこまをadvanceにより進めまたnextを呼ぶ。

```

find_non_terminal(n, m, k, c, p, x, Y) :-
    branch_next1(n, m, k, c, p, x, Y),!.
find_non_terminal は branch_next1と同じである。

```

ここでPopの際

```

Pop(0) :- !.
Pop(n) :- top(k), Pop1(k), retract(top(k)),
    asserta(top(k-1)), Pop(n-1),!.
Pop1(k) :- previous(k),!, retract(previous(k)),
    retract(Stack(k, c, p, x, Y)).
Pop1(k) :- retract(Stack(k, c, p, x, Y)),!.

```

のようにpreviousも減らせていくことに注意したい。

§ 4 おわりに

本トレーサではidentifierの取り扱いに問題があり, やや実用性に欠けるところがある。これを克服することが今後の課題である。

謝 辞

本論文は情報アーキテクチャ分散システム研究室で著者が行った研究をまとめたものである。研究室長であった塚本亨治氏および研究室の皆さまには公私にわたりたいへんお世話になりました。ここに厚く御礼申し上げます。また論文の作成に当たりましては大蒔情報アーキテクチャ部長にアドバイスをいただきました。また内容につきましては木下算譜言語論代表世話人にお世話になりました。

参 考 文 献

- 1) S. OHYAGI " General Tracer の研究--G.H.Cによる
Pascal系言語の意味記述--" 本シリーズ
(1998. 3.12受付)

著 者 紹 介

A Study of General Tracer (I)

General Tracer の研究 (II)

General Tracer の研究 (III)

General Tracer の研究 (IV)



大谷木 重夫

Shigeo OHYAGI

情報アーキテクチャ部

プログラム意味論とその応用の研究に従事。現在、新プログラミング言語とそのインタープリターの研究を行っている。