# 実行回数計数機能を追加したSNOBOL4処理系とその移し換えについて†

吉田和幸†† 牛島和夫††

ソフトウェア開発の初期の段階においてプロトタイプを組み立てて実際に動かしてみて問題点を認識する作業は有用である．開発中のソフトウェアが文字列データの処理を中心とする場合などにはSNOBOL4を使うと容易にそのプロトタイプの構築ができる．一方，プログラム中の各文の実行回数を知ることはプログラムの開発や改善に役立つ．SNOBOL4によるプロトタイプ作成を助けるために，既存のSNOBOL4処理系に実行回数計数機能を追加した．原著者の許可を得て入手したこの処理系は，SIL (SNOBOL4 Implementation Language) という抽象言語で書かれたもの (OS 360用) である．これをFACOM OS Ⅳ/F4のもとに移し換える際に各文の実行回数を計数してソーステキストと並べて表示する機能を追加した．さらにこの処理系をIBM VM/370 CMSとHITAC VOS 3のもとに移し換えた．本論文では，簡単なプロトタイプ作成の実例をとおして実行回数計数機能の評価を行う．さらに実行回数計数機能の実現の概要について述べ，機能追加作業が比較的容易に行えた理由をSIL設計の観点から考察する．最後に実行回数計数機能追加版SNOBOL4の移し換えについて述べる．

## 1. はじめに

ソフトウェア開発の初期の段階においてプロトタイプを組み立てて実際に動かしてみて問題点を認識する作業は有用である．解くべき問題領域が文字データ処理を中心とする場合などにSNOBOL4を使うと容易にプロトタイプの構築ができる[1)-3)]．一方，プログラム中の各文の実行回数を知ることはプログラムの開発や改善に役立つ．SNOBOL4によるプロトタイプ作成を助けるために，既存のSNOBOL4処理系に実行回数計数機能を追加した．原著者の許可を得て入手したこの処理系は，SIL (SNOBOL4 Implementation Language) という抽象言語で書かれたもの (OS 360用，以下SIL版SNOBOL4と呼ぶ) である[4)]．これをFACOM OS Ⅳ/F4のもとに移し換える際に各文の実行回数を計数してソーステキストと並べて表示する機能を追加した[5)]．さらにこの処理系をIBM VM/370 CMSとHITAC VOS 3のもとに移し換えた[6),7)]．

本論文では，2章で簡単なプロトタイプ作成の実例をとおして実行回数計数機能の評価を行う．3章では実行回数計数機能の実現の準備としてSIL版SNOBOL4とそれを記述しているSILシステムの構成について簡単に述べる．4章では実行回数計数機能の実現について述べ，機能追加作業が比較的容易に行えた理由をSIL設計の観点から考察する．5章では実行回数計数機能追加版SNOBOL4の移し換えについて述べる．

## 2. 実行回数計数機能の評価

SNOBOL4実行回数計数機能の使用例を**図1**に示す．SNOBOL4では各文ごとにその実行の成否を確かめ，それによってプログラムの流れを変えることができる．図の右に各実行文の実行回数，成功回数，失敗回数が出力されている．このプログラムはFORTRANのソーステキストを入力し，それを構成するプログラム単位の名前とその区切りを表示するものである[8)]．図1(a)から入力となったFORTRANプログラムの総行数が1,639行であったこと，(b)からコメント行と継続行を除いた行が987行あったこと，(c)からEND行が30行あったこと，(d)からサブルーチン文等が合計29あったことがわかる．

このプログラムの実行には実行回数を計数して約5.5秒かかる (FACOM M200 OS Ⅳ/F4のもとで)．1600行余のプログラムを処理するための5.5秒は長すぎるので実行回数情報を手掛かりに実行時間の短縮を試みる．(b),(d)に対応する実行文はそれぞれ1行で多くの仕事をしようとして複雑なパターンを使っている．しかもこれらの文は，(b)が1,639回実行され，(d)も929回実行されている．そこでそれぞ

† SNOBOL4 with Profiling Facility and Its Transferability by KAZUYUKI YOSHIDA and KAZUO USHIJIMA (Department of Computer Science and Communication Engineering, Faculty of Engineering, Kyushu University).

†† 九州大学工学部情報工学科

```
FACOM OSIV/F4 SNOBOL4(BELL TELEPHONE LABORATORIES) -750519-  VERSION 3.11(PROFILE-790327-) DATE 83/01/14    TIME 20:33:47

NO.                 SOURCE  STATEMENT                                                          EXECUTION     SUCCESS      FAILURE
        *       LIST THE LINE NUMBER OF EACH FORTRAN PROGRAM UNIT                          00000010
        *                                                                                  00000020
1               LINENO = 0                                                                 00000030          1            1            0
2               LINE   = '**MAIN**'                                                        00000040          1            1            0
3               STARTLINE = 1                                                              00000050          1            1            0
4               OUTPUT =                                                                   00000060          1            1            0
5               OUTPUT = '   FROM     TO  PROGRAM NAME'                                    00000070          1            1            0
6               OUTPUT =                                                                   00000080          1            1            0
        *                                                                                  00000090
7       INPUT   BUF = INPUT                                    :F(END)                     00000100       1640   (a)   1639                1
8               LINENO = LINENO + 1                                                        00000110       1639         1639            0
9               BUF LEN(72) . BUF                                                          00000120       1639         1639            0
10              BUF (POS(0) 'C' ABORT) |                                                   00000130       1639   (b)    987          652
10      +           (POS(5) (' ' | '0'))                   :S(ST)F(INPUT)                  00000140
11      ST      BUF ' FORMAT'                              :S(INPUT)                       00000150        987          28          959
12              BUF '    END    '                         :S(OUTPUT)                      00000160        959   (c)    30          929
13              BUF (' SUBROUTINE' | 'FUNCTION' |                                          00000170        929   (d)    29          900
13      +           ' BLOCK ')                             :S(PROG)F(INPUT)                00000180
        *                                                                                  00000190
14      PROG    LINE = BUF                                                                 00000200         29           29            0
15      DLTBLK LINE FENCE ' ' =                            :S(DLTBLK)F(INPUT)              00000210        203          174           29
        *                                                                                  00000220
16      OUTPUT OUTPUT = ' ' DUPL(' ',6 - SIZE(STARTLINE)) STARTLINE                        00000230         30           30            0
16      +                 ' ' DUPL(' ',6 - SIZE(LINENO))    LINENO                         00000240
16      +                 '  ' LINE                                                       00000250
17              STARTLINE = LINENO + 1                                                     00000260         30           30            0
18              LINE = '**MAIN**'                          :(INPUT)                        00000270         30           30            0
        *                                                                                  00000280
19      END                                                                                00000290          1            1            0
NO ERRORS DETECTED IN SOURCE PROGRAM
```

**図 1** 実行回数計数機能使用例
Fig. 1 SNOBOL4 program source list with a profile.

れ簡単なパターンを使った複数の文に置き換えて以前と同じ入力データを与えたものを実行回数情報つきで**図 2** に示す．図 2 のプログラムの実行時間は約 2.8 秒となりほぼ半分に短縮された．

FORTRAN の文は 72 ケタすべてを使うことはまれで，1 行の右の方は空白であることが多い．そこで図 2 のプログラムの入力部で SNOBOL4 の TRIM 機能を用いて 1 行の右側の空白を取り除き，パターンマッチの際の探索範囲を狭めた（**図 3** 参照）．このプログラムの実行時間は約 1.8 秒となり，図 2 のプログラムに対して実行時間が約 60% に短縮した．**表 1** にこの 3 個のプログラムの実行時間を示す．表 1 では実行回数を計数した場合とともに計数をしない場合の実行時間，およびそれらの比を合わせて示している．実行回数を計数した場合の実行時間の増加は 1 割以内に収まっていることがわかる．さらに表 1 には，改善の

```
FACOM OSIV/F4 SNOBOL4(BELL TELEPHONE LABORATORIES) -750519-  VERSION 3.11(PROFILE-790327-) DATE 83/01/14    TIME 20:34:48

NO.                 SOURCE  STATEMENT                                                          EXECUTION     SUCCESS      FAILURE
                                                                (中 略)
10              BUF FENCE 'C'                              :S(INPUT)                       00000160       1639          586         1053
11              BUF FENCE LEN(5) ' '                       :S(ST)                          00000170       1053          987           66
12              BUF FENCE LEN(5) '0'                       :S(ST)F(INPUT)                  00000180         66            0           66
13      ST      BUF ' FORMAT'                              :S(INPUT)                       00000190        987           28          959
14              BUF '    END    '                         :S(OUTPUT)                      00000200        959           30          929
15              BUF ' SUBROUTINE'                          :S(PROG)                        00000210        929           24          905
16              BUF 'FUNCTION'                             :S(PROG)                        00000220        905            4          901
17              BUF ' BLOCK '                              :S(PROG)F(INPUT)                00000230        901            1          900
                                                                (後 略)
```

**図 2** パターンマッチ部分の改良版（一部分）
Fig. 2 Improved program using simpler pattern matches (part).

```
FACOM OSIV/F4 SNOBOL4(BELL TELEPHONE LABORATORIES) -750519-  VERSION 3.11(PROFILE-790327-) DATE 83/01/14    TIME 20:35:10

NO.                 SOURCE  STATEMENT                                                          EXECUTION     SUCCESS      FAILURE
                                                                (中 略)
7               &TRIM = 1                                                                  00000120          1            1            0
        *                                                                                  00000130
8       *       INPUT('SYSPIT',5,72)                                                       00000140          1            1            0
        *                                                                                  00000150
9       INPUT   BUF = SYSPIT                               :F(END)                         00000160       1640         1639            1
                                                                (後 略)
```

**図 3** TRIM 機能使用による改良版（一部分）
Fig. 3 Improved program using TRIM-facility (part).

表1 各プログラムの実行時間
Table 1 Execution time of programs in Figs. 1-3.

| プログラム | 実行回数計数(P) [ms] | 計数なし(NP) [ms] | 改善比(NPにおける) | オーバヘッド(P/NP) |
|---|---|---|---|---|
| 図1 | 5552 | 5430 | 1 | 1.022 |
| 図2 | 2866 | 2753 | 0.507 | 1.041 |
| 図3 | 1771 | 1665 | 0.307 | 1.070 |

程度を明白にするため，実行回数を計数しない場合について，図1のプログラムを1（基準）とした図2および図3のプログラムの改善比を記入している．

図1のプログラムは「FORTRAN ソーステキスト中からサブルーチン等を切り出したい」という要求[8]があったときに30分ほどで作成したプログラムである．SNOBOL4 は挿入，削除を伴ったパターンマッチを1文で書け，その組合せにより文字列に関するかなり複雑な操作も簡単に記述できる．したがってテキストファイル処理などに際して適当なツールがない場合，その場でプログラムを作ってみるというラピッドプロトタイピングに適した言語である[3]．このような機能のツールを頻繁に使用する場合には他の言語を用いて高速版を作り直せばよい．しかし1.8秒ならばSNOBOL4 版でも十分実用になる．この場合 SNOBOL4 の実行回数計数情報は，入力データに関するドキュメントとして利用でき，作成したばかりのSNOBOL4 プログラムの正しさを確かめる手掛かりになる．

SNOBOL4 における実行回数計数機能の使用上の効用をまとめる．

（1） 作成したプログラムの動作の客観的認識が可能である．

（2） 採用したアルゴリズムの適否を実行回数から判定可能である．

（3） 実行個所と非実行個所が一目瞭然なので，プログラムの検査に役立つ．

（4） 実行回数がプログラムの入出力テキストに関する簡単なドキュメントとして利用できる．

（5） 実行回数を手掛かりにプログラムの効率改善に役立つ．

（6） 実行回数計数用のカウンタを中間コードであるプレフィックスコード（次章で説明する）の形で挿入するため実行回数計数に要する時間の増加はわずかである．

## 3. SIL 版 SNOBOL4 処理系

実行回数計数機能の追加作業を説明する準備として，本章では既存の SNOBOL4 処理系の構成を簡単に述べる．

SNOBOL4 処理系はトランスレータ，インタプリタ，記憶管理，システムインタフェースという四つの部分から構成される[4]．トランスレータは SNOBOL4 プログラムをプレフィックスコードに変換し，それをインタプリタが解釈実行する．記憶管理とシステムインタフェースはトランスレータとインタプリタの両方から使われる．記憶管理は領域の割当て，データの記憶，記憶域の再構成等を行う．システムインタフェースは入出力やプログラム割込み等を受け持つ．

トランスレータが生成するプレフィックスコードはディスクリプタという単位からなっており，それは値部，フラグ部，番地部をもっている．たとえば

L1 AA = BB : S(L2)

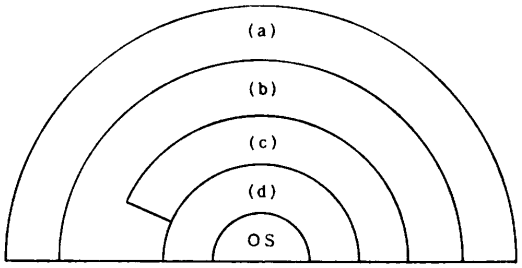
という文は**図4**のプレフィックスコードに変換される．フラグ部がFのディスクリプタは関数を表し，その値部が引数の個数を，番地部が関数の入り口番地をそれぞれ表す．フラグ部がAのディスクリプタはこの例では変数名（ラベル名）をさすポインタを番地部にもつ（文字列等の実際の値を指すこともある）．init の引数のディスクリプタは番地部と値部だけが意味をもつ．番地部には文の番号が入り，値部にはその文の実行が失敗したときに読み飛ばすディスクリプタの数（失敗オフセット，ここでは7 d）が入っている．

SIL 版 SNOBOL4 処理系を記述している SIL という抽象言語は**図5**のような階層構成により実現されている[4]．最も外側（図5（a））が SIL の世界で131個の命令をもつ．その内側（図5（b））が131個の命令をアセンブラのマクロ機能を用いて定義した SIL

| | 値部 | フラグ部 | 番地部 |
|---|---|---|---|
| ↑ | 0 | F | ⟶base |
| | 1 | F | ⟶init |
| | 7 d | 0 | *n*（行番号） |
| | 2 | F | ⟶= |
| | S | A | ⟶{AA} |
| | S | A | ⟶{BB} |
| | 1 | F | ⟶goto |
| ↓ | S | A | ⟶{L2} |

プレフィックスコード

**図4** 文 "L1 AA = BB : S(L2)" を変換したプレフィックスコード
Fig. 4 A SNOBOL4 statement "L1 AA = BB : S (L2)" is translated into the sequence of prefix codes.




(a) SIL 言語, (b) SIL マクロ定義, (c) IO ルーチン, (d) OS のマクロ命令, FORTRAN 実行時ルーチン

**図 5** SIL 処理系の階層構成

Fig. 5 Hierarchical structure of SIL processor.

マクロの層である．その内側に入出力等 OS に依頼する部分を IO ルーチンとしてまとめた層がある（図5(c)）．これらのルーチンはアセンブリ言語で記述され，SIL マクロから呼び出される．実際の入出力は FORTRAN の実行時ルーチンに処理を依頼し，異常終了時処理，時間管理等の OS に関係する部分は直接 OS のマクロ命令を用いているものもある．これらが最も内側の層を構成する（図5(d)）．

## 4. 実行回数計数機能の実現

各文の実行回数を計数するにはソースプログラムを解析し，そこに

count(k) := count(k)+1

という形式のカウンタを挿入する方式（前処理／後処理方式）をとることが多い．FORDAP, COBOLDAP, PASDAP 等はいずれもこの方式をとっている[9]．しかし SIL 版 SNOBOL4 処理系は解釈実行方式をとっているため，この方法を採用すると挿入されたカウンタも解釈実行され計数のための CPU 時間，記憶場所の増加が著しい．しかもこの方法では成功回数等を計数することは困難である．このため SNOBOL4 の実行回数計数機能はカウンタをソースコードではなくプレフィックスコードの形で挿入する方式をとることにした．

実行回数計数機能を実現するために SIL 版 SNOBOL4 に次の追加修正を加えた．

（1） カウンタの挿入 トランスレータが SNOBOL4 ソースプログラムを変換するときに，文の実行の直前（図4の init 関数呼び出し前）に実行回数を計数する関数 count を呼び出すプレフィックスコードを，文の実行部と分岐部の間に成功回数を計数する関数 scount を呼び出すプレフィックスコードをそれぞれ挿入するようにした（このためのトランスレータの書き替えは SIL 言語で22行）．図4と同じ文を変換してカウンタを挿入したプレフィックスコードの例を**図6**に示す．

| | 値部 | フラグ部 | 番地部 |
|---|---|---|---|
| ↑ | 0 | F | ⟶base |
| | 1 | F | ⟶count |
| | 0 | 0 | *n* |
| | 1 | F | ⟶init |
| | 11 d | 0 | *n* |
| 11 d | 2 | F | ⟶= |
| | S | A | ⟶{AA} |
| | S | A | ⟶{BB} |
| | 1 | F | ⟶scount |
| | 0 | 0 | *n* |
| | 1 | F | ⟶goto |
| ↓ | S | A | ⟶{L2} |

**図 6** 文 "L1 AA = BB : S(L2)" を変換してカウンタを挿入したプレフィックスコード

Fig. 6 A SNOBOL4 statement "L1 AA = BB : S (L2)" is translated into the sequence of prefix codes including two counters.

（2） ソーステキストの処理 実行回数をソーステキストと並べて表示するために，ソーステキストを一時ファイルに蓄える．トランスレータ中のソースリストを出力する部分を利用し，出力先を一時ファイルにすればよい．SNOBOL4 ではセミコロンで区切って1行に複数の文を書くことを許している．そのような場合はセミコロンで分割して1行1文として保存するようにした．

（3） カウンタ領域の確保 プレフィックスコードへの変換後，解釈実行に入る前にカウンタ領域を確保するようにした．このため SIL 言語にカウンタ領域を確保する命令を1個追加し，これに対応して領域の確保を OS に実際に依頼するサブルーチンを IO ルーチンに加えた．

（4） count, scount の追加 トランスレータが挿入したプレフィックスコードを実行するために count, scount の二つの関数をインタプリタに追加した（SIL 言語で25行）．

（5） 結果の出力 解釈実行後，ソーステキストとそれに対応する実行回数，成功回数，失敗回数とを出力する．ソーステキストの分割等の処理は(2)で処理済みであるので，ここでの処理はソーステキストを読み，ソーステキストとそれに対応する実行回数，成功回数，失敗回数を並べて出力することである（SIL 言語で61行）．

（6） 実行回数計数機能使用の有無 計数機能を処理系に組み込んだので TSS コマンド（あるいはジョブ制御文）のパラメータで使用の有無を選択できるよ

うにした．このために実行時パラメータを判定する部分を書き直した．

（7） 異常終了時の処理 実行回数計数中にプログラムが異常終了した場合，それまでに集めた実行回数情報はプログラムのデバッグ等に有用である．SIL版SNOBOL4ではオーバフロー，記憶保護例外等のプログラム割込みが起こった場合に，それまでの実行のサマリ（CPU時間，入出力行数等）を出力できる．実行回数計数機能追加版SNOBOL4ではプログラム割込み時に実行回数情報を出力するほか，CPU時間切れ，TSS使用時の端末割込み時にも実行回数情報を出力できるようにした．これを実現するためにトランスレータの実行に先立って1回呼び出されるSILマクロINITと終了処理を行うIOルーチンFINISとの一部を書き替えた．すなわち，異常終了後ただちに後処理部に制御を移す指示をOSに登録することをINITで行わせ，その指示の解除をFINISが行う．CPU時間切れ時の処理に関してはスーパバイザマクロを用いて直接OSに依頼している．端末割込み時の処理は，FORTRANの実行時ルーチンを利用している．なお，このINITは図5(b)に属するSIL言語の定義の一部であり，図4に現れるSIL言語で書かれたinit関数（図5(a)に属する）とは別のものである．

これらの追加修正の量をまとめる．

・ (1),(2),(4),(5)はSIL言語（図5(a)）で書かれた部分（全体で約6,800行）の修正である．修正量は合計で約280行である．

・ (3)の前半はSILマクロ定義（図5(b)）の追加である．この部分は6行である．また(7)の前半はSILマクロ定義の変更である．このためにSILマクロINITの定義が3行増えた．

・ (3),(7)の後半と(6)はIOルーチン（図5(c)）での修正である．IOルーチンはモジュールの集りであり，それらのモジュール間での副作用はない．ここでの修正はモジュールの置き換え，追加であり，IOルーチン全体で2,260行のうち，修正量は160行である．

SIL版SNOBOL4は約10年前に設計実現されたにもかかわらず，実行回数計数機能の追加作業をこのように比較的容易に遂行できたのは，ドキュメントがよく整備されていたこと，SILシステムの設計が明快でよく機能分割されていたことが大きな理由と考えられる[4]．以下に要点をまとめる．

（1） SILシステムの構成 3章にまとめたように階層的に構成されており，機械またはOSに独立な部分と依存する部分が明確に分離されている．したがって一部の修正がプログラムの広域に影響を及ぼすことがなかった．

（2） データの表現 SILプログラム中のデータ構造はディスクリプタで表現する．実行回数計数機能のために加えたカウンタ領域等のデータ領域もディスクリプタを用いて表現できるので，実行回数計数機能のための追加部分のほとんどをSIL言語で記述できた．

（3） プレフィックスコード プレフィックスコードの構造やSNOBOL4ソーステキストからの変換の規則がドキュメント[4]から容易にわかるので，カウンタの挿入場所の決定が容易にできた．

（4） SIL言語の定義 SIL言語の文法に関するドキュメント[4]によりSIL言語の修得が容易であった．そのため(2)で述べた追加部分のSIL言語での記述は容易であった．さらにSIL言語の拡張の際にも既存のSILの機能と矛盾することなく拡張できた．

（5） IOルーチンの仕様 外部仕様がドキュメント[4]によりはっきりわかっており，IOルーチン間の副作用がないので書き替えが必要なルーチンは容易に書き替えられた．さらにSILで書かれたSNOBOL4本体と連絡する名前（サブルーチン名，大域変数名等）が明示されているので，作業変数等の追加変更が自由にできた．

## 5. 実行回数計数機能追加版の移し換え

SIL版SNOBOL4は現在40以上の計算機[10]で稼働している．実行回数計数機能追加版もできるだけ多くの機種の上で動くことが望ましい．手初めに互換機といわれるIBM機，HITAC Mシリーズ機への移し換えを試みた．

### 5.1 OS 360 から OS Ⅳ/F4 への移し換え

われわれが入手したSIL版SNOBOL4はOS 360用のものであったので実行回数計数機能を追加するに先立ってまず九大大型計算機センターのFACOM OS Ⅳ/F4上に移し換える必要があった．

SNOBOL4処理系が入出力に用いているFORTRAN実行時ルーチンの呼出し方法がOS 360とOS Ⅳ/F4とでは異なっているため，入出力に関係するSILマクロとIOルーチンとを変更せねばならない．

SILマクロ（図5(b)）では，FORTRANの実行

環境を整える INIT，出力の一部を行う OUTPUT とそれに関係する命令，リワインド等の補助入出力命令を実行する REWIND 等，計10個のマクロを変更した．

IO ルーチン(図5(c))では，入力を行う STREAD，大部分の出力を行う STPRNT，終了処理を行う FINIS の3ルーチンを変更した．

OS Ⅳ/F4 への移し換えは，SNOBOL4 処理系本体を詳しく理解することなしに，ドキュメントが整備されている SIL マクロと IO ルーチンの変更のみで済んだ．

しかし，FORTRAN 実行時ルーチンの呼出し方法が OS 360 と OS Ⅳ/F4 とで違うという事実に気づくこと，および OS Ⅳ/F4 の FORTRAN 実行時ルーチンの呼出し方法を知ることに大変苦労した．OS 360 と OS Ⅳ/F4 は互換性のあるそれぞれのハードウェア上で動作し，OS 自体も互換性があるとされているため，FORTRAN 実行時ルーチンも互換性があるものと単純に考えていたためである．しかし，互換機といっても OS も違い，FORTRAN コンパイラ，実行時ルーチンも違う．一般にユーザが直接扱うことのない実行時ルーチンの仕様が異なるのは当然ありうることであった．OS Ⅳ/F4 の FORTRAN 実行時ルーチンの使用法に関する情報は一部がマニュアル中に散見される程度で整理して提供されていない．そのため FORTRAN コンパイラの出力をダンプするなど試行錯誤したが，よくわからず，結局富士通の FORTRAN 処理系の担当者にたずねたところすぐに判明し，上に述べたようにわずかな修正で動作した．

### 5.2 VM/370 CMS と VOS 3 への移し換え

OS Ⅳ/F4 上で実行回数機能を追加した SNOBOL4 処理系を IBM VM/370 CMS と HITAC VOS 3 とに移し換えることは比較的簡単であった．アセンブリ言語が OS Ⅳ/F4 と CMS, VOS 3 とで同等であり，FORTRAN 実行時ルーチンは CMS と OS 360 がほぼ同じ，CMS と VOS 3 が同じであったからである．OS Ⅳ/F4 への移し換えの経験から FORTRAN 実行時ルーチンの呼出し方法に CMS と OS 360, OS Ⅳ/F4 とで違いがあることが予想されたので，初めからそのつもりでその調査にあたったためでもある．

異常終了時の処理は，オーバフロー，記憶保護例外等に対しては OS Ⅳ/F4 と CMS, VOS 3 とで同じ仕様のスーパバイザマクロを用いるので，それが起こった場合の実行回数計測結果の出力は可能である．CMS は会話型の単一ユーザ用 OS であるため，CPU 時間切れ等による異常終了は原則として起こらない．しかし，OS Ⅳ/F4 で実現されている端末割込み時の計測結果の出力は CMS と VOS 3 とでは実現できなかった．これは，OS Ⅳ/F4 FORTRAN の実行時ルーチンではその初期設定の際に端末割込み時の処理のためのルーチンを設定できるが，CMS と VOS 3 とではこのことが実現できるかどうか調査の方法がなかった．OS Ⅳ/F4 でもこのことはマニュアルには書いてなく，実行時ルーチンをダンプして初めてわかったものである．CMS や VOS 3 は手元の計算機ではないのでダンプのような方法はとりにくかった．

この移し換えでは実行回数計数機能を実現するために OS Ⅳ/F4 上で SNOBOL4 処理系に追加した部分はまったく変更する必要がなかった．

なお，VOS 3 への移し換えは1981年末から利用可能になったN1方式大学間ネットワークを利用して九大大型計算機センターから東大大型計算機センターへ行ったものである．ネットワークを利用したことによる移し換え上の問題点については文献7)を参照されたい．

### 5.3 移し換えにおける問題点

実行回数計数機能追加版 SNOBOL4 を実現した FACOM M200 とそれを移し換えた HITAC M200H/M280H, IBM 370/138 とは互換機であるといわれ，アーキテクチャは基本的に同じである．しかし，OS が異なり，FORTRAN コンパイラ，実行時ルーチンが異なるため，今回のような問題が生じた．

SNOBOL4 処理系が使用する OS 等の支援ルーチンは次の二つに分けられる．

(1) OS の機能を直接用いるもの　異常終了時処理等はスーパバイザマクロを用いて直接 OS に要求を出す．これらのマクロは3機種間でほとんど違いがなく，ほぼそのまま動作した．しかもスーパバイザマクロ等に関しては3機種ともマニュアルがしっかりしているのでそれらの比較も容易である．

(2) FORTRAN 実行時ルーチンを用いるもの　SNOBOL4 の入出力はすべて FORTRAN 実行時ルーチンを通して行われる．FORTRAN 実行時ルーチンが用いている OS のデータ管理マクロは3機種ともほとんど同じであるにもかかわらず，当該ルーチンには互換性がない．しかも，このルーチンは FORTRAN 利用者が明示的に用いるものではないの

で，それに関するドキュメントが（存在したとしても）ほとんど入手できない．そのため，SNOBOL4処理系の移し換えに当たってFORTRAN実行時ルーチンの仕様が問題となった．これがはっきりわかってしまえば，5.1節で述べたように移し換えに当たっての変更個所はわずかである．

## 6. むすび

SIL版SNOBOL4処理系に実行回数計数機能を追加し，さらに二つの計算機システム上に移し換えた．SILの設計は元来，多くの計算機システムの上でSNOBOL4処理系を実働させることが第一義であったものと思われる．しかしその設計のゆえに，機能拡張を容易に行うことができたことを強調しておく．この容易さを利用して，さらに日本語文字ファイルの扱いを可能とする拡張も行った[11]．これに関する詳細については別の機会に譲る．

なお，現在OS Ⅳ/F4の上で動く実行回数計数機能追加版SNOBOL4は九州大学大型計算機センターのほか，名古屋大学大型計算機センター，大阪大学社会経済研究所，慶応大学情報科学研究所でも稼動している．

この研究の一部は昭和57年度文部省科学研究費（一般研究B）の補助を受けた．

## 参考文献

1) Zerkowitz, M. V.: A Case Study in Rapid Prototyping, *Softw. Pract. Exper.*, Vol. 10, pp. 1037-1042 (1980).
2) Fleck, A. C.: Verifying Abstract Data Types with SNOBOL4, *Softw. Pract. Exper.*, Vol. 12, pp. 627-640 (1982).
3) 木村：SOFTWARE TOOLとは何か，情報処理，Vol. 20, No. 8, pp. 673-680 (1979).
4) Griswold, R. E.: *The Macro Implementation of SNOBOL4*, p. 310, W. E. Freeman & Co., San Francisco (1972).
5) 牛島，高比良：SNOBOL4輪郭作成機能の使用について，九大大型計算機センター広報，Vol. 12, No. 2, pp. 110-113 (1979).
6) 牛島，吉田，高比良：実行回数計数機能を追加したSNOBOL4処理系とその移植および使用について，第21回情報処理学会全国大会論文集，pp. 223-224 (1980).
7) 吉田，牛島：N1ネットワークを経由したソフトウェアの異機種間移し換え，ソフトウェア工学研究会資料 25-4 (1982).
8) 牛島，田町：プログラム移換作業から見たプログラミング環境の評価について，ソフトウェア工学研究会資料 17-12 (1981).
9) 牛島，藤村：実行回数計数ツール実現の総括，第21回プログラミングシンポジウム報告集 (1980).
10) Griswold, R. E.: A History of the SNOBOL Programming Languages, *ACM SIGPLAN Notices*, Vol. 13, No. 8, pp. 275-308 (1978).
11) 牛島，黒坂：日本語文字処理機能を追加したSNOBOL4処理系，第25回情報処理学会全国大会論文集，pp. 255-256 (1982).

（昭和58年2月24日受付）
（昭和58年4月19日採録）