べたづめ方式 SNOBOL3 処理系の移植について

角田博保（東京工業大学　理学部）

概要
機械の特徴を生かして作られた SNOBOL3 処理系の高速性を維持したもとでの移植について述べる．

0．序

文字列処理用言語 SNOBOL3 の処理系において，データ構造を単純化し，事務計算用命令を積極的に利用することで，処理速度の大幅な高速化をはかることができることは，FACOM 230-45S において実証済みである[1]．そこでこの処理方式が他の計算機でもうまくいくのかを検証すること，また同時に高速な SNOBOL3 処理系を広範に提供することを目標として，IBM system/370 に移植することを考えた（以後 FACOM 230-45S を 45S，IBM system/370 を 370，45S 上の SNOBOL3 処理系を SN3/45S，同じく 370 上のものを SN3/370 と略す）．

SN3/45S は移植を考えて作られた処理系ではない．その場合，移植作業にもただ移すというやりかた以外に，移植用処理系を新たに作るというやりかたが存在する．しかしながら，機械の特徴を最大限に利用した移植用処理系を作ることはかなりむつかしいので，今回はただ移すことにした．そのかわりできるだけ移植用処理系の感触を得られるような方法をとった．

SN3/45S はシステム作成用言語 SL45（PL360 風な言語）で書かれている．移植作業は SL45 のクロスコンパイラを作るという方針でおこなうことにした．

1．SNOBOL3 言語

SNOBOL3 言語は文字列処理用言語で Griswold らによって設計された[2]．SNOBOL 言語といえば，SNOBOL4 [3] を指すのが通例となっているが，SNOBOL3 のシンプルな構造は捨てがたい．また，文字列処理系の内部構造の研究には SNOBOL3 で十分であると考える．

図 1.1 は radix sort をする SNOBOL3 プログラムである．図 1.2 に出力例を示した．

```
* ALPHABETIZATION USING A RADIX SORT TECHNIQUE
BEGIN SYSPIT      *SIZE* ' '
START SYSPIT      *WORDS* ' '                                  /F(LO)
      LIST        = LIST WORDS                                /(START)
LO    SYSPOT      = 'THE LIST TO BE ALPHABETIZED IS : ' LIST
      SYSPOT      =
L1    ALPHABET    = 'ABCDEFGHIJKLMNOPQRSTUVWXYZ'
L2    SIZE        = SIZE - '1'
      SIZE        '-'                                         /S(FIN)
L3    LIST        *WORD* ',' =                                /F(L5)
      WORD        *HEAD/SIZE* *PIT/'1'*                       /F(L4)
      *PIT        = *PIT WORD ','                             /(L3)
L4    BIN         = BIN WORD ','                              /(L3)
L5    BIN         *LIST* =
L6    ALPHABET    *PIT/'1'* =                                 /F(L1)
      LIST        = LIST *PIT
      *PIT        =                                           /(L6)
FIN   SYSPOT      = 'THE ALPHABETIZED LIST IS : ' LIST
END
```

図1.1　radix sort をするプログラム

```
THE LIST TO BE ALPHABETIZED IS : ARMY,TEST,GLOBAL,ARMORY,GLOBE,ARM,TENSOR,ALIBI,A
RE,GLOW,TENSE,TOTAL,CANCEL,TONSIL,GLADIATOR,MOOILE,MOTILE,ANY,TORSION,PLATITUDE,F
UMBLE,

THE ALPHABETIZED LIST IS : ALIBI,ANY,ARE,ARM,ARMORY,ARMY,CANCEL,FUMBLE,GLADIATOR,
GLOBAL,GLOBE,GLOW,MOOILE,MOTILE,PLATITUDE,TENSE,TENSOR,TEST,TONSIL,TORSION,TOTAL,
```

図1.2 出力結果

2. SN3/45S 処理系

SN3/45S は compile & go 型の処理系である．処理系はコンパイラ部とサポートルーチン群とから成る．コンパイラはプリミティブルーチンの呼び出し列を出力し，文字列や名前をデータ領域に割り付ける．コンパイル終了後オブジェクトコードに制御が移り，プリミティブルーチンがつぎつぎと呼びだされる．コンパイラの処理を図2.1に示す．図2.1ではオブジェクトコードをALGOL流に書いた．こ

```
L    A  "A"  =          /S(L)
```

⇓コンパイル

```
L: CONTROL(n,6,L1); ALCTNA(→A); ALCT(→"A"); MATCH;
   REPLACE;  L1: GOTOV(L,L2);  L2:
```

図2.1 コンパイラの処理

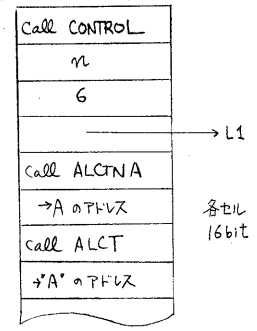

図2.2 オブジェクトの内部表現

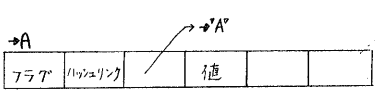

図2.3 名前ブロック →A

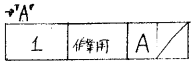

図2.4 べたづめブロック →"A"

ここで関数名はプリミティブルーチン名である．オブジェクトの実際の表現を図2.2に示す．Call CONTROL等は45Sの分岐命令を表わす．名前の表現を図2.3に，文字列の表現を図2.4に示す．文字列がバイト単位に格納される以外は，すべてのセルは1ワード（2バイト）で表現される．

サポートルーチン群は，プリミティブルーチン27個とそれに対するサポートとからなる．

SN3/45S は SL45 で記述されている．コンパイラ部は約2000枚，サポートルーチン群は約4000枚である．

3. うつしかえ

3.1 基本方針

ある処理系を計算機Aから計算機Bに移植するということは，その処理系が記述されている言語Cに対し，Cを入力としBの機械語を出力とする変換作業をおこなうことである．CがFORTRAN等の機械独立言語である場合には，簡単な作業でこの変換をおこなうことができる（計算機B上にはFORTRANコンパイラという自動変換器があるだろうから）．Cがアセンブラ語等の機械従属言語である場合には，この変換は大変わずらわしいものになる．人間が変換作業をおこなう機械に

なることが単純な解決方法であろう．

一般的にみて，記述言語が高級である方が移植作業は楽になるが，その処理系の速度は遅くなる．また，機械的置き換え（ある規則に従った自動変換）に徹するほど虫を入れずに移植できるが処理系の能率は悪くなり、手作業に徹するほど移植された処理系の能率はよくなるが虫がはいりやすくなる．

今回の移植作業は，SN3/45S を 370 に処理系の効率を維持したままうつすことをねらっているので，低級なシステム記述言語を高級言語に書き直すことをしないでそのまま移植のためのソースとして扱うことにした．また，虫が入るのは極力おさえたいし，以後の利用価値も考えて，できるかぎり機械的変換で移植することにした．

SN3/45S は大体 SL45 で記述されているので，それに対するクロスコンパイラを作ることで移植を達成しようと考える．機械依存性が強くまたひとまとまりになっているルーチン（I/O ルーチン，ハッシュルーチン等）は，370 用に手動で書き直すことにした．また，SL45 でコーディングする際に落ちてしまった情報のうち，移植にとって必要となるものは人力で再生しなければならない．再生した情報を表現できるように SL45 を格上げして SL45E とし，必要最小限の手作業と機械的変換で SN3/45S を SL45E で記述し，SL45E に対するクロスコンパイラを作ることで移植をおこなうことにした．

SN3/45S において，コンパイラ部はサポートルーチン群と単純なインタフェースで結合しているので，相互分離可能である．また，コンパイラを機械独立に作ることはわりと簡単である．そこで移植の機会を利用して，コンパイラ部は機械独立型に書き直すことにした（この部分だけは移植用処理系になる）．

3.2 45S と 370 とのアーキテクチャーの比較

SN3/45S で使われている 45S の命令の機能を 370 語で表現するという立場で，相方のアーキテクチャーを比較してみた[4][5]．

- 基本データ単位

45S は 1 ワード 16 ビットのワードマシン，370 は 1 ワード 32 ビットのバイトマシン（半ワード命令がある）．

- アドレスとアドレッシング

45S はワードアドレスが基本．バイトインデックス修飾可能．間接指定あり．最大 64K 語までの論理的なメモリーバンク 2 つ（PN1, PN2）．全領域を直接参照可能．アドレスは 16 ビット．オフセット 16 ビット．

370 はバイトアドレスが基本．ベースレジスタ方式．アドレスは 24 ビット．オフセットは 12 ビット．

- レジスタ

45S は 16 ビットレジスタ 8 個（R0 から R7）．そのうち R1～R3 はインデックスレジスタとして利用可能．370 は 32 ビットジェネラルレジスタ 16 個．

- 命令

45S において，分岐命令は分岐すると R0 に次のアドレスが入る．ロード命令はコンディションコードをセットする．あと探索命令，BOB 命令，MBN 命令，MBT 命令がなく，変換命令，掛算・割算命令が少し異っている点を除けば，45S のほとんどの命令は，370 の命令と論理的に同じ働きをする．

3.3 SN3/370 の基本データ構成の決定

SN3/45S で1語で表現されたデータは SN3/370 でも1語で表現することにした。たいていの場合は半語でよいが，アドレスを表わすのに1語必要になるので統一的に1語にした。したがって，図2.4 のべたづめブロックは図3.3.1のように表現される。

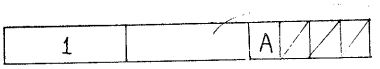

図3.3.1 SN3/370 におけるべたづめブロック

SN3/45S ではポインタは，ある領域の先頭番地を起点にした相対アドレスで表現したが、SN3/370 ではすべて絶対アドレスにした。

図2.1 に示したコンパイラのオブジェクトコードを分岐命令で表現すると，図3.3.2のようになり，領域も多く必要としスピードもあまり速くないので，図3.3.3のような threaded code 方式に改めた。threaded code 方式では図3.3.4に示す driver が

CONTROL(a,b,c)

```
L     15,=A(CONTROL)
CNOP  2,4
BALR  0,15
DC    A(a)
DC    A(b)
DC    A(c)
```

370アセンブラによる表現

図3.3.2 オブジェクトコード(命令)

```
DC    A(CONTROL)
DC    A(a)
DC    A(b)
DC    A(c)
```

図3.3.3 オブジェクトコード(threaded code)

```
L     15,0(11)
BALR  0,15
```

レジスタ11が図3.3.3のオブジェクトの先頭を指している。

図3.3.4 threaded code の driver

図3.3.2 の L と BALR の仕事を代行している。この方式の方が機械独立性が高い。

3.4 コンパイラ部の移植

SNOBOL3 の処理系において，一般に翻訳時間は解釈実行時間に比べて非常に短かい。したがって，コンパイラ部分は移植のしやすさとか変更のしやすさとかが重要になる。

コンパイル作業を図3.4.1に示す基本操作の列で記述し，threaded code 方式で処理した。基本操作はほとんどがアセンブラ語で数行でおさまる。名前登録用基本操作は SL45 で書いてあればよかった。また，scan 用とメッセージ用は I/O を使うのである程度大きくなるのは仕方ない。

		アセンブラ語で
オブジェクト生成	65行	EMIT() EMITB OBJPTR OBJPTRX() OBJSET INITOBJ OBJST
スタック操作	150行	INITST PU PDQ() PD PU2 PDQ2() PD2 INCR MAX ADDB DECR X SAV RCV CMP() CMPB CMPS C
名前登録	200行	VAR LABEL LITERAL FNAME DEFLABEL NOTEND REFLABEL
scan	180+20行	GETSKIP INITIN GETCHAR GETLBL GETLIT
メッセージ	100+20行	INITOUT ENDMES ERRPRNT
その他	40行	STNO STARTP SETSTART FIN ERROR() NOTFIN EOC

図3.4.1 基本操作　　機械依存部分

3.5 サポートルーチン群の移植

機械依存性の強い3ルーチン，INPUT，OUTPUT，REFTBL（ハッシュで表を引く）は以下で説明する方法ではなく，直接370アセンブラ語で書き下した．合計で150行ほどになった．

SL45は45S従属言語であるからそのまま370アセンブラ語に変換するには無理がある．そこで，SL45と370に共通な記述言語を設定し（便宜上SL45Eと呼ぶ）SL45の記述をSL45Eに変える作業とSL45Eから370アセンブラ語を生成するコンパイラを作成する作業とに分けて移植をおこなうことにした．SL45とSL45Eの文法概略を付録に示す．なおここでいうSL45はサポートルーチン群を記述するのに必要な最小構成にしぼって考えている。

3.5.1 SL45版をSL45E版に変更する作業

統一的におこなえる変更は，SNOBOL3のプログラムでおこなった．個別な変更はテキストエディタを使っておこない，ユーティリティFCOMP（ファイルどおしの比較をし，異なっている行を印刷するプログラム）によって変更が意図どおりおこなわれたかを検証した．

1)リテラルの意味付けプログラム

SL45のリテラルには，バイト単位のインデックスとして使われるもの（BI型），語単位のインデックスとして使われるもの（WI型），整数として使われるもの（I型），そしてこれら以外の目的として使われるもの（ビットパターンを表わすBP型）があると考えられる．これはSL45の変数がとる値でもある．ところが字面上はこれらの型は何ら区別されてない．この点が変更作業中最大の困難となった．それは，45Sと370とのアドレス指定の違いによっている．

45Sのインデックス修飾は命令の扱うデータの大きさを単位としているが，370では一様にバイトを単位としている．したがって，図3.5.1.1.aに示すSL45プログラ

```
R4=B;
R2=1;
A(R2)=R4;
```

図3.5.1.1.a

```
R4=B;
R2=1WORD;
A(R2)=R4;
```

図3.5.1.1.b

```
R4=B;
R2=1;
A(R2)=RB4;
```

図3.5.1.2.a

```
R4=B;
R2=1BYTE;
A(R2)=RB4;
```

図3.5.1.2.b

ムにおけるリテラル1は45Sでは1でよいが370では4として扱わなければならない．また，図3.5.1.2.aでは370でも1のままでよい．これらの違いはリテラルの型の違いによっているので，リテラルに型表記をつけて，図3.5.1.1.b，図3.5.1.2.bに示すように区別することにした．この作業はリテラルの型認識を必要とする．

リテラルの型はデータフロー解析によってかなり認識することができる．図3.5.1.3に示す例では，R1がワードインデックスを扱っているので，それにロードされたリテラル1の型はWIであることが判定できる．また図3.5.1.4に示す例では，変

```
R1=1;
A(R1)=R4;
```

図3.5.1.3

```
R1=P;
A(R1)=R4;
R2=P+1;
```

図3.5.1.4

数Pの型は前例の考察によってWIであることがわかり，Pとリテラル1とが同じ型を持つことから，リテラル1の型はWIであることがわかる．しかしながら100%

認識できるわけではない．必要最小限の手作業をほどこすことになる．

リテラルの意味付けプログラムは SNOBOL3 で書かれている（約600行）．SL45 プログラムを入力し，リテラルに型表示をつけて出力する．処理は2パスからなっている．1パス目にデータフロー解析をおこなって変数とリテラルのクラス分けをし（チェックリストとして印刷），2パス目にそれに従って型表示を付ける．ある変数が2つ以上の型で扱われている場合（チェックリストをみることでわかる）は，その変数名を外から与えることで正しい型認識をおこなった．以下に実例を示す．

```
DCL A(10)BIN(16);                    DCL A(10)BIN(16);
DCL B(10)CHAR;                       DCL B(10)CHAR;
DCL C BIN(16);                       DCL C BIN(16)WORD;
DCL D BIN(16);                       DCL D BIN(16)BYTE;
R1=0;                                R1=0WORD;
R2=0;          リテラルの意味付け→    R2=0BYTE;
R4=0;                                R4=0;
DO R3=1 TO 10 BY 1;                  DO R3=1 TO 10 BY 1;
A(R1)=R4;                            A(R1)=R4;
B(R2)=R4;                            B(R2)=R4;
C=R1;                                C=R1;
D=R2;                                D=R2;
...                                  ...
R1=C+1;                              R1=C+1WORD;
R2=D+1;                              R2=D+1BYTE;
END;                                 END;
```

2). PMACの持ち上げ

SL45には PMAC（procadure macro）といってアセンブラ語を直接書ける構文がある．これをSL45E の構文で記述しなければならない．この変換もSNOBOL3 のプログラムを組んでおこなった．

PMACとして現れたアセンブラ語のうち，SL45のその他の構文で記述できるものはそれに置き換え，どうしても置き換え得ないものは PMAC(…) の代りに &(…) で表現した．このようにして導入された&構文の第1引数は命令を表している．それを以下に列挙する．数字は出現頻度を示す．

SKIP	4		MBN	6	
T	4	…変換命令	MOVEL	6	
MOVE	3	…移動命令	SRBE	3	…探索命令
MV	19	…移動命令	JUMP	20	

&(<命令>,<オペランド>) の<命令>を示した．

3). 単純置き換え

パターンの統一的置き換えだけですむ変更作業は SNOBOL3 で書いた．変更内容を以下に示す．

① 変数の属性の変更…ポインタをアドレスで一律に表現することにしたので，今までの間接属性を変更しなければならない変数が13個あった．

② ADDL #8000の変更…45Sの語の第0ビット目を消すのに ADDL #8000 と書いていたのでこれを EOR #8000 に直した．

③ 型変換の持ち上げ…ADDL 1 SHLR 1 はバイトインデックスをワードインデックスに換えることを意味している．これを &(CWORD) に書き直した．同様に

&(CBYTE)を導入した．これは「SHLL 1」に対応している．この置き換えをすると型変換でないシフト命令が誤って置き換えられることもあるだろうが，それは手作業で修正する．

以上の1)，2)，3) はプログラムを使った変更作業である．次に，手作業でやらざるを得なかった変更について述べる．

4). 手作業による変更

①リテラルの持ち上げ

1). でうまくいかなかった部分の書き換え．リスティングを上から下まで逐次ながめて修正した．10件あった．

②判定条件の修正

たまたまサインビットにあたるビットのon-off 判定に正負を使っている場合は，そのビットがon-off という条件に書き換えた．

③プリミティブルーチンの入口・出口

プリミティブルーチンも他の手続き同様 PROC … END で書かれている．今回プリミティブルーチンはthreaded code方式で使われるので他の手続きと区別しなければならない．そこで入口を PROCP(n)，出口を RETURNP と書くことにした．ここでnは引数の個数を示す．

④&構文の持ち上げ

&(JUMP,…)には意味を考えると CALL，RETURNP，RETURN，GOTO の4種類がある．はじめの3種類については それぞれ &(JSB,…) &(PC,…) &(RTN,…) と書き直した．また45Sの事務処理命令を表わしている &(MV,…)&(T,…)&(SRBE,…)も，その意味をより表わした構成に書き直した．

⑤その他

1語の大きさに由来した定数の書き換え．

3.5.2 SL45Eコンパイラ

SL45Eを370アセンブラ語におとすコンパイラを作成した．SNOBOL3で書いた3つのプログラム（325行，783行，12行）からできている．SL45E の構文解析は単純であるから，コード生成についてのみ述べる．

1). コード生成の方針

45Sの命令1個につきなるべく370命令1個を対応させるように努めた．

①レジスタの使いかた

370のレジスタの使いかたを以下に示す．

R0－R7	… 45SのR0－R7にあてる
R8－R12	… 作業用
R13	… プリミティブルーチン用driverのbase
R14	… baseレジスタ保存用
R15	… entry point 用

②アドレス指定

SL45Eの手続きの先頭番地をbaseにして（R15を利用）その手続き内のアドレス

を指定する。したがって，手続き呼び出しと戻りにおいてbaseのset, resetをおこなう。この際にR14を利用する。

間接アドレス指定，負のoffset指定，外部名の指定には，作業用レジスタを利用する。

2). 実例

```
DCL OUTPUT BIN(16)EXT;
DCL TRC BIN(16)WORD EXT;
DCL STACKTTP BIN(16)WORD EXT;
DCL GETCLN3S EXT ENT;
DCL TOTO BIN(16)WORD;
DCL FLAG(100)BIN(16)DEF 0;
DCL VFIELD(100)BIN(16)DEF 3;
DCL STACK(100)BIN(16)DEF 0;
DCL OUTBIT BIN(16)DEF 8;
DCL OUTPUT ENT EXT;
  ..............
ASSIGN: PROCP;


&(IN); &(MEAS,3); TRC=R0;
STACKTTP=R1;

R1=R1 SUBL 3WORD;
ASTV=R1;
  ..............
IF R6=1 THEN GOTO ASS5P;

CALL GETCLN3S;

TOTO=R3;

R3=2WORD;&(CBYTE);
  ..............
ASS3P: R3=STACKBM;

R3=STACK(R3+6);
VFIELD(R3)=R2;
R4=FLAG(R3);
IF OUTBIT THEN CALL OUTPUT;




&(N,3); RETURNP;
END ASSIGN;
```

入力プログラム

```
         EXTRN    OUTPUT
         EXTRN    TRC
         EXTRN    STACKTTP
         EXTRN    GETCLN3S
TOTO     DS       1F
FLAG     EQU      0
VFIELD   EQU      12
STACK    EQU      0
OUTBIT   EQU      8
         EXTRN    OUTPUT
         ..................
ASSIGN   DS       0F
         USING    *,R15
         ST       R14,SYSA99
         ST       R0,TRC
         L        W1,=A(STACKTTP)
         ST       R1,0(W1)
         S        R1,=F'12'
         ST       R1,ASTV
         ..................
         C        R6,=F'1'
         BZ       ASS5P
         LR       R14,R15
         L        R15,=A(GETCLN3S)
         BALR     R0,R15
         ST       R3,TOTO
         LA       R3,8
         ..................
ASS3P    L        W1,=A(STACKBM)
         L        R3,0(W1)
         L        R3,0+24(R3)
         ST       R2,0+12(R3)
         L        R4,0(R3)
         L        W1,=A(B'10000000')
         NR       W1,R4
         BZ       SYSI102
         LR       R14,R15
         L        R15,=A(OUTPUT)
         BALR     R0,R15
SYSI102  BAL      R0,CPUP
         LTORG    ,
```

生成されたコード

4. 作業量

コンパイラ部の移植は，新たに書き直しておこなったので，コンパイル作業を基本操作で記述するのに2週間，基本操作のコーディングに1週間，デバッグに1週間，合計で約1月もかかった。今後は基本操作の移植だけでよいので短期間に移植できるようになるだろう。

サポート部の移植は，リテラル意味付けプログラム作成に2週間，SL45Eコンパイラ作成に2週間，あとは手作業で，合計約1月位になる。

移植作業には輸入と輸出とがあるが，今回のは輸出作業であり，移植先の機械にあまりなれていないことからの困難が多かった。

5. 考察

○ SNOBOL3 処理系の構造について

今回の移植作業は，同系統の計算機間のものなので割合すっきりとおこなわれた。べたづめ方式がどの計算機においても有効であるかどうかは，今後のテスト結果によっている．移植という関点からながめると，べたづめ方式という単純な構造は移植しやすいことが実証されたと思う．

○処理系の記述のしかたについて

処理系がSL45で書かれていたので，プログラミングの際にはあったがコーディング中に落ちてしまった情報を再生する持ち上げ作業が必要になった．移植を前提にしてSL45で書いたのなら情報はあまり落ちなかったと思う．でもそれはコーディングしやすさをかなりそこなうだろう．言語の制約に従って書けば必要な情報は落ちないといった言語があればよかった．

370 アーキテクチャーには 455 のそれを含む部分が多いので，たとえばレジスタアロケーションはそのまま利用できるなど，低レベル言語のクロスコンパイラを容易に作ることができた．しかし一般的にみれば，機械の特徴を生かすといっても，効率をひどく落さないぎりぎりはできるだけ高級言語で記述したい．移植作業とは記述言語のコンパイラを作ることになるので，記述言語はできるだけ移植しやすく，つまり単純にしたい．そのことは移植されるもの（処理系）に依存した記述言語という考えを導き出す（MACROを使った方法など）．結局のところは，移植されるものに依存した記述言語を作り，それを別の記述言語で書く．その記述言語は機械独立部分と依存部分とを書き分けられる、といった構成がうまくいくのではないかと思う．

今回SL45で苦労したが，これがアセンブラ語だったらお手上げだった．少しでも高級な言語を使うことはよいことである．

○移植方法について

なるべく手作業をはぶくという精神のもとに，クロスコンパイラを作るという方式で移植をおこなった．リテラル表味付け作業を考えてみると，プログラムを全部ながめて手で修正すればすむはなしだ．それをわざわざプログラムを組んでおこなったのは，虫を入れずに移植することに重点を置いたからである．また，ハンドコンパイルでやった方が労力は少なかったかもしれないが，455から370への移植路線が引けたことを考えるに，この方法の方が今後役に立つと思う．

○今後の課題

SL45E版のSNOBOL3 処理系を逆コンパイルするかたちでより高級な言語で記述すること，それにSL45Eのコンパイラのオブジェクトをオプティマイズすることが今後の課題である．

謝辞

本研究をするにあたって，前野年紀助教授，白濱律雄氏には多大の御支援を頂いた．あつく感謝したい．また本研究にはユニコンを利用した．

文献

(1) 角田博保：べたづめ方式の SNOBOL3 処理系における性能測定，第16回プログラミングシンポジウム報告集，pp. 197-207，1975年1月9-11日，情報処理学会．

(2) Farber, D.J., Griswold, R.E., and Polonsky, I.P. : The SNOBOL3 Programming Language, The Bell System Technical Journal, July-August, 1966.

(3) Griswold, R.E., Poage, J.F., and Polonsky, I.P. : The SNOBOL4 Programming Language, Prentice-Hall, Englewood Cliffs, N.J., 1968.

(4) 富士通："FACOM 230-45S ハードウェア総合解説編"，EX-011-1-2．

(5) IBM ："IBM System/370 Principles of Operation"，GA22-7000-3．

付録． SL45 文法と SL45E 文法の概説

SL45文法のうち SN3/45S を記述するのに使われている部分のみを示す．SL45は PL/I 流の文言語である．

```
<SL45プログラム> ::= <手続き宣言>
<手続き宣言>     ::= <id> : PROC ; [<宣言> ;]ooo ( [<ℓ> :] <文> ; )ooo END <id> ;
<宣言>           ::= <手続き宣言> | DCL <id> [( <num> )] <atrib> ooo
<文>             ::= IF ( <reg> <bop> <term> | <cc> | <term> ) THEN [<ℓ> :] <単純文> |
                     DO <load文> TO <term> BY <term> ; ( [<ℓ> :] <文> ; )ooo END |
                     DO WHILE ( <reg> <bop> <term> ) ; ( [<ℓ> :] <文> ; )ooo END |
                     SWITCH ( <reg> ) <ℓ> {,}ooo | <単純文>
<単純文>         ::= <load文> | <store文> | <machine-dep文> | GOTO <ℓ> | RETURN <id> |
                     CALL <id> [( (<id> | <num>) {,}ooo )]
<load文>         ::= <reg> = (<reg> | <term>) [<op> <term>] ooo
<store文>        ::= <idp> = <reg>
<machine-dep文>  ::= MOVE3( <id> | <num> , <idp> , <idp> ) | PMAC( <id> [, <operand>] )
<idp>            ::= <id> | <id> ( <reg> [(+|-) <num>] ) | <id> ( [+|-] <num> )
<atrib>          ::= ENT | EXT | GLB | DEF (<num> | <id> [( <num> )]) | BOUN( <num> ) | BASED( <id> ) |
                     BIN( <num> ) | CHAR | INIT( <literal> {,}ooo )
<term>           ::= <idp> | <literal>
<literal>        ::= <num> | " <string> " | # <hex>
<reg>            ::= R0 | R1 | R2 | R3 | R4 | R5 | R6 | R7
```

<op> は operation
<bop> は boolean operation
<cc> は condition code

SL45E 構文は，上に示した構文に，① <手続き宣言> に PROCP を追加，② <atrib> に WORD/BYTE/ANY を追加，③ <literal> に <num> WORD / <num> BYTE を追加，④ <machine-dep文> から PMAC 構文を除き <&構文> を追加することで得られる．

```
<&構文> ::= &( <命令> [, <operand>] )
<命令>  ::= PC | JUMP | RTN | JSB | SUBTM | SETTM | MEAS | WTO | N | OUT | CWORD | CBYTE |
            IN | SKIP | MBN | STOP | MOVE | MOVEL | TRT | SRCH
```

注) なお構文の記述には AN 記法を使った．