新高速文字列処理手法とその評価

角田博保（電気通信大学）

1. 序

SNOBOL3[1]の処理系をもとにして、文字列の新高速処理手法を開発した。本文では処理手法のうち一般性を持つ部分について、高速である根拠を述べ、実例をもとにして評価する。

SNOBOL3はSNOBOL族の言語としては一世代古いものであるが、動的な文字列を唯一のデータ型とした構成をしており、文字列処理としての基本的機構を十分含んでいる。ここで述べる手法はSNOBOL3に依存して開発されたものであるが、上記のSNOBOL3の特徴により大部分は一般性を持っている。普通に使われているプログラム言語の文字列処理機構として利用することができるであろう。

本手法の基本的アイディアは文字列を均一なデータ構造で表現する（たとえばべたづめ方式SNOBOL3処理系[2]）のではなく、各文字列の使われかたに応じたいくつかの利用属性を導入し、各属性に対して処理が最適になるようなデータ構造を与える、というものである。また、各文字列がどの属性を持つかは固定したものではなく、属性変更規則に従って処理が最適になるように随時変更されていくというものである。この規則がいかに妥当であるかが問題である。

以下、第2節では文字列に対する処理を概観し、いくつかの処理手法について考察する。また、文字列をどういうデータ構造で扱うかについても考察する。第3節では新手法について、各属性に対応するデータ構造および属性変更規則を中心に述べる。第4節は変更規則の妥当性について、実例をもとにして評価する。第5節はまとめとして、本手法を適用して作成されたSNOBOL3処理系とその実用性について述べる。

2. 文字列操作とデータ構造

SNOBOL3における文字列に対する基本操作を抽出したものを図2.1に示す。

命名	A = O	A = O O O	O … *A* …	f(u)
	右辺単項	右辺複項	パターン照合における代入	引数渡し
参照	… A …			
連結	… A B …			
比較	O … A …			
パターン照合	A … C … ** … *()*			
置き換え	A O = O			
関接	$A			
算術演算	A+B A-B A*B A/B			
I/O	SYSPOT = SYSPIT			
その他	define trim 算術比較 read size			

図2.1 SNOBOL3における文字列操作

追加　　A = A ○○○

挿入　　A = ○○○ A

図2.2 追加/挿入

この他に、意味を考えることによって図2.2に示すように、ある変数の持っている値に対する追加操作，挿入操作を抽出することができる．この操作を可変操作と呼ぶことにする．図2.1の置き換えもこの部類に入る．

変数と値との対応を実現するには大きく分けて2方法ある．値の共有を許す方法とそうでない方法である．ここで扱っている文字列は固定長ではないので、それを値としてあらわすには，適当な文字列表現へのポインタを使うのが一般的である．変数RIGHTが値"ABC"を持っている状況を図2.3のようにあらわすことにする．すると代入文　LEFT = RIGHT　を実行した結果の状況は共有を許す，許さないによって図2.4の通りになる（以下，共有型／非共有型と呼ぶ）．

図2.3　変数と値

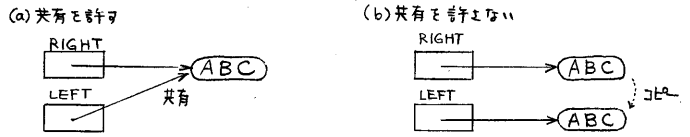

図2.4　代入(命名)操作の実行

共有型では代入操作は単純で高速に処理できる．その代り、可変操作に対してはその表現を書き換えることはできないので，新たに表現を生成してやらなければならない．逆に，表現を書き換える必要がないのでコンパクトなデータ構造で表現を実現することが可能である．

非共有型では代入操作は文字列表現のコピーを伴なうので高速には処理し難い．可変操作に対してはその表現自体を書き換えることで実現できる．ただし，そのためには，書き換えが可能であるような表現にしておく必要がある．

文字列表現の構成法はいくつも考えられるが，大きく分けてリスト方式とべたづめ方式に分かれよう．書き換え可能かどうかによって、その構成に違いが生じる．

リスト方式は可変操作に適しているといえよう．リンクのつけかえで，追加、挿入が簡単に処理できる．べたづめ方式（連続記憶領域に文字バイトを詰めて格納する）では可変操作を実現するためには付加的な機構が必要になる．図2.5の(a)に書き換えを考慮してない構成，(b)に書き換えを考慮した構成を示す．各々を不変型べたづめ，可変型べたづめと呼ぶことにする．

変数と値との対応のしかた（共有型，非共有型）と文字列表現の違い（リスト方式，不変型べたづめ，可変型べたづめ）による6通りの文字列処理法に対して、

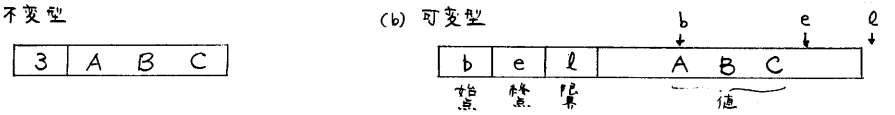

図2.5　2種のべたづめ構造

図2.1で述べた操作がどれだけうまく実現できるかを表2.1に示す．リスト方式は同じ表現の複製を作る際に移動すべき情報量がべたづめに比べて多いので（リンクの分だけ多い）ほとんどの項目でべたづめに比べて劣っている．唯一非共有型で連結操作をおこなうときのみリンクのつけかえによる高速化が可能になる点で勝る．べたづめ方式の内で，不変型で共有の場合と可変型で非共有の場合が優秀である．可変型共有は不変型共有と表2.1では同じ結果になっているが，細かくみるとすべての点で，少しだけ無駄な処理をしていることがわかる．次節で述べる新方式では不変型共有と可変型非共有の融合を計っている．

表2.1 各方式の処理効率

	リスト		不変型べたづめ		可変型べたづめ	
	共有	非共有	共有	非共有	共有	非共有
命名（代入）	○	××	○	×	○	×
連結	××	△	×	×	×	×
比較	△	△	○	○	○	○
パターン照合	△	△	○	○	○	○
置き換え	××	○	×	×	×	○
追加・挿入	××	○	×	×	×	○
間接	△	△	○	○	○	○
I/O	×	×	△	△	○	○
長さ	×	×	○	○	○	○

3. 新高速文字列処理手法

べたづめ方式SNOBOL3処理系では文字列表現として不変型べたづめを使い，変数と値との対応は共有型によった．機械語の移動命令（事務処理用）を利用することでコピーの速度の向上を計っているが，可変型べたづめと比べて，可変操作は大幅に劣っていた．

新手法では文字列表現の均一化をやめ，操作に適した表現を持たせるようにした．必然的に，どういう表現のしかたをしているかを区別するための識別子（属性）が必要となった．この識別子は変数に持たせてある（図3.1）．

属性には，不変，可変の他に，空，文字，整数，入力，出力，等を導入した．各属性に対応した文字列表現を図3.2に示す．

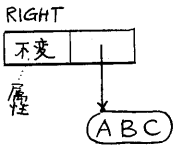

図3.1 変数と属性

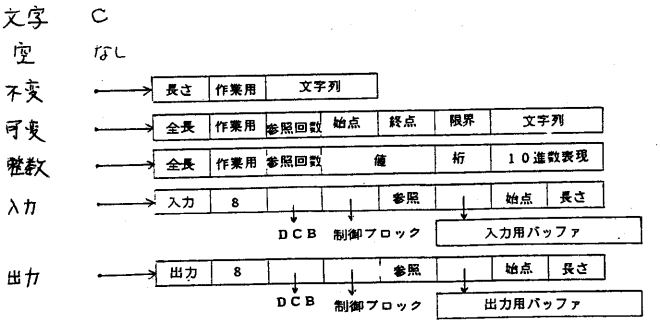

図3.2 属性と文字列表現

空属性と文字属性は不変属性の特別な場合として導入した．整数属性はSNOBOL3に特有な処理のために導入した．入出力に関しては入出力用バッファへのポインタとして値を表現できるという点から導入した．

可変属性の文字列表現が不変属性のそれに外見上は類似しているのは，領域再生に係わっている．不変属性の文字列は文字列領域の中に順次とられ，領域が尽きた時に領域再生がおこなわれる．領域再生は文字列の詰め合わせと文字列へのポインタ値の調整によっておこなう．この際に作業用欄を使う．可変属性の場合も同じ領域内にとられ，領域再生時には不変属性と同じ扱いをうける．それが可能であるように，可変属性の場合も先頭の1語に全長を格納し，次の語を作業欄にしてある．始点，終点，限界，および参照回数は絶対値で表現されているので領域再生による位置の移動に対しても矛盾を生じない．

可変属性の文字列表現は内部にバッファを持ち，対応する文字列を格納している．可変操作に対してはそのバッファ内で表現できる限りはその場でおこなわれるが，バッファ長より大きくなる場合には，新たな文字列表現が生成されることになる．その際には一回り大きなブロックがとられる．

属性は，変数と値との対応方法，および文字列表現のしかたをまとめてあらわしているということができる．変数がどの属性を持つかということは重要である．可変操作をおこなうときは可変属性，代入操作をおこなうときは不変属性がよいといえる．以下に属性決定（変更）規則を示す．

変数の属性決定（変更）規則

①初期値として全変数は空属性を持つ．
②入力・出力属性はその変数に対して要求が出された時点で決定される（組込み関数 PRINT，PUNCH，および READ による）．
③整数属性は整数演算の結果の代入によって生じる．
④可変操作が施された変数は可変属性を持つ．
⑤代入操作において，その右辺，左辺の属性に応じて，代入後の左辺の変数の属性が決定される（表3.1）．

ここで代入操作とは，1)代入文，2)パターン照合における代入，および3)利用者定義関数呼び出しにおける引数渡し，である．代入される側を左辺，代入する側を右辺と呼ぶことにする．右辺が単項の場合（変数，リテラル，関数呼び出し）はそれが持つ属性を右辺の属性と呼ぶ．右辺が連結演算であれば，便宜上，式属性として表には示した．単項とそれ以外を区別してるのは，単項であればポインタの共有で処理できる可能性があるからである．

表3.1 代入後の属性

左辺の属性 ＼ 右辺の属性	文字・空・不変	整数	可変・式・入出力
文字・整数・空・不変	ポインタ代入 右	コピー 右	コピー 長さで 空/文字/不変
可変	コピー 左	コピー 右	コピー 左
入・出力	コピー 左	コピー 左	コピー 左

右辺が文字，空，不変で左辺が文字，整数，空，不変のときはポインタ代入がおこなわれる．その他の場合はコピーがおこる．右辺が可変，式，入出力で左辺が文字，整数，空，不変のときはコピーが必要となるが，その際，右辺の値（文字列）の長さに応じて属性が決定され，対応する表現に置き換えられる．

4．評価

本節では属性変更規則の妥当性について論じ，実例をあげて評価する．不変属性しか存在しないとみなすことができるべたづめ方式[2]を比較の対象とした．

4.1 考察

入力，出力，整数属性の導入は明らかに妥当である．不変属性のうちで長さ0と1のものをそれぞれ空属性，文字属性として表現することも妥当である．すると，問題は不変属性と可変属性の導入である．2節で述べたように，可変操作に対しては可変属性が有効である．以下，可変属性と不変属性とに焦点を絞って論ずる．

可変属性が導入されるのは，規則④で示してあるように，可変操作が施された場合である．可変属性の文字列表現に対しては可変操作が高速に処理できることは明らかである．同じことを不変属性においておこなった場合との優位度は歴然である．その代り，図4.1で示すような文を実行すれば，不変属性ならポインタの代入でよいところをコピーが必要となる．以上のことから，各文の実行回数の比率が問題である．

A = b	代入文の左辺，右辺単項
b = A	代入文の右辺．
f(A)	利用者定義関数の実引数
f(b)	〃　の仮引数で実引数が変数

図4.1 コピーが必要な文（Aの属性は可変）

可変属性を持つ変数の使われかたを調べてみると，その大部分が入力バッファ型と出力バッファ型に分類できる．ここで入力バッファ型とは図4.2の変数INBUFで代表され，出力バッファ型とは図4.3の変数OUTBUFで代表される．

```
INBUF   =   init value          (a)
INBUF   pattern =               (b)
```

図4.2 入力バッファ型

```
OUTBUF  =                       (a)
OUTBUF  =  OUTBUF  data         (b)
v       =  OUTBUF               (c)
```

図4.3 出力バッファ型

入力バッファ型では、図4.2(a)のように、まず初期値が与えられ、ついで(b)のように先頭から削除されていく。データが尽きたら、また(a)によって供給される。出力バッファ型では、図4.3(a)のように、初期設定として空にされる。ついで(b)のようにdataを追加される。この文を何度か実行した後で、結果を他へ代入する(図4.3(c))。そして(a)から繰り返す。以下、双方について詳しく検討する。

①入力バッファ型

図4.2において、initvalueの長さをn、文(b)の実行回数をm $(m\geqslant 1)$とする。新方式での各文の結実行時間をそれぞれTa, Tb, 旧方式(=べたづめ方式, 不変型のみ)におけるそれぞれをTa', Tb'とする。するとラフな近似から、

$$Ta \fallingdotseq Ta' + nC$$
$$Tb + \left(\frac{m-1}{m} + \frac{m-2}{m} + \cdots + \frac{1}{m}\right)\cdot nC \fallingdotseq Tb' \qquad \text{ここでCは1文字コピーに要する時間}$$
つまり $$Tb + \frac{m-1}{2}\cdot nC \fallingdotseq Tb'$$

よって、

$$Ta + Tb \fallingdotseq Ta' + Tb' - \frac{m-3}{2}\cdot nC$$

となる。これはmが3以上であれば新方式が速いことをあらわしている。また、図4.2において、initvalueが連結の形で与えられていれば、旧方式でもコピーが必要となるので、$Ta = Ta'$ となり $Ta + Tb \fallingdotseq Ta' + Tb' - \frac{m-1}{2}\cdot nC$ となって、mが1以上、つまりこの形の文では新方式が必ず速いということができる。

(注)上記の式は　　、(b)の実行は新方式では始点柄の書き換えだけでよいところを、旧方式では$\frac{m-1}{m}\cdot n$, $\frac{m-2}{m}\cdot n$, …, $\frac{n}{m}$ 文字分のコピーが必要となることから得られる。

②出力バッファ型

図4.3において、文(b)の実行回数をm $(m>1)$、最終的にOUTBUFが持つ値の長さをnとする。①と同様にして $Ta, Tb, Tc, Ta', Tb', Tc'$を定義する。すると、

$$Ta \fallingdotseq Ta'$$
$$Tb + \left(\frac{1}{m} + \frac{2}{m} + \cdots + \frac{m}{m}\right)\cdot nC \fallingdotseq Tb' + nC$$
つまり $$Tb + \frac{m+1}{2}\cdot nC \fallingdotseq Tb' + nC$$
$$Tc \fallingdotseq Tc' + nC$$

よって

$$Ta + Tb + Tc \fallingdotseq Ta' + Tb' + Tc' - \frac{m-3}{2}\cdot nC$$

となる。これもmが3以上であれば新方式が速いことをあらわしている。また、図4.3においてOUTBUFの代入が(c)の形でない場合(実際はその方が多い)は $Tc \fallingdotseq Tc'$ となるので①の場合と同様にmが1以上、つまり必ず新方式が速いということができる。またmが十分大きければ、

$$Tb \fallingdotseq nC \qquad Tb' \fallingdotseq \left(\frac{1}{m} + \frac{2}{m} + \cdots + \frac{m}{m}\right)nC \fallingdotseq \frac{m+1}{2}\cdot nC$$

となり、これより

$$Tb \cdot \frac{m+1}{2} \fallingdotseq Tb'$$

となる。$\frac{m}{2}$に比例して高速に処理ができるわけである。

4.2 実例

1) 字句解析（名前を抜き出すプログラム）

変数は6つ．属性のうちわけは，整数1，入力1，出力1，文字1，可変2であった．可変2つは入力バッファ型と出力バッファ型各々1であった．

入力バッファ型は図4.2の(a)1回当り、(b)28回であった（$m=28$）．実測値で $T_b \cdot 3.3 = T_b'$ であった．また $T_a = T_a'$ であった．

出力バッファ型の方は $T_c = T_c'$ であり、$m=3.7$ であった．この場合実測値は $T_b \cdot 6.1 = T_b'$ であった．これはdataが1文字であったので文字属性として扱われたためと思われる．

2) マクロ展開

変数は56個．入出力属性4，文字属性13，不変属性4，文字，空，あるいは不変属性となったもの27，および可変属性8であった．可変属性8つは入力バッファ型と出力バッファ型各々4であった．新方式にとって有利であった場合を図4.4に、不利であった場合を図4.5に示す．図4.4ではT.INBUFは入力バッファ型の変形となっている．図4.4(b)は読み過ぎたデータを返す作業（put back）にあたっている．(a)と(c)の比率をみれば $m = 119$ になっている．それに加えて(b)は可変操作であるので、大幅に新方式が高速になっている．

図4.5は出力バッファ型の例である．この際 $m = 2.5$ となり、新方式が不利となっている（$m<3$ である）．このような例は実際にはあまり生じていない．また，この例に限っていえば、T.CHARが文字属性を持つこと，および実行回数が全体に比べて少ないこと（2.7%にあたる）が指摘される．

```
T.INBUF   *T.N/'1'* =          (a)  11645回
T.INBUF   = T.TOKEN T.INBUF    (b)   3202回
T.INBUF   = T.ATRIM            (c)     98回
```

図4.4 有利な状況

```
T.GTOK    = T.GTOK T.CHAR      (a)   2585回
T.GETTOK  = T.GTOK             (b)   1016回
```

図4.5 不利な状況

3) テスト用13題

筆者の手元にある例題（13件）について，変数の個数，その内の可変属性のものの数，新方式が完全に有利であるもの（入力バッファ型で $T_a = T_a'$ または出力バッファ型で $T_c = T_c'$）の数、不利なものの数を調べた（表4.1）．この例題では不利

表4.1 13例題

	1	2	3	4	5	6	7	8	9	10	11	12	13
変数	14	13	20	19	11	8	7	10	2	6	25	32	13
可変	5	3	4	5	4	2	2	5	0	2	5	3	4
有利	4	1	2	3	4	0	0	1		1	0		2
不利													

1~4 簡単なプリプロセッサ
5 ハノイの塔
6 数式微分
7 WANG
8 エディタ
9 コピー
10~11 プリプロセッサ
12 正規化
13 一般射撃

な場合は発見されなかった。ここでいう完全に有利な場合とは可変操作以外が可変属性に対して施されないことに対応している。

4.3 結論

3節で示した属性変更規則にしたがって導入される可変属性は大むね可変操作のみが適用される。そうでない場合でも、可変属性を導入したことによって高速化していることがわかった。

5. まとめ

本手法を適用した実験版SNOBOL3処理系はFACOM 230-45S上に作成された。コンパイラ本体はAGN (Attribute Grammar Notation)という記述法で書かれ(約1000行)、アセンブラに展開して約20000行である。コンパイラの出力はマクロアセンブラ語である。実行時支援ルーチン群はマクロアセンブラ語で約3000行である。

4節で述べた字句解析プログラムで旧方式に比べて約3倍の高速化が達成されている。同じ処理をPascalで書いたものと比べて、約半分の速度で動くことがわかっている。十分に実用性があるといえる。

参考文献

1) Farber, D.J., Griswold, R.E. and Polonsky, I.P. : The SNOBOL3 Programming Language, Bell System Technical Journal, Vol. 45, pp. 895-944 (July-Aug. 1966).

2) 角田博保 : SNOBOL3言語のベたづめ方式処理系とその評価, 情報処理学会論文誌, Vol. 22, No. 5 (Sep. 1981).