

Regular Expression の同値性判定プログラム

電通大 金山 裕

鈴木康年

0. まえがき

Salomaaによつて regular expression の同値性に対する無矛盾で、かつ完全な公理系がつけられた。この完全性の証明において、 $\forall \alpha, \beta, \alpha = \beta$ がこの公理系の定理であるか否かを判定するアルゴリズムが示されている[1]。

我々は公理系を計算機で扱う一つの例として、このアルゴリズムに着目し regular expression の同値性を判定するプログラム compare について述べる。我々は compare を LISP言語[2], [3]で表現し、HITAC 8350 によつて実行した[4]。我々は Salomaa が示したアルゴリズムにしたがうが、やや異なるものを用いたので compare においては regular expression α に対して α の方程式系をつくる(例を受理する有限オートマトンをつくる)ための独立なサブプログラムを用意してゐない。

したが、例を受理するオートマトンをつくるプログラム

am についても簡単に述べる. am は LISP 言語との比較のため SNOBOL3 言語[5]で表現し, HEWLETT PACKARD 2100 で実行したもののについて示す.

またスタックを用いて同様のプログラムを表現する方法についても述べる.

1. 準備

$\Sigma = \{A, B, \dots, Z\}$ とする. Σ 上の regular expression (以下 re と略記する) を次のように定義する.

- (1) $0 \notin \Sigma$ は re である.
- (2) $a \in \Sigma$ ならば a は re である.
- (3) α, β が共に re ならば $(\alpha \cdot \beta), (\alpha + \beta), (\alpha^*)$ はそれぞれ re である. この第3のものは $(,)$ でくく, 1 がある点が Salomaa のものとは異なる.

例 1.1 $A, (0^*), (A + ((B \cdot C)^*))$ はそれぞれ re である.

Salomaa の公理系は文献[1]にあるが再掲すると次のようになる. axiom schema $A1 \sim A11$ と推論規則 $R1, R2$ より構成される.

- | | | | |
|----|--|-----|--|
| A1 | $(\alpha + (\beta + \gamma)) = ((\alpha + \beta) + \gamma)$ | A7 | $((0^*) \cdot \alpha) = \alpha$ |
| A2 | $(\alpha \cdot (\beta \cdot \gamma)) = ((\alpha \cdot \beta) \cdot \gamma)$ | A8 | $(0 \cdot \alpha) = 0$ |
| A3 | $(\alpha + \beta) = (\beta + \alpha)$ | A9 | $(\alpha + 0) = \alpha$ |
| A4 | $(\alpha \cdot (\beta + \gamma)) = ((\alpha \cdot \beta) + (\alpha \cdot \gamma))$ | A10 | $(\alpha^*) = ((\alpha \cdot (\alpha^*)) + (0^*))$ |

$$A5 \quad ((\alpha + \beta) \cdot r) = ((\alpha \cdot r) + (\beta \cdot r)) \quad A11 \quad (\alpha^*) = ((\alpha + (0^*))^*)$$

$$A6 \quad (\alpha + \alpha) = \alpha$$

R1: 代入 R2: 方程式の解

(A10, A11 は Salomaa のものと異なる.)

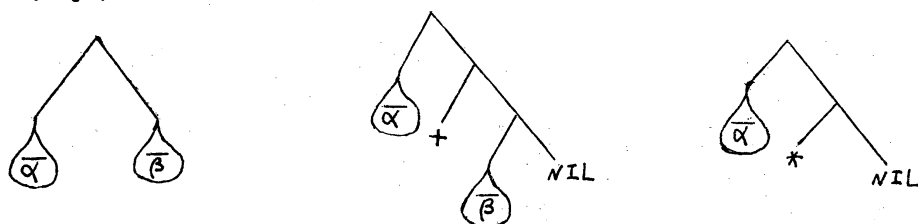
re α は次の形をしているとき normal form をしていると言い、normal form をしている re のことを normal re (以下 nre と略記する) と言う。

$\alpha \equiv (\alpha_n + \dots + (\alpha_1 + \alpha_0) \dots)$, $n \geq 0$, $\alpha_0 \equiv 0$ 又は $\alpha_0 \equiv (0^*)$, $1 \leq i \leq n$ に対して $\alpha_i \equiv (a_i \cdot \beta_i)$ 又は $\alpha_i \equiv a_i$ $a_i \in \Sigma$, β_i は re, $\equiv$ で α_i を linear term, a_i をその handle, また $\alpha_i \equiv (a_i \cdot \beta_i)$ に対して β_i をその derivative, $\alpha_i \equiv a_i$ に対しては (0^*) をその derivative とする。 $n \geq 2$ ならば $a_n, \dots, a_1$ は重複がなくかつアルファベット順にならなければならない。このことを sort されていると言う。

例 1.2 (0^*) , $(A + 0)$, $(A + ((C \cdot (B^*)) + ((D \cdot (A + B)) + 0)))$ はそれぞれ nre で (0^*) の linear term は存在しない。 $(A + 0)$ の linear term は A でその handle は A , derivative は (0^*) である。第 3 番目の nre の linear term は A , $(C \cdot (B^*))$, $(D \cdot (A + B))$ で, A , C , D がそれぞれの handle, (0^*) , (B^*) , $(A + B)$ がそれぞれの derivative である。

re の LISP における内部表現について述べる。 re α の内部表現を $\overline{\alpha}$ とかく, $\overline{\alpha}$ は re α を S 式とみなした値とする。(ただし

+) の前後と * の前には空白を入れるものとする。) したが、
 α, β に対して $\overline{(\alpha \cdot \beta)}$, $\overline{(\alpha + \beta)}$, $\overline{(\alpha *)}$ を図式表現すると次のよ
うになる.



α に対しては処理能率を向上させるために特別の内部表
現 $\tilde{\alpha}$ を用いる. $\alpha \equiv (\alpha_n + \dots + (\alpha_1 + \alpha_0) \dots)$ $n \geq 0$ に対して
 $\tilde{\alpha}$ を次のようにする. $\tilde{\alpha} = ((\overline{\alpha_n} \sqcup \dots \sqcup \overline{\alpha_1}) \cdot \tilde{\alpha}_0)$

$$\tilde{\alpha}_0 = \begin{cases} 0 & : \alpha_0 \equiv 0 \text{ のとき} \\ 1 & : \alpha_0 \equiv (0 *) \text{ のとき} \end{cases}$$

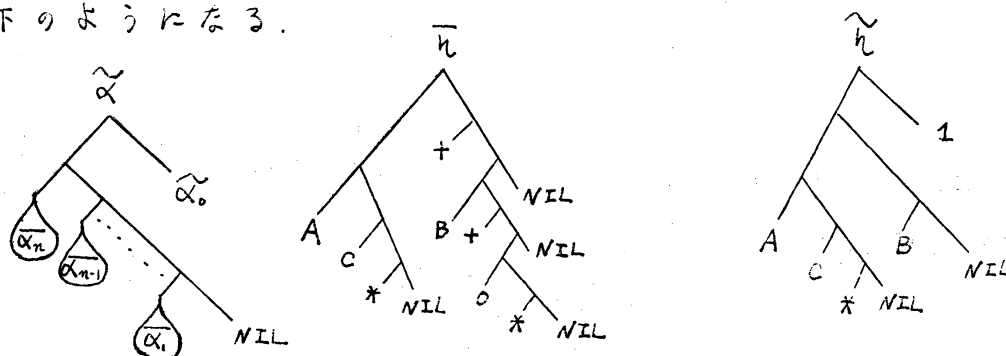
$\text{car}[\tilde{\alpha}] = (\overline{\alpha_n} \sqcup \dots \sqcup \overline{\alpha_1})$ を $\tilde{\alpha}$ の linear part, $\text{cdr}[\tilde{\alpha}] = \tilde{\alpha}_0$ を $\tilde{\alpha}$ の
constant part と言う. $\tilde{\alpha}$ の図式表現は下の左図のようになる.

例 1.3 $\eta \equiv ((A \cdot (C *)) + (B + (0 *)))$ のとき

$$\overline{\eta} = ((A \cdot (C \sqcup *)) \sqcup + \sqcup (B \sqcup + \sqcup (0 \sqcup *)))$$

$$\tilde{\eta} = (((A \cdot (C \sqcup *)) \sqcup B) \cdot 1) \quad \text{でこれらは図式表現する}$$

と下のようになる.



2. プログラム compare

Salomaa は前節に示した公理系の完全性の証明のなかで次のことを示している [1].

- (1) 任意の $re\ \alpha$ に対して $\vdash \alpha = \beta$ なる $re\ \beta$ が存在する (Lemma 4)
- (2) 任意の $re\ \alpha$ は方程式系を有す. (Lemma 4)
- (3) 任意の 2 つの $re\ \alpha, \beta$ の同値性は決定可能である. (Theorem 2)

この事実にもとづいて任意の $re\ \alpha, \beta$ に対して $\vdash \alpha = \beta$ (すなわち α と β が同値) であるか否かを判定するプログラム compare をつくる. Salomaa は上の (1), (2) をアルゴリズムを示すことで同時に証明した. 我々が compare において用いたアルゴリズムはこれとはいくらの異なる. プログラムは LISP の M 式によつて示す. S 式で表現したプログラムは付録 1 に示す.

式の読みやすさのために $[[,]], [,]$ を $[,]$ と同じ意味に用いることにする. 以下説明のない関数 (述語) はすべて LISP システムの基本関数又は組み込み関数である. ただし eg1 は, 2 つの S 式が等しいことを判定する述語で我々が定義したものである.

- (1) handle 任意の linear term に対してその handle を対応させる関数

$handle[x] = [atom[x] \rightarrow x; T \rightarrow car[x]]$

例 2.1 $handle[A] = A$, $handle[(A.(B.+C))] = A$

(2) drv (derivative) linear term に対してその derivative を対応させる関数

drv[x]=[atom[x]→list[0,*];T→cdr[x]]

例 2.2 drv[A]=(0,*) , drv[(A.(B,+.C))]=(B,+.C)

(3) order linear part $(\overline{\alpha}_n, \dots, \overline{\alpha}_1)$ に対して $\overline{\alpha}_n$ の handle の alphabetical order を対応させる関数. linear part = () のときは order[()]=27 とする.

order[x]=[null[x]→27;T→cdr[assoc[handle[car[x]];((A.1) ... (Z.26))]]]

例 2.3 order[(B)]=2 , order[((A.(B,*))_.B.C)]=1

(4) member x が リスト の 要素 である かどうか を 判定 する 述語

member[x;y]=[null[y]→NIL;eq1[x;car[y]]→T;T→member[x;cdr[y]]]

例 2.4 member[A;(B,A)]=T , member[(A);(B,A)]=NIL

(5) form 式 を その 形 に よって 分類 する 関数 で 次 を 示す.

form[x]=
 0 : x=0
 1 : $\exists a[x=a \wedge a \in E]$
 2 : $\exists \alpha[x=(\alpha,*)]$
 3 : $\exists \alpha, \exists \beta[x=(\alpha,+. \beta)]$
 4 : $\exists \alpha, \exists \beta[x=(\alpha, \beta) \wedge \sim \exists \gamma[\beta=(*, \gamma) \vee \beta=(+, \gamma)]]$
 NIL : otherwise

form[x]=[atom[x]→[eq[x;0]→0;member[x;(A,...,Z)]→1;T→NIL];

atom[cdr[x]]→4;

eq1[cadr[x];*]→[null[cddr[x]]→2;T→NIL];

eq1[cadr[x];+]→[atom[cddr[x]]→NIL;T→[null[cdddr[x]]→3;T→NIL]];

T → 4]

例 2.5 form[*]=NIL , form[(A)]=4 , form[((A,+.B),*)]=2

(6) elementary $\alpha = \overline{\alpha}$ に対して α が $(\beta + \gamma)$ なる形をしていることを示す述語

elementary[x]=not[eq[form[x];3]]

例 2.6 elementary[(A.₊.(B.₊*))]=NIL, elementary[((A.₊.(B.₊*)).*)]=T

(7) mergel

$$\text{mergel}[(\overline{\alpha_n + \dots + (\alpha_i + \alpha_1) \dots}); \overline{\beta}] = \begin{cases} \overline{(\alpha_n + \dots + (\alpha_i + (\alpha_1 + \beta)) \dots)} & : (\forall i) [1 \leq i \leq n \rightarrow \alpha_i \neq \beta] \\ \overline{(\alpha_n + \dots + (\alpha_i + \alpha_1) \dots)} & : (\exists i) [1 \leq i \leq n \rightarrow \alpha_i \equiv \beta] \end{cases}$$

mergel[x;y]=[elementary[x]→eq1[x;y]→x;T→list[x;+;y]];

T→[eq1[car[x];y]→x;T→list[car[x];+;mergel[caddr[x];y]]]

例 2.7 mergel[(B+C);(A+(B+C))]=(B+(C+(A+(B+C))))

mergel[(A+((A*)+B));(A*)]=(A+((A*)+B))

(8) merge $x = \overline{(\alpha_n + \dots + (\alpha_i + \alpha_1) \dots)}$, $y = \overline{(\beta_n + \dots + (\beta_i + \beta_1) \dots)}$ $m, n \geq 1$ に対して

$\overline{(\alpha_m + \dots + (\alpha_1 + (\beta_{i_k} + \dots + (\beta_{i_1} + \beta_1) \dots)) \dots)}$ $i_k \geq 0, i_k > \dots i_2 > i_1$ を対応させる関数.

これは次の(9)の中で用いられ, compareの停止性に対して本質的な役わりをもっている. これについては次節で述べる.

merge[x;y]=[elementary[y]→mergel[x;y];

T→merge[mergel[x;car[y]];caddr[y]]]

例 2.8 merge[(B+C);(A+(B+C))]=(B+(C+A))

(9) add1 (add linear parts) 2つのそれぞれ sort されている linear part に対して1つの sort された linear part を対応させる関数.

add1[x;y]=[lessp[order[x];order[y]]→cons[car[x];add1[cdr[x];y]];

greaterp[order[x];order[y]]→cons[car[y];add1[x;cdr[y]]];

T→[null[x]→NIL;T→cons[cons[handle[car[x]];merge[drv[car[x]];

drv[car[y]]];add1[cdr[x];cdr[y]]]]]

例 2.9 add1[(A);()]=(A)

add1[(A.(C.(A.₊.(B.₊)))];((B.A).(C.((A.₊*)₊.(B.₊))))]

=(A.(B.A).(C.(A.₊.(B.₊.(A.₊*)₊))))

- (10) addc (add constant parts) 2つの constant parts に 対して 1つの constant part を 対応させる関数.

addc[x;y]=[eq[x;0]→y;T→1]

addc, addl は nre α , $\tilde{\alpha} = ((\overline{\alpha}_1 \cup \dots \cup \overline{\alpha}_n) \cdot \tilde{\alpha}_0)$ と nre β , $\tilde{\beta} = ((\overline{\beta}_1 \cup \dots \cup \overline{\beta}_m) \cdot \tilde{\beta}_0)$ から $(\alpha + \beta)$ と同値な nre γ , $\tilde{\gamma} = ((\overline{\gamma}_1 \cup \dots \cup \overline{\gamma}_n) \cdot \tilde{\gamma}_0)$, $n \leq l+m$, $\tilde{\gamma}_0 = \text{addc}[\tilde{\alpha}_0; \tilde{\beta}_0]$ をもてめるのに用いられる.

- (11) dtbt (distribute) 1つの sort されている linear part $(\overline{\alpha}_1 \cup \dots \cup \overline{\alpha}_n)$ と re β に 対して sort された linear part $(\overline{\gamma}_1 \cup \dots \cup \overline{\gamma}_n)$ を 対応させる関数 $\gamma = ((\alpha_1 + \dots + (\alpha_2 + \alpha_1) \dots) \cdot \beta)$ なる形の re からこれと同値な re $(\gamma_1 + \dots + (\gamma_2 + \gamma_1) \dots)$.

$$\gamma_i \equiv \begin{cases} (a_i \cdot \beta) & : \alpha_i \equiv a_i \\ (a_i \cdot (\gamma_i \cdot \beta)) & : \alpha_i \equiv (a_i \cdot \gamma_i) \end{cases} \text{ を もてめるのに用いられる.}$$

dtbt[x;y]=[null[x]→NIL;

atom[car[x]]→cons[cons[car[x];y];dtbt[cdr[x];y]];

T→cons[cons[caar[x];cons[cdr[x];y]];dtbt[cdr[x];y]]]

例 2.10 dtbt[(A.(B.C));(A.*)]=((A.(A.*)).(B.(C.(A.*)))) これは

((A+(B.C)).(A*))より((A.(A.*))+(B.(C.(A.*)))) をもてめることに 相当する.

- (12) nmlz 標準化関数といひ, 任意の α に対して $\vdash \alpha = \beta$, β は nre なる β を 対応させる関数 $\vdash$ compare の中で用いられる重要な関数である. nmlz[x] を x の形で場合わけして定義すると次のようになる.

- 0) $x=0$ するときは form[x]=0 のとき

nmlz[0]=(().0)

- 1) $x=a, a \in \Sigma$ するときは form[x]=1 のとき

nmlz[a]=((a).0)

- 2) $x=(\overline{\alpha}^*)$ するときは form[x]=2 のとき

nmlz[($\overline{\alpha}^*$)]=cons[dtbt[car[nmlz[$\overline{\alpha}$]];($\overline{\alpha}^*$)];1] (i)

$nm|z[\bar{\alpha}] = \tilde{\mu}$, $\vdash \alpha = \mu$, μ は nre かつ $\mu \equiv (\mu_m + \dots + (\mu_1 + \mu_0) \dots)$ $m \geq 0$ なる仮定のもとで (i) 式の右辺は, $\vdash (\alpha *) = \tau$, τ は nre, なる $\tilde{\tau}$ があることを示そう.

$\mu_0 \equiv 0$ なる A9 により $\mu_0 \equiv (0 *)$ なる A11 により

$$\begin{aligned} \vdash (\alpha *) &= ((\mu_m + \dots + (\mu_2 + \mu_1) \dots) *) \\ &= (((\mu_m + \dots + (\mu_2 + \mu_1) \dots) \cdot ((\mu_m + \dots + (\mu_2 + \mu_1) \dots) *)) + (0 *)) \quad [\because A10] \\ &= (((\mu_m + \dots + (\mu_2 + \mu_1) \dots) \cdot (\alpha *)) + (0 *)) \\ &= (\tau_m + \dots + (\tau_1 + (0 *)) \dots) \end{aligned}$$

$$\text{したがって } \tau_i \equiv \begin{cases} (a_i \cdot (\alpha *)) & : \mu_i \equiv a_i \text{ のとき} \\ (a_i \cdot (\beta_i \cdot (\alpha *))) & : \mu_i \equiv (a_i \cdot \beta_i) \text{ のとき} \end{cases}$$

- $\bar{\tau} = \text{car}[nm|z[\bar{\alpha}]] = (\bar{\tau}_m \cup \dots \cup \bar{\tau}_1)$, したがって left の機能により (i) 式の右辺は $((\bar{\tau}_m \cup \dots \cup \bar{\tau}_1) \cdot 1)$ でありこれは $\tilde{\tau}$ にほかならない.

3) $x = \overline{(\alpha + \beta)}$ となる $\text{form}[x] = 3$ のとき

$$nm|z[\overline{(\alpha + \beta)}] = \text{cons}[\text{addl}[\text{car}[nm|z[\bar{\alpha}]]]; \text{car}[nm|z[\bar{\beta}]]];$$

$$\text{addc}[\text{cdr}[nm|z[\bar{\alpha}]]]; \text{cdr}[nm|z[\bar{\beta}]]] \dots \dots \dots (ii)$$

この場合には $nm|z[\bar{\alpha}]$, $nm|z[\bar{\beta}]$ をもとめて, addl を用いて 2つの linear parts より $\overline{(\alpha + \beta)}$ の linear part をつくり, addc を用いて 2つの constant parts より $\overline{(\alpha + \beta)}$ の constant part をつくり, それらを cons でつなげる. $nm|z[\bar{\alpha}] = \tilde{\mu}$, $nm|z[\bar{\beta}] = \tilde{\nu}$, $\vdash \alpha = \mu$, $\vdash \beta = \nu$, μ, ν は nre, なる仮定のもとで (ii) 式の右辺は, $\vdash (\alpha + \beta) = \tau$, τ は nre, なる $\tilde{\tau}$ であることは addl , addc の機能より明らか.

4) $x = \overline{(\alpha \cdot \beta)}$ となる $\text{form}[x] = 4$ のとき. このときは $\text{cdr}[nm|z[\bar{\alpha}]]$ となる $nm|z[\bar{\alpha}]$ の constant part の 0, 1 によって更に場合わけする.

4.0) $\text{cdr}[nm|z[\bar{\alpha}]] = 0$ のとき

$$nm|z[\overline{(\alpha \cdot \beta)}] = \text{cons}[\text{dtbt}[\text{car}[nm|z[\bar{\alpha}]]]; \bar{\beta}]; 0]$$

4.1) $\text{cdr}[nm|z[\bar{\alpha}]] = 1$ のとき

$$nm|z[\overline{(\alpha \cdot \beta)}] = \text{cons}[\text{addl}[\text{dtbt}[\text{car}[nm|z[\bar{\alpha}]]]; \bar{\beta}]; \text{car}[nm|z[\bar{\beta}]]]; \text{cdr}[nm|z[\bar{\beta}]]] \dots \dots (iii)$$

$nm|z[\bar{\alpha}] = \tilde{\mu}$, $\vdash \alpha = \mu$, $\mu \equiv (\mu_m + \dots + (\mu_1 + (0 *)) \dots)$ $nm|z[\bar{\beta}] = \tilde{\nu}$, $\vdash \beta = \nu$

$\nu \equiv (\nu_n + \dots + (\nu_1 + \nu_0) \dots)$ なる仮定のもとで (iii) 式の右辺は, $\vdash (\alpha \cdot \beta) = \tau$, τ は nre, なる $\tilde{\tau}$ であることは addl , dtbt の機能と次の事実よりわかる.

$$\begin{aligned} \vdash (\alpha \cdot \beta) &= ((\mu_m + \dots + (\mu_1 + (0 *)) \dots) \cdot \beta) \\ &= (((\mu_m + \dots + (\mu_2 + \mu_1) \dots) \cdot \beta) + ((0 *) \cdot \beta)) \\ &= (((\mu_m + \dots + (\mu_2 + \mu_1) \dots) \cdot \beta) + (\nu_m + \dots + (\nu_1 + \nu_0) \dots)) \end{aligned}$$

実際のプログラムは効率の点からLISPのprog形式を用いて次のように定義する。また $\text{form}[x] = \text{NIL}$ のときは $\text{nmlz}[x] = \text{NIL}$ とする。

```

nmlz[x]=prog[(alpha beta frm);
  [null[setq[frm;form[x]]]→return[NIL]];
  [eq[frm;0]→return[cons[NIL;0]]];
  [eq[frm;1]→return[cons[list[x];0]]];
  [null[setq[alpha;nmlz[car[x]]]→return[NIL]];
  [eq[frm;2]→return[cons[dtbt[car[alpha];x];1]]];
  [eq[frm;3]→[null[setq[beta;nmlz[caddr[x]]]→return[NIL];
    T→return[cons[add1[car[alpha];car[beta]];addc[cdr[alpha];cdr[beta]]]]];
  [eq[cdr[alpha];0]→return[cons[dtbt[car[alpha];cdr[x]];0]]];
  [null[setq[beta;nmlz[cdr[x]]]→return[NIL]];
  return[cons[add1[dtbt[car[alpha];cdr[x]];car[beta]];cdr[beta]]] ]

```

例は付録1で示す。

(13) compare 比較関数といひ nmlz を利用して、2つの α, β に対して $\alpha = \beta$ (すなわち α と β は同値) か否かを決定する関数である。はじめに次の2つの関数を定義する。

1) nconcl リスト x の最後に y をつけ加えてできるリストを値とする関数で

$\text{nconcl}[(S_1, \dots, S_n); S] = (S_1, \dots, S_n, S)$ $n \geq 0$ をみたす。

$\text{nconcl}[x; y] = \text{nconc}[x; \text{list}[y]]$

2) drvpair 2つのlinear part $(\bar{\alpha}_1, \dots, \bar{\alpha}_n)$, $(\bar{\beta}_1, \dots, \bar{\beta}_m)$ に対して derivative のベクトリスト

$(\bar{P}_1, \dots, \bar{P}_n)$ $n = \max[1; m]$ を対応させる関数

例 2.11 $\text{drvpair}[(A, (B, \bar{\beta}), (C, \bar{\gamma}))]; ((A, \bar{\alpha}), (C, \bar{\gamma}))]$

$= ((0, *) \cdot \bar{\alpha}) (\bar{\beta}, 0) (\bar{\gamma}, \bar{\gamma})$

```

drvpair[x;y]=[lessp[order[x];order[y]]+cons[cons[drv[car[x];0];drvpair[cdr[x];y]]];
greaterp[order[x];order[y]]+cons[cons[0;drv[car[y]]];drvpair[x;cdr[y]]];
null[x] → NIL;
T→cons[cons[drv[car[x]];drv[car[y]]];drvpair[cdr[x];cdr[y]]]

```

re α, β に対して compare は次のうごきをする。

- (A) $u1$ を $((\bar{\alpha} \cdot \bar{\beta}))$, $u2$ を $()$ として (B) へすすむ
- (B) $nr1$ を $nm[z[car[u1]]]$, $nr2$ を $nm[z[cdr[u1]]]$ として, $nr1$ と $nr2$ の constant part が等しいか否かをみる。等しくなければ $\neg \alpha = \beta$ であり処理を終える。等しければ $u2$ を $nconc1[u2; car[u1]]$, $u1$ を $cdr[u1]$, pairlist を $drvpair[car[nr1]; car[nr2]]$ として (C) へすすむ
- (C) pairlist が $()$ なら (D) へすすむ, $()$ でなければ $apair$ を $car[pairlist]$ とし, pairlist を $cdr[pairlist]$ とし, $apair$ と等しい要素が $u1$ または $u2$ の中にあれば (C) へすすむ, なければ $u1$ を $nconc1[u1; apair]$ として (C) へすすむ。
- (D) $u1$ が $()$ ならば $\alpha = \beta$ であり, 処理を終える。 $()$ でなければ (B) へすすむ。

compare に用いたアルゴリズムは Salomaa [1] に従ったが、次の点が異なる。

- 1) Salomaa では α, β についてそれぞれ方程式系をつくる。そのつくり方は re α に対してその sub re のなかでもっと小さなものについて方程式系をつくり順次より大きな sub re に対する方程式系をつくり最終的に α に対する方程式系をつくる。そして二つの方程式系を比較して $\alpha = \beta$ の validity に対する矛盾があるか否かをみる。一方 compare では上に述べたように, α, β のヘッダに対してそれぞれ同値な nre のヘッダをもとめ, その constant part を比較して validity に対する矛盾がなければ derivative のヘッダ達に対してまた nre のヘッダをもとめていなりものについて順次 nre のヘッダをもとめ constant part を比較していく。したがって compare では α と β の方程式系をつくる(間接的であるが)ことと二つの方程式系を比較することを全く並行して行う。
- 2) Salomaa では各方程式の右辺にはアルファベットのすべての元が handle として現われる。compare においてはアルファベットの元のうちで必要なものだけが handle として現われる。

例 2.12

 $\alpha \equiv ((A \cdot (B *)) + C)$ に対して Salomaa では $\vdash \alpha = ((A \cdot (B *)) + ((B \cdot 0) + ((C \cdot (0 *)) + ((0 \cdot 0) + \dots + ((Z \cdot 0) + 0) \dots)))$

を方程式とみなし compare では

 $\vdash \alpha = ((A \cdot (B *)) + (C + 0))$ を方程式とみなす.

$$\text{compare}[x;y] = \begin{cases} ((\bar{\alpha}, \bar{\beta}) (\bar{\alpha}_1, \bar{\beta}_1) \dots (\bar{\alpha}_n, \bar{\beta}_n)) : x=\bar{\alpha}, y=\bar{\beta}, \vdash \alpha=\beta \\ \text{NOT-EQUIVALENT} : x=\bar{\alpha}, y=\bar{\beta}, \not\vdash \alpha=\beta \\ \text{SYNTAX-ERROR} : X \text{ と } Y \text{ が re の表現でない} \end{cases}$$

ただし $(\bar{\alpha}_1, \bar{\beta}_1), \dots, (\bar{\alpha}_n, \bar{\beta}_n)$ derivative のへず

となるように compare を prog 形式で定義すると次のようになる.

```
compare[x;y]=prog[(ul u2 nr1 nr2 pairlist apair);
 setq[ul;list[cons[x;y]]];
COMP1 [or[null[setq[nr1;nmlz[caar[ul]]]]
  ;null[setq[nr2;nmlz[cdar[ul]]]]]→return[SYNTAX-ERROR]];
[not[eq[cdr[nr1];cdr[nr2]]]→return[NOT-EQUIVALENT]];
setq[u2;nconcl[u2;car[ul]]];
setq[ul;car[ul]];
setq[pairlist;drvpair[car[nr1];car[nr2]]];
COMP2 [null[pairlist]→go[COMP3]];
setq[apair;car[pairlist]];
setq[pairlist;cdr[pairlist]];
[or[member[apair;ul];member[apair;u2]]→go[COMP2]];
setq[ul;nconcl[ul;apair]];
go[COMP2];
COMP3 [null[ul]→return[u2]];
go[COMP1] ]
```

3. compare の停止性

まずプログラム compare の中で用いられる各サブプログラムの停止性が問題になる。しかしこれらのほとんどのものの停止性は自明である。nm12 はやや複雑な形をしており、その停止性は、正当性も含めて、 $re\ \alpha$ 中に現われる演算子 $(\cdot, +, *)$ の個数に関する帰納法によ、て証明される。その証明の一部は nm12 の説明のところで示した。

次に compare の停止性は $re\ \alpha$ の相異なる derivative の個数が有限 (α に対する方程式系が存在する) か否かによる。derivative が有限個になることの証明は Salomaa [1] の Lemma 4 における証明と並行して、やはり α 中の演算子の個数に関する帰納法によ、てできるが次のことに注意しなくてはならない。

derivative $(\alpha_1 + (\alpha_2 + \dots + (\alpha_{n-1} + \alpha_n) \dots))$ nm12 におりて $\alpha_1, \dots, \alpha_n$ の中に相等しいものがあるかを調べて derivative をそれと同値な $(\alpha_{i_1} + \dots + (\alpha_{i_{m-1}} + \alpha_{i_m}) \dots)$, $i_m \geq 1$, $i_1 < i_2 < \dots < i_m \leq n$, $\alpha_{i_1}, \dots, \alpha_{i_m}$ は相異なる。に変形しなくてはならない。(merge はこれを行なう) これは $(\alpha \cdot \beta)$, $(\alpha *)$ なる形の re の相異なる derivative の個数を有限に抑える上で必要である。merge を単に

$$\text{merge}[x; y] = \text{list}[x; +; y]$$

とすると上で述べたことを行なうなりので、 $((A + ((A \cdot B) + (B \cdot A)))^*)$, $((A^*) \cdot (A^*))$ 等の derivative の個数は有限にならなり。例えば

$((A^*) \cdot (A^*))$ に対しては $((A^*) \cdot (A^*))$, $((((A^*) \cdot (A^*)) + (A^*)))$,
 $((((A^*) \cdot (A^*)) + ((A^*) + (A^*))))$, ... と無限に derivative が現われる.

4. 有限オートマトンの生成

この節では $re \alpha$ に対して α を受理する有限オートマトンを生成するプログラムについて簡単に述べる. プログラムは LISP との比較のために SNOBOL3 で表現した[5]. 全プログラムと実行例は付録2に示す.

$re \alpha$ に対して α の方程式系; $\vdash \alpha_i = \beta_i, i=1, \dots, n, \alpha_i \equiv \alpha, \text{ 各 } \beta_i$ は nre で β_i の右 derivative γ_{ij} に対して $\gamma_{ij} \equiv \alpha_k, 1 \leq k \leq n$ なる k が存在する; は α を受理する有限オートマトンの構造を完全に表現している. この事実にもとづいてプログラムを SNOBOL3 で表現する. SNOBOL3 と LISP の主な違いはその扱うデータ構造である. LISP は binary tree 構造をもつデータを扱うのに対して SNOBOL3 では記号列そのものをデータとして扱う. recursive call は LISP と同様 SNOBOL3 にも許されている.

我々はこの節では, re に対して Salomaa のものと同じ定義を採用する. ((α^*) ではなく α^* を用いる.) $re \alpha$ に対して $\bar{\alpha}$ は α のものとし, $nre \alpha \equiv (\alpha_n + \dots + (\alpha_1 + \alpha_0) \dots)$ に対して $\tilde{\alpha}$ は次のものとする. $\tilde{\alpha} = ((\alpha_n + \dots + (\alpha_1 + NIL) \dots) + \alpha_0)$ この様に定義しておくとも LISP のときと全く同様にして SNOBOL3 で nml が

表現できる. SNOBOL 3 で表現した nm/z の中で用いられるサブプログラムプログラムのプログラム名は LISP でのそれらと多少異なる, たので, 機能別にそれらの対応を下に示す.

SNOBOL 3	LISP
left	$\text{car} \{x x = \bar{\alpha}\}$
right	$\text{cdr} \{x x = \overline{(\alpha \cdot \beta)}\}$ 又は $\text{caddr} \{x x = \overline{(\alpha + \beta)}\}$
oper	$\text{form} \{x x = \overline{(\alpha * \beta)} \vee x = \overline{(\alpha \cdot \beta)} \vee x = \overline{(\alpha + \beta)}\}$
handle	handle
drv	drv
order	order
cona	merge1
con	merge
mrq	addl
addcnst	addc
dister	dtbt

この表で同じ行のプログラムは同じ機能がある.

$re \alpha$ に対して $|x|$ を受理する有限オートマトンを生成するプログラム am は nm/z を用いて α に対する方程式系をつくることを行う. 出力する方程式系をみやすくする為に右方程式の derivative を S_i (i は自然数) で置き換え左ものを出力するようにしてある. 実行例として $\alpha \equiv (((B \cdot A) + (A^* \cdot (A \cdot B)))^* \cdot A^*)$ と

$\beta \equiv (A + ((A \cdot B) + (B \cdot A)))^*$ にッリテ付録2で示した.

5. スタックを用いて標準化関数 nm/z をつくる方法

この節では recursive call を使わずに nm/z をつくる1例を簡単に述べる. ここでも re の定義は Salomaa のものと同じものとする.

re のものが入力されるとし, $nre \alpha \equiv (\alpha_n + \dots + (\alpha_1 + \alpha_0) \dots)$
 $n \geq 0$ に対する $\tilde{\alpha}$ は次のようにする.

$$\begin{aligned}\tilde{\alpha} &= \tilde{\alpha}_n + \dots + \tilde{\alpha}_1 + \tilde{\alpha}_0 \\ \tilde{\alpha}_i &= \begin{cases} a & : \alpha_i \equiv a \text{ のとき} \\ a\beta & : \alpha_i \equiv (a \cdot \beta) \text{ のとき} \end{cases} \quad (1 \leq i \leq n) \\ \tilde{\alpha}_0 &= \begin{cases} 0 & : \alpha_0 \equiv 0 \text{ のとき} \\ 1 & : \alpha_0 \equiv 0^* \text{ のとき} \end{cases}\end{aligned}$$

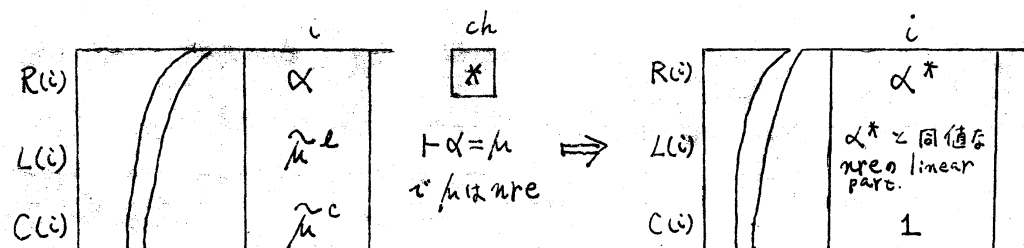
また $\tilde{\alpha}^L = \tilde{\alpha}_n + \dots + \tilde{\alpha}_1 +$ を $\tilde{\alpha}$ の linear part, $\tilde{\alpha}^C = \tilde{\alpha}_0$ を $\tilde{\alpha}$ の constant part という. $re \alpha$ と同値な nre を逐次的にもとめるために3種のスタック R, L, C を用いる. $R(i)$ には与えられた $re \alpha$ の sub $re \beta$ が (か + か $\cdot$ をしきう). $L(i)$ には $\vdash \beta = \gamma$, γ は nre なる $\tilde{\gamma}^L$, $C(i)$ には $\tilde{\gamma}^C$ をしきう.

例 5.1 $((B+A) \cdot C^*)$ を処理しているときのスタックの状態は次のとおりである.

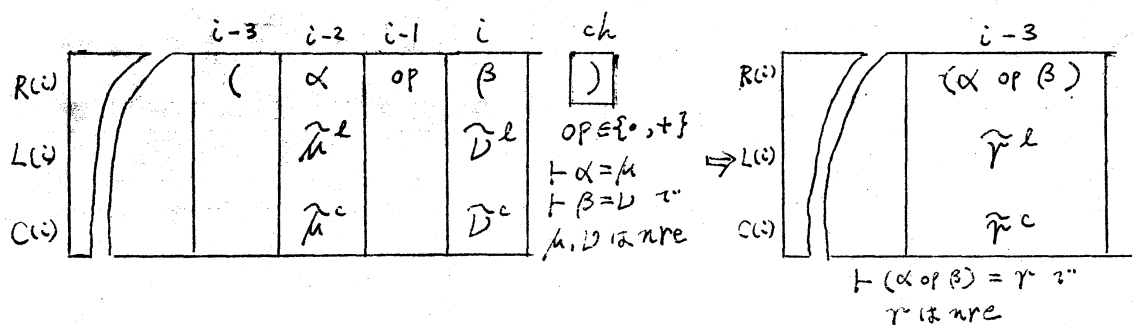
$i \backslash$	1	2	3	4	
$R(i)$	(	$(B+A)$	$\cdot$	C	
$L(i)$		$A+B+$		$C+$	
$C(i)$		0		0	

以下にスタックのうごきを明確化するために LISP 型言語に類似な架空の言語でプログラムを示すが、条件式, if , $return$, $::$ は LISP の $prog$ 形式におけるそれらと同じ意味に使う。また $addl$, $addc$, $dtbtt$ は第2節のそれらと同様の機能とする。0 は string の concatenation を意味する。

プログラムにおいて NMLZ4 の次のステートメント以後のそれらはスタックの状態を次のようにかえる。



また NMLZ5 以後のステートメントでは次のようになる。



```

nmlz[x]=[ i:=0 ;
  NMLZ1  [xは空 → return[ERROR]];
          ch:= xの左端1字 ;
          x:= xより左端1字をのぞいたもの ;
          i:= i+1 ; R(i):=ch ;
          [ch=( → go[NMLZ1]];
          [ch=0 → go[NMLZ2]];
          [chは空でない → return[ERROR]];
          L(i):=ch+ ;
  NMLZ2  C(i):=0 ;
  NMLZ3  [xは空 → [i=1 → return[L(1)◦C(1)];T→return[ERROR]];
          ch:= xの左端1字 ;
          x:= xより左端の1字をのぞいたもの ;
          [ch≠.かつch≠+ → go[NMLZ4]];
          R(i):=ch ; go[NMLZ1];

  NMLZ4  [ch≠* → go[NMLZ5]];
          L(i):=dtbt[L(i);R(i)]; C(i):=1; R(i):=R(i)◦* ; go[NMLZ3] ;

  NMLZ5  [ch≠)又はi<4又はR(i-3)≠( → return[ERROR]];
          [R(i-1)=+ → go[NMLZ7]];
          [R(i-1)≠. → return[ERROR]];
          L(i-3):=dtbt[L(i-2);R(i)];
          [C(i-2)=1 → go[NMLZ8]];
          C(i-3):=0 ;
  NMLZ6  R(i-3):=(◦R(i-2)◦R(i-1)◦R(i)◦) ; i:=i-3 ; go[NMLZ3];

  NMLZ7  L(i-3):=add1[L(i-2);L(i)]; C(i-3):=addc[C(i-2);C(i)]; go[NMLZ6];

  NMLZ8  L(i-3):=add1[L(i-3);L(i)]; C(i-3):=C(i); go[NMLZ6] ]

```

謝辞

LISPシステムに關してお世話になった通産省、電子技術総合研究所の田村浩一郎さんに感謝します。

文献

- [1] Salomaa, A., Two Complete Axiom System for the Algebra of Regular Events, JACM, vol. 13, No. 1, PP. 158-169, 1966
- [2] McCarthy, J., et al., LISP 1.5 Programmer's Manual, MIT Press, Cambridge, Mass., 1962
- [3] 日立製作所, HITAC 8350 LISP 仕様書, 1972
- [4] 日立製作所, HITAC 8350 LISP 操作仕様書, 1972
- [5] Harber, D.J., et al., The SNOBOL 3 Programming Language, The Bell System Technical J., July-August, 1966

付録1

199

HITAC R000 SERIES ***FDS*** *JOB CONTROL*

DATE 02/15/74 TIME 17:55:25

*JOB CLASS B *JOB NAME ZYSDR *JOB# 0560 *MEMORY 9/KH

RESERVED DEVICE C0

// VDC SYSUT3,EDUS.SYSUT3A,EDUS11

// EXEC LISP

#PACKET

C EQUIVALENCE PRUVER FOR REGULAR EXPRESSIONS

REX00000

REX00010

REX00020

REX00030

REX00040

REX00050

REX00060

REX00070

REX00080

REX00090

REX00100

REX00110

REX00120

REX00130

REX00140

REX00150

REX00160

REX00170

REX00180

REX00190

REX00200

REX00210

REX00220

REX00230

REX00240

REX00250

REX00260

REX00270

REX00280

REX00290

REX00300

REX00310

REX00320

REX00330

REX00340

REX00350

REX00360

REX00370

REX00380

REX00390

REX00400

REX00410

REX00420

REX00430

REX00440

REX00450

REX00460

REX00470

REX00480

REX00490

REX00500

REX00510

REX00520

REX00530

REX00540

REX00550

REX00560

REX00570

REX00580

REX00590

REX00600

REX00610

REX00620

REX00630

REX00640

REX00650

REX00660

REX00670

REX00680

REX00690

REX00700

REX00710

REX00720

REX00730

REX00740

REX00750

REX00760

REX00770

REX00780

REX00790

REX00800

REX00810

REX00820

REX00830

REX00840

REX00850

REX00860

REX00870

REX00880

REX00890

REX00900

REX00910

REX00920

REX00930

REX00940

REX00950

REX00960

REX00970

REX00980

REX00990

REX01000

REX01010

REX01020

REX01030

REX01040

REX01050

REX01060

REX01070

REX01080

REX01090

REX01100

REX01110

REX01120

REX01130

REX01140

REX01150

REX01160

REX01170

REX01180

REX01190

REX01200

REX01210

REX01220

REX01230

REX01240

REX01250

REX01260

REX01270

REX01280

REX01290

REX01300

REX01310

REX01320

REX01330

REX01340

REX01350

REX01360

REX01370

REX01380

REX01390

REX01400

REX01410

REX01420

REX01430

REX01440

REX01450

REX01460

REX01470

REX01480

REX01490

REX01500

REX01510

REX01520

REX01530

REX01540

REX01550

REX01560

REX01570

REX01580

REX01590

REX01600

REX01610

REX01620

REX01630

REX01640

REX01650

REX01660

REX01670

REX01680

REX01690

REX01700

REX01710

REX01720

REX01730

REX01740

REX01750

REX01760

REX01770

REX01780

REX01790

REX01800

REX01810

REX01820

REX01830

REX01840

REX01850

REX01860

REX01870

REX01880

REX01890

REX01900

REX01910

REX01920

REX01930

REX01940

REX01950

REX01960

REX01970

REX01980

REX01990

REX02000

REX02010

REX02020

REX02030

REX02040

REX02050

REX02060

REX02070

REX02080

REX02090

REX02100

REX02110

REX02120

REX02130

REX02140

REX02150

REX02160

REX02170

REX02180

REX02190

REX02200

REX02210

REX02220

REX02230

REX02240

REX02250

REX02260

REX02270

REX02280

REX02290

REX02300

REX02310

REX02320

REX02330

REX02340

REX02350

REX02360

REX02370

REX02380

REX02390

REX02400

REX02410

REX02420

REX02430

REX02440

REX02450

REX02460

REX02470

REX02480

REX02490

REX02500

REX02510

REX02520

REX02530

REX02540

REX02550

REX02560

REX02570

REX02580

REX02590

REX02600

REX02610

REX02620

REX02630

REX02640

REX02650

REX02660

REX02670

REX02680

REX02690

REX0270

```

(COND (NULL (SETQ ALPHA (NMLZ (CAR X)))) (RETURN NIL))
(COND (EQ FRV (QUOTE 2)) (RETURN (CONS (QUOTE (CAR ALPHA) X)
    (QUOTE 1))))
(COND (EQ FRV (QUOTE 3))
    (COND (NULL (SFTO BETA (NMLZ (CADDR X)))) (RETURN NIL))
    (T (RETURN (CONS (ADDL (CAR ALPHA) (CAR BETA))
        (T (RETURN (CONS (ADDL (CDR ALPHA) (CDR BETA)))))))))
(COND (EQ (CAR ALPHA) (QUOTE 0))
    (RETURN (CONS (QUOTE (CAR ALPHA) (CDR X)) (QUOTE 0))))
(COND (NULL (SETQ BETA (NMLZ (CDR X)))) (RETURN NIL))
(RETURN (CONS (ADDL (DIAT (CAR ALPHA) (CDR X)) (CAR BETA))
    (CDR BETA))))
C
(NCONC1 (LAMBDA (X Y) (NCONC X (LIST Y))))
C
(DRVPAR (LAMBDA (X Y) (COND ((LESSP (ORDER X) (ORDER Y))
    (CONS (CONS (DRV (CAR X)) (QUOTE 0)) (DRVPAR (CDR X) Y)))
    ((GREATERP (ORDER X) (ORDER Y)) (CONS (CONS (QUOTE 0)
    (DRV (CAR Y)) (DRVPAR X (CDR Y)))) (NULL X) NIL)
    (T (CONS (CONS (DRV (CAR X)) (DRV (CAR Y)))
        (DRVPAR (CDR X) (CDR Y))))))
C
(COMPARE (LAMBDA (X Y) (PROG (U1 U2 NR1 NR2 PAIRLIST APAIR)
    (SETQ U1 (LIST (CONS X Y)))
    COMPI (COND (OP (NULL (SETQ NR1 (NMLZ (CAAR U1))))
        (NULL (SETQ NR2 (NMLZ (CDR U1))))
        (RETURN (QUOTE SYNTAX-ERROR))))
    (COND ((NOT (EQ (CDR NR1) (CDR NR2)))
        (RETURN (QUOTE NOT-EQUIVALENT))))
    (SETQ U2 (NCONC1 U2 (CAR U1)))
    (SETQ U1 (CDR U1))
    (SETQ PAIRLIST (DRVPAR (CAR NR1) (CAR NR2)))
    COMPI2 (COND (NULL PAIRLIST) (GO COMPI3))
    (SETQ APAIR (CAR PAIRLIST))
    (COND ((UR (MEMBER APAIR U1) (MEMBER APAIR U2)) (GO COMPI2)))
    (SETQ U1 (NCONC1 U1 APAIR)) (GO COMPI2)
    (SETQ U2 (NCONC1 U2 (T (GO COMPI1))))))
    )
C
EXAMPLES
NMLZ( ((B.C).D) + ((A.H) + C) + A) )
NMLZ( ((A) + ((A.B) + (B.A))) * )
NMLZ( (((B.A) + ((A) * ((A.B) * (A *))) * (A *)))
    COMPARE( A (A + (((B.C).D).E).0)) )
COMPARE( (A * (((((A * *) * *) * *) * *) * *)
    COMPARE( ((A.B) * A) (A.(B.A) *)))
    COMPARE( ((A + B) * ((A * ((B.(A *))) *)))
    COMPARE( ((A.(B *).A) * (((A.(B *).A) * (A.(B *))) * )
    COMPARE( ((A + (B.A)) * C) (((A *).(B.A)) * ((A *).C)) )
    COMPARE( ((A + (B.A)) + (B.A)) *
    COMPARE( ((B.A) + ((A *).(A *))) * (A *))
    COMPARE( ((B + C).(A *).B)) * ((C.(A *).C) + A)
    COMPARE( (((B *).C).(A *).F) * ((B *).(C.(A *).C)) + ((B *
    (A)))
    COMPARE( (F.(((B.(G.(C.G) *).U))) * (A.G)))
    (F.(((B.(G.D)) + (B.(C.(C.G) *).U)))) * (A.G))) )
STOP

```

END OF SOURCE LISTING.

FUNCTION EVALQUOTE HAS BEEN ENTERED. ARGUMENTS...

DEFINE

```
((EQ1 (LAMBDA (X Y) (COND ((AND (ATOM X) (ATOM Y)) (OR (ATOM X) (ATOM Y)) NIL) ((EQ1 (CAR X) (CAR Y)) (EQ1 (CDR X) (CDR Y))) (T NIL)))) (HANDLE (LAMBDA (X) (COND ((ATOM X) X) (T (CAR X)))) (DRV (LAMBDA (X) (COND ((ATOM X) (LIST (QUOTE 0) (QUOTE #))) (T (CDR X)))) (ORDER (LAMBDA (X) (COND ((NULL X) (QUOTE 2)) (T (CDR (ASSOC (HANDLE (CAR X)) (QUOTE (A 1) (B 2) (C 3) (D 4) (E 5) (F 6) (G 7) (H 8) (I 9) (J 10) (K 11) (L 12) (M 13) (N 14) (O 15) (P 16) (Q 17) (R 18) (S 19) (T 20) (U 21) (V 22) (W 23) (X 24) (Y 25) (Z 26)))))) (MEMBER (LAMBDA (X Y) (COND ((NULL Y) NIL) ((EQ1 X (CAR Y)) (T (MEMBER X (CDR Y)))))) (FORM (LAMBDA (X) (COND ((ATOM X) (COND ((EQ X Y) (COND ((MEMBER X (QUOTE (A B C D E F G H I J K L M N O P Q R S T U V W X Y Z))) (QUOTE 1)) (T NIL))) (ATOM (CDR X)) (QUOTE 4)) ((EQ1 (CAR X) (QUOTE #)) (COND ((NULL (CDR X)) (QUOTE 2) (T NIL))) ((EQ1 (CDR X) (QUOTE +)) (COND ((ATOM (CDR X)) NIL) (T (COND ((NULL (CDR X)) (QUOTE 3)) (T NIL)))))) (T (LIST X (QUOTE +) Y))) (T (COND ((EQ1 (CAR X) Y) X) (T (LIST (CAR X) (QUOTE +) (MERGE1 (CADDR X) Y)))))) (MERGE (LAMBDA (X Y) (COND ((ELEMENTARY Y) (MERGE1 X Y)) (T (MERGE (MERGE1 X (CAR Y)) (CDR Y)))))) (ADDL (LAMBDA (X Y) (COND ((LESSP (ORDER X) (ORDER Y)) (CONS (CAR X) (ADDL (CDR X) Y)) (CONS (CAR Y) (ADDL X (CDR Y)))))) (ADDC (LAMBDA (X Y) (COND ((EQ X (CONS (CONS (HANDLE (CAR X)) (MERGE (DRV (CAR X)) (DRV (CAR Y)))) (ADDL (CDR X) (CDR Y)))) (ADDC (LAMBDA (X Y) (COND ((EQ X (QUOTE 0)) Y) (T (QUOTE 1)))) (OTBT (LAMBDA (X Y) (COND ((NULL X) NIL) (ATOM (CAR X)) (CONS (CONS (CAR X) Y) TOTBT (CDR X) Y))) (T (CONS (CONS (CAAR X) (CONS (CDAR X) Y) (TOTBT (CDR X) Y)))) (NMLZ (LAMBDA (X) (PROG (ALPHA BETA FRM) (COND ((NULL (SETQ FRM (FORM X)) (RETURN NIL))) (COND ((EQ FRM (QUOTE 0)) (RETURN (CONS NIL (QUOTE 0)))) (COND ((EQ FRM (QUOTE 1)) (RETURN (CONS (CAR ALPHA) X) (QUOTE 1)))) (COND ((EQ FRM (QUOTE 3)) (COND ((NULL (SETQ BETA (NMLZ (CADDR X)))) (RETURN NIL)) (COND ((EQ FRM (QUOTE 2)) (RETURN (CONS (ADDL (CAR ALPHA) (CAR BETA)) (ADDC (CDR ALPHA) (CDR BETA)))))) (COND ((EQ (SETQ BETA (NMLZ (CDR X))) (RETURN NIL)) (RETURN (CONS (ADDC (DIAT (CAR ALPHA) (CDR X)) (QUOTE 0))) (RETURN (CONS (ADDC (DIAT (CAR ALPHA) (CDR X)) (CAR BETA)) (CDR BETA)))))) (NCONC1 (LAMBDA (X Y) (NCONC X (LIST Y)))) (DRVPAIR (LAMBDA (X Y) (COND ((LESSP (ORDER X) (ORDER Y)) (CONS (CONS (DRV (CAR X)) (QUOTE 0)) (DRVPAIR (CDR X) Y)) (GREATERP (ORDER X) (ORDER Y)) (CONS (CONS (QUOTE 0) (DRV (CAR Y)) (DRVPAIR X (CDR Y)))) (NULL X) NIL) (T (CONS (CONS (DRV (CAR X)) (DRV (CAR Y)) (DRVPAIR (CDR X) (CDR Y)))) (COMPARE (LAMBDA (X Y) (PROG (U1 U2 NR1 NR2 PAIRLIST APAIR) (SETQ U1 (LIST (CONS X Y))) (COMP1 (COND ((OR (NULL (SETQ NR1 (NMLZ (CAAR U1)))) (NULL (SETQ NR2 (NMLZ (CDR U1)))) (RETURN (QUOTE SYNTAX-ERROR))) (COND ((NOT (EQ (CDR NR1) (CDR NR2))) (RETURN (QUOTE NOT-EQUIVALENT))) (SETQ U2 (NCONC1 U2 (CAR U1))) (SETQ U1 (CDR U1)) (SETQ PAIRLIST (DRVPAIR (CAR NR1) (CAR NR2))) (COMP2 (COND ((NULL PAIRLIST) (GO COMP3)) (SETQ APAIR (CAR PAIRLIST)) (SETQ PAIRLIST (CDR PAIRLIST)) (COND ((OR (MEMBER APAIR U1) (MEMBER APAIR U2)) (GO COMP2))) (SETQ U1 (NCONC1 U1 APAIR)) (GO COMP2) (COMP3 (COND ((NULL U1) (RETURN U2)) (T (GO COMP1)))))))))
```

END OF EVALQUOTE.VALUE IS...

((EQ1 HANDLE DRV ORDER MEMBER FORM ELEMENTARY MERGE1 MERGE ADDL ADDC DIAT NMLZ NCONC1 DRVPAIR COMPARE)

FUNCTION EVALQUOTE HAS BEEN ENTERED. ARGUMENTS...

NMLZ

((((B.C).D) + (((A.B) + C) + A)))

END OF EVALQUOTE.VALUE IS...

((A B + (0 #)) (B C.D) C). 0)

FUNCTION EVALQUOTE HAS BEEN ENTERED. ARGUMENTS...

NMLZ

((A + (A.B) + (B.A))) #)

END OF EVALQUOTE.VALUE IS...

((A ((0 #) + B) (A + ((A.B) + (B.A))) #) (B A (A + ((A.B) + (B.A))) #)). 1)

FUNCTION EVALQUOTE HAS BEEN ENTERED. ARGUMENTS...

NMLZ

(((((B.A) + ((A #) A.B))) #) A #)

END OF EVALQUOTE.VALUE IS...

((A (((A #) A.B) + B) ((B.A) + ((A #) A.B))) #) A # + (A #) (B (A ((B.A) + ((A #) A.B))) #) A #)). 1)

FUNCTION EVALQUOTE HAS BEEN ENTERED. ARGUMENTS...

COMPARE

(A (A + (((B.C).D).E). 0)))

HITAC LISP

02/15/74

PAGE 5

A)) * ((A *),C)))

FUNCTION EVALQUOTE HAS BEEN ENTERED. ARGUMENTS...

COMPARE

(F (B G ((C,G) #),D) #) A,G) (F ((B G,D) + (B G C G ((C,G) #),D)) #) A,G))

END OF EVALQUOTE, VALUE IS...

```
((F (B G ((C,G) #),D) #) A,G) F ((B G,D) + (B G C G ((C,G) #),D)) #) A,G) (((B G ((C,G) #),D) #) A,G) ((B G C G ((C,G) #),D) #) A,G) (G,G) ((G ((C,G) #),D) (B G ((C,G) #),D) (B G ((C,G) #),D) #) A,G) ((G,D) + (G C G ((C,G) #),D)) #) A,G) ((B G,D) + (B G C G ((C,G) #),D) #) A,G) ((0 #) 0 #) (((C,G) #),D) (B G ((C,G) #),D) #) A,G) ((D + (C G ((C,G) #),D)) (B G,D) + (B G C G ((C,G) #),D)) #) A,G) (((C,G) #),D) (B G ((C,G) #),D) #) A,G) (G ((C,G) #),D) #) A,G) (G ((C,G) #),D) (B G,D) + (B G C G ((C,G) #),D)) #) A,G) (((C,G) #),D) (B G ((C,G) #),D) #) A,G) ((C,G) #),D) #) A,G) (((C,G) #),D) #) A,G) ((B G ((C,G) #),D) (B G ((C,G) #),D) #) A,G) ((B G C G ((C,G) #),D)) #) A,G))
```

END OF EVALQUOTE OPERATOR,

STATISTICS

RUN TIME(SEC)... 58.

CONS COUNTER ... 53755.

RECLAIMER ... 17.

#FIN

// END

```

***** ACCOUNTING INFORMATION *****
** JOB NAME--ZY556      JOB#--0560  JOB CLASS--B
**
** CPU TIME--000100  START TIME--175526  END TIME--175639  MEMORY--100000
**
** SYSIPT--000115  SYSLSI--000237  SYSUPT--000000
**
** C0--000192
**
*****EDOS V10 R00*****
*****

```

```

0001 SNOBOL
0002 *
0003     ALPHBT = 'ABCDEFGHIJKLMNOPQRSTUVWXYZ' / (MIN1)
0004 *
0005 *
0006 *
0007     DEFINE('LEFT(X)', 'LEFT1')
0008     LEFT = '#' X '#'
0009     LEFT '#(' *(X1)* '+' *(X2)* ')' #' = X1 /S(RETURN)
0010     LEFT '#(' *(X1)* '.' *(X2)* ')' #' = X1 /S(RETURN)
0011     LEFT '#' *(X1)* '*' #' = X1 / (RETURN)
0012 *
0013 *
0014 *
0015     DEFINE('RIGHT(X)', 'RIGHT1')
0016     RIGHT = '#' X '#'
0017     RIGHT '#(' *(X1)* '+' *(X2)* ')' #' = X2 /S(RETURN)
0018     RIGHT '#(' *(X1)* '.' *(X2)* ')' #' = X2 / (RETURN)
0019 *
0020 *
0021 *
0022     DEFINE('OPER(X)', 'OPER1')
0023     OPER = '#' X '#'
0024     OPER '#(' *(X1)* '+' *(X2)* ')' #' = '+' /S(RETURN)
0025     OPER '#(' *(X1)* '.' *(X2)* ')' #' = '.' /S(RETURN)
0026     OPER '#' *(X1)* '*' #' = '*' / (RETURN)
0027 *
0028 *
0029 *
0030     DEFINE('HNDL(X)', 'HNDL1')
0031     HNDL = .EQ(SIZE(X), '1') X /S(RETURN)
0032     HNDL = LEFT(X) / (RETURN)
0033 *
0034 *
0035 *
0036     DEFINE('DRV(X)', 'DRV1')
0037     DRV = .EQ(SIZE(X), '1') '0*' /S(RETURN)
0038     DRV = RIGHT(X) / (RETURN)
0039 *
0040 *
0041 *
0042     DEFINE('ORDER(X)', 'ORDER1')
0043     ORDER = EQUALS(X, 'NIL') '26' /S(RETURN)
0044     X1 = EQUALS(OPER(X), '+') HNDL(LEFT(X)) /S(ORDER2)
0045     X1 = HNDL(X)
0046     ORDER2 ALPHBT *X2* X1
0047     ORDER = SIZE(X2) / (RETURN)
0048 *
0049 *
0050 *
0051     DEFINE('CONA(X,Y)', 'CONA1')
0052     CONA = Y
0053     X1 = EQUALS(OPER(Y), '+') LEFT(Y) /F(CONA2)
0054     CONA = UNEQL(X, X1) '(' X1 '+' CONA(X, RIGHT(Y)) ')'
0055     / (RETURN)
0056     CONA2 CONA = UNEQL(X, Y) '(' Y '+' X ')' / (RETURN)
0057 *
0058 *
0059 *
0060     DEFINE('CON(X,Y)', 'CON1')
0061     CON = UNEQL(OPER(X), '+') CONA(X, Y) /S(RETURN)
0062     CON = CON(RIGHT(X), CONA(LEFT(X), Y)) / (RETURN)
0063 *

```

```

0064 *
0065 *
0066 DEFINE('MRG(X,Y)', 'MRG1')
0067 MRG1 ORD1 = ORDER(X)
0068 ORD2 = ORDER(Y)
0069 MRG = .LT(ORD1,ORD2)
0070 . ' (' LEFT(X) '+' MRG(RIGHT(X),Y) ')' /S(RETURN)
0071 MRG = .GT(ORD1,ORD2)
0072 . ' (' LEFT(Y) '+' MRG(X,RIGHT(Y)) ')' /S(RETURN)
0073 MRG = .EQ(ORD1,'26') 'NIL' /S(RETURN)
0074 MRG = ' ((' HNDL(LEFT(X)) '.'
0075 . CON(DRV(LEFT(X)),DRV(LEFT(Y))) '+'
0076 . MRG(RIGHT(X),RIGHT(Y)) ')' /S(RETURN)
0077 *
0078 *
0079 *
0080 DEFINE('DISTR(X,Y)', 'DISTR1')
0081 DISTR1 DISTR = EQUALS(X,'NIL') X /S(RETURN)
0082 DLFT = LEFT(X)
0083 DISTR = .EQ(SIZE(DLFT),'1')
0084 . ' ((' DLFT '.' Y ')' + DISTR(RIGHT(X),Y) ')'
0085 . /S(RETURN)
0086 DISTR = ' ((' LEFT(DLFT) '.' (RIGHT(DLFT)
0087 . '.' Y ')') + DISTR(RIGHT(X),Y) ')' /S(RETURN)
0088 *
0089 *
0090 *
0091 DEFINE('ADDCNST(X,Y)', 'ADDCNST1')
0092 ADDCNST1 ADDCNST = EQUALS(X,'0') Y /S(RETURN)
0093 ADDCNST = '0*' /S(RETURN)
0094 *
0095 *
0096 *
0097 DEFINE('NML(X)', 'NML11', 'ALPHA')
0098 NML11 OPE = .NE(SIZE(X),'1') OPER(X) /F(NML12)
0099 EQUALS(OPE,'*') /S(NML21)
0100 EQUALS(OPE, '+') /S(NML31)
0101 EQUALS(OPE, '.') /S(NML41) F(FRETURN)
0102 *
0103 NML12 NML = EQUALS(X,'0') '(NIL+0)' /S(RETURN)
0104 ALPHBT X /F(FRETURN)
0105 NML = ' ((' X '+NIL)+0)' /S(RETURN)
0106 *
0107 *
0108 NML21 ALPHA = NML(LEFT(X)) /F(FRETURN)
0109 NML = ' (' DISTR(LEFT(ALPHA),X) '+0*)' /S(RETURN)
0110 *
0111 NML31 ALPHA = NML(LEFT(X)) /F(FRETURN)
0112 BETA = NML(RIGHT(X)) /F(FRETURN)
0113 NML = ' (' MRG(LEFT(ALPHA),LEFT(BETA)) '+'
0114 . ADDCNST(RIGHT(ALPHA),RIGHT(BETA)) ')' /S(RETURN)
0115 *
0116 NML41 ALPHA = NML(LEFT(X)) /F(FRETURN)
0117 NML = EQUALS(RIGHT(ALPHA),'0')
0118 . ' (' DISTR(LEFT(ALPHA),RIGHT(X)) '+0)' /S(RETURN)
0119 BETA = NML(RIGHT(X)) /F(FRETURN)
0120 NML = ' (' MRG(DISTR(LEFT(ALPHA),RIGHT(X)),LEFT(BETA))
0121 . '+' RIGHT(BETA) ')' /S(RETURN)
0122 *
0123 *

```

```

0124 *
0125 DEFINE('SUBST(X,Y)','SUBST1')
0126 SUBST1 SUBST = X
0127 SUBST2 X '(' *(TRM)* '+' *(X1)* ')' = X1 /F(RETURN)
0128 HL = HNDL(TRM)
0129 DV = DRV(TRM)
0130 I = '1'
0131 SUBST3 I = UNEQL($ (Y I),DV) I + '1' /F(SUBST4)
0132 $(Y I) = .GT(I,$ ('LH' Y)) DV /F(SUBST3)
0133 $ ('LH' Y) = I
0134 SUBST4 SUBST TRM = '(' HL '.' Y I ')' /F(SUBST2)
0135 *
0136 *
0137 *
0138 DEFINE('PRINT(X)','PRINT1')
0139 PRINT1 X *SYSPOT/'70'* *X* /S(PRINT1)
0140 SYSPOT = X /F(RETURN)
0141 *
0142 *
0143 *
0144 DEFINE('AM(X)','AM1')
0145 AM1 J = '1'
0146 S1 = X
0147 LHS = '1'
0148 AM2 NR = NML($ ('S' J)) /F(AM3)
0149 SYSPOT =
0150 PRINT('S' J ' = ' $ ('S' J))
0151 PRINT(' = ' NR)
0152 PRINT(' = (' SUBST(LEFT(NR),'S') '+' RIGHT(NR) ')')
0153 J = .NE(J,LHS) J + '1' /S(AM2)F(RETURN)
0154 *
0155 AM3 SYSPOT = 'NOT RE' /F(RETURN)
0156 *
0157 *
0158 *
0159 *
0160 MIN1 X1 = SYSPIT
0161 X1 = UNEQL(X1,'END') AM(X1) /S(MIN1)
0162 END
**** LIST END ****
@

```

209 : PR, SNOB, 2, 99
1 SNOBOL

FIN DE COMPILATION . NOMBRE D'ERREURS: 0

NOMBRE DE MOTS OCCUPES PAR LE PROGRAMME: 2006

((B.A)+(A*(A.B))*A*)

S1 = (((B.A)+(A*(A.B))*A*)
= (((A.(A*(((B+(A*(A.B))) . ((B.A)+(A*(A.B))*A*))) + (B.((A.((B.A)+(A*(A.B))*A*))+NIL)))+0*)
= (((A.S2)+((B.S3)+NIL))+0*)

S2 = (A*(((B+(A*(A.B))) . ((B.A)+(A*(A.B))*A*)))
= (((A.(((B+(A*(A.B))) . ((B.A)+(A*(A.B))*A*))+A*))+ (B.(((B.A)+(A*(A.B))*A*))+NIL))+0*)
= (((A.S4)+((B.S1)+NIL))+0*)

S3 = ((A.((B.A)+(A*(A.B))*A*))
= (((A.(((B.A)+(A*(A.B))*A*))+NIL))+0*)
= (((A.S1)+NIL))+0*)

S4 = (((B+(A*(A.B))) . ((B.A)+(A*(A.B))*A*))+A*)
= (((A.(A*(((B+(A*(A.B))) . ((B.A)+(A*(A.B))*A*))) + (B.(((B.A)+(A*(A.B))*A*))+NIL))+0*)
= (((A.S2)+((B.S1)+NIL))+0*)

(A+((A.B)+(B.A)))*

S1 = (A+((A.B)+(B.A)))*
= (((A.((B+0*). (A+((A.B)+(B.A))*) + (B.(A.(A+((A.B)+(B.A))*) + NIL)))+0*)
= (((A.S2)+((B.S3)+NIL))+0*)

S2 = ((B+0*). (A+((A.B)+(B.A)))*
= (((A.((B+0*). (A+((A.B)+(B.A))*) + (B.((A.(A+((A.B)+(B.A))*) + (A+((A.B)+(B.A))*) + NIL)))+0*)
= (((A.S2)+((B.S4)+NIL))+0*)

S3 = (A.(A+((A.B)+(B.A)))*
= (((A.(A+((A.B)+(B.A))*) + NIL))+0*)
= (((A.S1)+NIL))+0*)

S4 = ((A.(A+((A.B)+(B.A))*) + (A+((A.B)+(B.A))*)
= (((A.((B+0*). (A+((A.B)+(B.A))*) + (A+((A.B)+(B.A))*) + (B.(A.(A+((A.B)+(B.A))*) + NIL)))+0*)
= (((A.S5)+((B.S3)+NIL))+0*)

S5 = (((B+0*). (A+((A.B)+(B.A))*) + (A+((A.B)+(B.A))*)
= (((A.((B+0*). (A+((A.B)+(B.A))*) + (B.((A.(A+((A.B)+(B.A))*) + (A+((A.B)+(B.A))*) + NIL)))+0*)
= (((A.S2)+((B.S4)+NIL))+0*)

END

SNOBOL HP FRANCE - FIN DE L'INTERPRETATION
00075 GARBAGE COLLECTION(S)

SEND SNOB