ソフトウェア工学　28－20
（１９８３．２．１０）

実行回数計数機能を追加したＳＮＯＢＯＬ４処理系

吉田　和幸・牛島　和夫
（九州大学　工学部）

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［1，2］．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（ＯＳ３６０用，以下ＳＩＬ版ＳＮＯＢＯＬ４と呼ぶ）である［3］．これをＦＡＣＯＭ　ＯＳⅣ／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ならべて表示する機能を追加した［4］．さらにこの処理系をＩＢＭ　ＶＭ／３７０　ＣＭＳとＨＩＴＡＣ　ＶＯＳ３のもとに移し換えた［5，6］．一方，漢字等を含む日本語ファイルの扱いを可能とする機能拡張をも行った［7］．

本稿では，２章で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．３章で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．４章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．５章では実行回数追加版ＳＮＯＢＯＬ４の移し換えについて触れる．日本語ＳＮＯＢＯＬ４の詳細については別稿に譲る．

２．ＳＩＬ版ＳＮＯＢＯＬ４処理系

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［3］．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り当て，データの記憶，記憶域の再構成等を行う．システムインター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，番地部を持っている．例えば図１（ａ）の文は図１（ｂ）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わし，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わす．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持つ（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持つ．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）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２のような階層構成により実現されている［3］．最も外側（図２（ａ））がＳＩＬの世界で１３６個の命令を持つ．その内側（図２（ｂ））が１３６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２（ｃ））．これらのルーチンはアセンブリ語で記述され，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２（ｄ））．

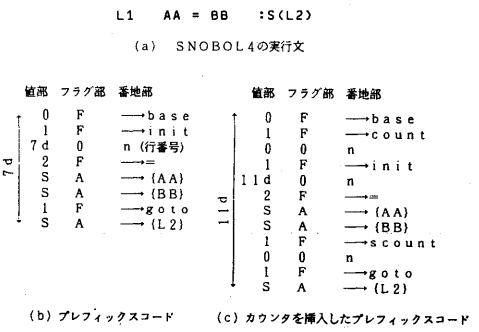


図１．ＳＮＯＢＯＬ４プログラムのプレフィックスコードへの変換例

3. 実行回数計数機能の実現

各文の実行回数を計測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［8］．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．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．ＳＮＯＢＯＬ４では各文の実行の成否によってプログラムの流れを変えることができる．従って，各文ごとに成功，失敗の数を計測することも重要である．そこで各文の実行回数とともにそれらも計測する．

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（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（図1．（c）参照，このためのトランスレータの書き替えはＳＩＬ言語で２２行）．

（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．

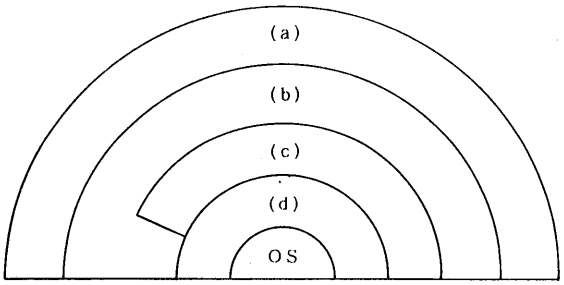
（3）カウンタ領域の確保　プレフィックスコードへの変換後，解釈実行に入る前にカウンタ領域を確保するようにした．このためＳＩＬ言語にカウンタ領域を確保する命令を追加し，これに対応して領域の確保をＯＳに実際に依頼するサブルーチンをＩＯルーチンに加えた．

（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２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ではオーバーフロー，記憶保護例外等のプログラム割り込みが起こった場合に，それまでの実行のサマリー（ＣＰＵ時間，入出力行数等）を出力できる．実行回数計数版ＳＮＯＢＯＬ４ではプログラム割り込み時に実行回数情報を出力するほか，ＣＰＵ時間切れ，ＴＳＳ使用時の端末割り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，異常終了後直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



（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2．ＳＩＬ処理系の階層構成

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．端末割り込み時の処理は，ＦＯＲＴＲＡＮの実行時ルーチンを利用している．

これらの追加修正をまとめる．

・(1), (2), (4), (5)はＳＩＬ言語（図２（ａ））で書かれた部分（全体で約６８００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２（ｂ））の追加である．この部分は６行である．

・(3)の後半と(6), (7)はＩＯルーチン（図２（ｃ））での修正である．ＩＯルーチンはモジュールの集まりであり，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２２６０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［3］．

（1）ＳＩＬシステムの構成　第２章にまとめたように構成が透明で機械または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．従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はディスクリプタ単位で数える．このためＳＩＬマクロ定義の段階でバイトマシン，ワードマシンの違い等を吸収できる．

（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によりＳＩＬ言語の修得が容易であった．そのためカウンタの挿入，結果の出力等の追加部分のほとんどをＳＩＬ言語で書けた．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によりはっきりわかっている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．さらにサブルーチン名，大域変数名等のＳＩＬで書かれたＳＮＯＢＯＬ４本体と連絡する名前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．

4．実行回数計数機能の評価

実行回数計数機能の使用例を図３に示す．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

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

                SOURCE  STATEMENT                                                    EXECUTION      SUCCESS      FAILURE
*       LIST THE LINE NUMBER OF EACH FORTRAN PROGRAM UNIT                   00000010
*                                                                           00000020
*       AUTHOR.        DR. N. FUJIMURA.                                     00000030
*       DATE-WRITTEN. 80/10/03.                                             00000040
*                                                                           00000050
        LINENO = 0                                                          00000060              1            1            0
        LINE   = '**MAIN**'                                                 00000070              1            1            0
        STARTLINE = 1                                                       00000080              1            1            0
        OUTPUT =                                                            00000090              1            1            0
        OUTPUT = '   FROM      TO  PROGRAM NAME'                            00000100              1            1            0
        OUTPUT =                                                            00000110              1            1            0
*                                                                           00000120
INPUT   BUF = INPUT                                :F(END)                  00000130           1640    (a)   1639            1
        LINENO = LINENO + 1                                                 00000140           1639          1639            0
        BUF LEN(72) . BUF                                                   00000150           1639          1639            0
        BUF (POS(0) 'C' ABORT) |                                            00000160           1639    (b)    987          652
+           (POS(5) (' ' | '0'))           :S(ST)F(INPUT)                   00000170
ST      BUF ' FORMAT'                      :S(INPUT)                        00000180            987           28          959
        BUF '    END    '                  :S(OUTPUT)                       00000190            959    (c)    30          929
        BUF (' SUBROUTINE' | 'FUNCTION' |                                   00000200            929    (d)    29          900
+           ' BLOCK ')                     :S(PROG)F(INPUT)                 00000210
*                                                                           00000220
PROG    LINE = BUF                                                          00000230             29           29            0
DLTBLK LINE FENCE ' ' =                    :S(DLTBLK)F(INPUT)               00000240            203          174           29
*                                                                           00000250
OUTPUT OUTPUT = ' ' DUPL(' ',6 - SIZE(STARTLINE)) STARTLINE                 00000260             30           30            0
+               ' ' DUPL(' ',6 - SIZE(LINENO))    LINENO                    00000270
+               '   ' LINE                                                  00000280
        STARTLINE = LINENO + 1                                              00000290             30           30            0
        LINE = '**MAIN**'                  :(INPUT)                         00000300             30           30            0
*                                                                           00000310
END                                                                         00000320              1            1            0
```

図３．実行回数計数機能使用例

位の名前とその区切りを表示するプログラムである［9］．図3（a）から入力となったＦＯＲＴＲＡＮプログラムの総行数が１６３９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．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一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１６３９回実行され，（d）も９２９回実行されている．そこでそれぞ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４に示す．図４のプログラムの実行は約２．８秒で済み，実行時間は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４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ところ，実行時間は約１．８秒となり，図４のプログラムに対して約６０％実行時間が短縮した．表１にこの３個のプログラムの実行時間を示す．表１では実行回数を計測した場合とともに計測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．実行回数を計測した場合の実行時間の増加は１割以内に収まっている．

図３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［9］があった時に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一文で書け，その組み合わ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．従ってテキストファイル処理

表１．各プログラム実行時間

プログラム	実行回数計数（P）〔ミリ秒〕	計数なし（NP）〔ミリ秒〕	比（P／NP）
図３	５５５２	５４３０	１．０２２
図４	２８６６	２７５３	１．０４１
図５	１７７１	１６６５	１．０７０

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［１０］．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）カウンタをプレフィックスコード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．

５．実行回数計数機能追加版の移し換え

ＳＩＬ版ＳＮＯＢＯＬ４は現在４０以上の機種で稼働している．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

```
        BUF FENCE 'C'                   :S(INPUT)           00000160    1639     586    1053
        BUF FENCE LEN(5) ' '            :S(ST)              00000170    1053     987      66
        BUF FENCE LEN(5) 'O'            :S(ST)F(INPUT)      00000180      66       0      66
ST      BUF ' FORMAT'                   :S(INPUT)           00000190     987      28     959
        BUF '    END    '               :S(OUTPUT)          00000200     959      30     929
        BUF ' SUBROUTINE'               :S(PROG)            00000210     929      24     905
        BUF 'FUNCTION'                  :S(PROG)            00000220     905       4     901
        BUF ' BLOCK '                   :S(PROG)F(INPUT)    00000230     901       1     900
```

図４．パターンマッチ部分の改良版（部分）

```
        &TRIM = 1                                           00000120       1       1       0
*                                                           00000130
        INPUT('SYSPIT',5,72)                                00000140       1       1       0
*                                                           00000150
INPUT   BUF = SYSPIT                    :F(END)             00000160    1640    1639       1
```

図５．ＴＲＩＭ機能使用による改良版（部分）

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．ＯＳ３６０からＯＳⅣ／Ｆ４への移し換え

我々が入手したＳＩＬ版ＳＮＯＢＯＬ４はＯＳ３６０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　ＯＳⅣ／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び出し方法がＯＳ３６０とＯＳⅣ／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．

ＯＳⅣ／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び出し方法がＯＳ３６０とＯＳⅣ／Ｆ４とで違うという事実に気付くこと，およびＯＳⅣ／Ｆ４のＦＯＲＴＲＡＮ実行時ルーチンの呼び出し方法を知ることに大変苦労した．ＯＳ３６０とＯＳⅣ／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．ＯＳⅣ／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等試行錯誤したが，よくわからず，結局富士通の当事者に訪ねたところすぐに解決し，上に述べた様にわずかな修正で動作した．

5．2．ＶＭ／３７０　ＣＭＳとＶＯＳ３への移し換え

ＯＳⅣ／Ｆ４上で実行回数計数機能を追加したＳＮＯＢＯＬ４処理系をＩＢＭ　ＶＭ／３７０　ＣＭＳとＨＩＴＡＣ　ＶＯＳ３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ＯＳⅣ／Ｆ４とＣＭＳ，ＶＯＳ３とで同等であり，ＦＯＲＴＲＡＮ実行時ルーチンはＣＭＳとＯＳ３６０がほぼ同じ，ＣＭＳとＶＯＳ３が同じであったからである．ＯＳⅣ／Ｆ４への移し換えの経験からＦＯＲＴＲＡＮ実行時ルーチンの呼び出し方法にＣＭＳとＯＳ３６０，ＯＳⅣ／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，オーバーフロー，記憶保護例外等に対してはＯＳⅣ／Ｆ４とＣＭＳ，ＶＯＳ３とで同じ仕様のスーパー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，ＯＳⅣ／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ＶＯＳ３とでは実現できなかった．これは，ＯＳⅣ／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ＶＯＳ３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．ＯＳⅣ／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ＶＯＳ３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ＯＳⅣ／Ｆ４上でＳＮＯＢＯＬ４処理系に追加した部分は全く変更する必要がなかった．

なお，ＶＯＳ３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．ネットワークを使用したための移し換え上の問題点については［６］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２つに分けられる．

（１）ＯＳの機能を直接用いるもの　異常終了処理等はスーパーバイ

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．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２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．しかしその設計の故に，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．

なお，現在ＯＳⅣ／Ｆ４の上で動く実行回数計数機能版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，大阪大学社会科学研究所，慶応大学情報科学研究所でも稼働している．

参考文献

[1] Zerkowitz, M.V.: A Case Study in Rapid Prototyping, Softw. Pract. exp., Vol.10, pp.1037-1042, 1980.

[2] Fleck, A.C.: Verifying Abstract Data Types with SNOBOL4, Softw. Pract. exp., Vol.12, pp.627-640, 1982.

[3] Griswold, R.E.: The Macro Implementation of SNOBOL4, W.E.Freeman & Co., p.310, 1972.

[4] 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．２，１９７９．

[5] 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．

[6] 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．

[7] 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．

[8] 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．

[9] 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．

[10] 木村：ＳＯＦＴＷＡＲＥ　ＴＯＯＬとは何か，情報処理，Ｖｏｌ．２０，Ｎｏ．８，ｐｐ．６７３－６８０，１９７９．