GPMとそのプログラム

和田 英一[1,a)]

概要：Christopher Strachey 他が, 英国製商用計算機 Atlas などのプログラム言語 CPL の実装手法として, GPM(general purpose macrogenerator) を開発したのは 1960 年頃のことだ. いまでは古典になり, 知る人も少くなった GPM は, 例えばマクロ a を`$def,a,<b~1d>;` と定義し, `$a,c;` と呼び出すと, `~1` が第 1 引数なので, `bcd` が得られ, `1+`を`$def,1+,<$1,2,3,4,5,6,7,8,9,10,$def,1,<~>~1;;>;` と定義し, `$1+,4;` と呼び出すと, 局所定義のマクロ `1` が起動して, `5` が得られるものである. これだけでどの程度のことが出来るか.
GPM の論文にある例題の評価法を理解すべく, 処理系を書いたのはかなり前だが, 最近 その例題以外にも GPM 処理系で走らせるプログラムをいくつか書いてみた.
基底にある GPM には算術演算もないので, 例題にはその辺から書いてあり, それを活用すれば, 小さい整数を使う ある程度のプログラムの書けることが分った.
GPM による風変りなプログラミングのノウハウ, プログラミング支援環境, さらに GPM 処理系の実装法などについて述べる.

キーワード：マクロジェネレータ, 記号処理, 文字列処理, スタック実装

GPM の使い方

英国ケンブリッジ大学の EDSAC は, プログラム記憶方式の最初の実用計算機であった. これを基本として商用計算機 Titan や Atlas が開発され, それで使うプログラム言語として, 当時制定されたばかりの Algol 60 を手本にした CPL が設計, 開発された [0].
CPL のコンパイラを作るにあたり, 例えばスタック操作では似たようなアセンブリコード列を繰り返し書く場合が多いので, 多少のパラメータを取り入れ, アセンブリコード列を吐き出すシステムが有用であろうということになった.
その解決策として提案され, 開発された記号処理系がこのマクロ言語 GPM である [1].

[1] IIJ 技術研究所
[a)] eiiti.wada@nifty.com

図 0 CPL GPM のテープ [2]

Cristopher Strachey はそれをかなり一般的な仕様にしたので, 出来上がったものは, それなりに面白いものであった. (使い難いという説もあり)
例えば 1 個の引数をとるマクロ a を, `$def,a,<b~1d>;` と定義すると, `~1` が第 1 引数を表わし, `$a,c;` と呼び出すと, マクロの定義が評価され, `~1`

に実引数の c が代入されて, 文字列 bcd が返えるものである. (元の論文では, マクロ呼出しの先頭は section sign (§) だが, ここでは$を使う.)
今の例を Scheme で書いてみると
```
(define a (lambda (~1) (list 'b ~1 'd)))
(a 'c) ⟹ (b c d)
```
のようなのもである.
~0 はマクロ名を表わし, $def,a,<~0b~1d>; の定義なら, 呼出し$a,c; は abcd になる.
定義自体もマクロ呼出しであり, その際, 定義本体の~1 が評価されないよう, <と>で囲む.
上の$a,c; の例では, 呼出しのマクロ名も実引数も文字列であったが, そういう場所にもマクロ呼出しが書ける.
$a,$a,c;; と呼び出すと, まず第 1 引数が評価されて bcd になり, それで a が呼び出されるので, 最終結果は bbcdd になる.
マクロ名がマクロ呼出しの例では, マクロ bcd を, $def,bcd,<b~1c~2d>; と定義しておき, $$a,c;,e,f; と呼び出すと, まず$a,c; が呼び出されて, bcd になり, 次に$bcd,e,f; の呼出しになって, becfd が出来上がる.
素晴らしいのはマクロ呼出しのなかに局所定義が書けることである. $a,c,$def,a,<b~1d>;; のように書くと, 第 2 引数が a の定義になっており, 最後のセミコロン; でマクロ a の評価が始まる時には, a の定義は出来ているのである.
これを局所定義といい, このマクロ評価が終ると局所定義は消える.
同じマクロ名のマクロが複数定義されていると, 最後に定義されたものが使われる. その様子を示すのが次の例だ. (各行の左端の斜体数字は行番号)
0 $def,a,b;
1 $a,$def,a,c;,$def,a,d;; ⟹ d
2 $a; ⟹ b
行 *0* でマクロ a を b と定義する. 行 *1* でマクロ a を呼び出すが, a の局所定義を 2 個持っている. この場合, 後の方の$def,a,d; が有効であり, 評価は d である. この行の評価が終わると局所定義は削除され, 次の行 *2* の評価では最初の a の定義が復活し, この評価は b である.
この機能を使い条件式 (if (eq? α β) γ δ) が書ける. つまり
$$\alpha,$def,\alpha,<\delta>;,$def,\beta,<\gamma>;;$$
とすれば, $\alpha = \beta$ なら後方の β の定義が使われて γ が得られ, そうでないなら前方の α の定義が生き, δ となる.
```
$def,if,<$~1,$def,~1,~4;,$def,~2,~3;;>;
$if,x,y,t,f; ⟹ f
$if,x,x,t,f; ⟹ t
```
元の論文にある suc(後者) の定義は次のようだ.
```
$def,suc,<$1,2,3,4,5,6,7,8,9,10,
  $def,1,<~>~1;;>;
```
これを$suc,3; で呼び出すと, suc の本体でマクロ 1 が呼び出される. このマクロの定義は後の方にあり, 定義は引数が代入されて$def,1,~3; になる. 従ってマクロ呼出しの先頭を 0 として 3 番目の 4 が得られる. $suc,0; で呼び出すと, マクロ 1 の名前が取り出されて, 1 が得られる.
tilde(~) に続く引数は数字 1 文字なので, 最大が 9 であり, その時の suc で 10 が得られる. 元の論文には他の文字を使って 10 以上に対応させてもよいと書いてある. 例えば ASCII コードの第 3 列 0123456789:;<=>?を使い
```
$def,a,
  <~?~<~=~>~;~:~9~8~7~6~5~4~3~2~1~0>;
$a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p;
```
⟹pmnolkjihgfedcba (m と o の位置に注意:-)
suc を下請けに, 2 桁の十進数の後者を求めるマクロ successor がある. $successor,2,3;⟹2,4, $successor,2,9;⟹3,0 になるのもである.
```
$def,successor,<$~2,
  $def,~2,~1<<,>$suc,>~2<;>;,
  $def,9,<$suc,>~1<;<,>0>;;>;
```
当然ながら 1 の桁 (~2) が 9 かどうかと調べる必要がある. それが 9 なら 10 の桁に 1 を足し, 1 の桁は 0 とする. そうでないなら, 10 の桁はそのままで, 1 の桁に 1 を足す. 上の定義は
```
(define (suc a) (1+ a))
(define (successor a b)
 (if (= b 9) (list (suc a) 0)
  (list a (suc b))))
```