8. ストラクチャードプログラミング支援システムSPOTとその評価

紫合　治，下村建之，井元邦夫，岩元莞二，前島　亨
（日本電気　中央研究所）

1. はじめに

ソフトウェアの多様化，大規模化に伴って信頼性の高い保守し易いシステムをいかに効率よく作成するかということが重要な問題となってきた。ストラクチャードプログラミング[1]（以降SPと略す）はこの問題に対する1つの答を与えるものとして注目されているプログラミングの方法論で，

① トップダウン　プログラミング

② GOTOフリー　プログラミング

等の手法をまとめたものである。

これらの手法を実際に使っていく場合，それに適したツールが重要になってくる。例えば，GOTOを使わないプログラムを作るにはそれに適した制御文（CASE，REPEAT等）をもった言語が必要であり，トップダウンプログラミングには，それを推進するマクロ機能等が要求される。更にシステム開発を総合的に支援するツールとしては，この他プログラムの編集，管理，デバッグ用ツール等を含み，これらが有機的に統合された総合的な開発支援システムが望まれる。SPOTはこの様な観点からSP手法に基くソフトウェア作成支援システムの実験システムとして開発された。

SPOTシステムの主な特長は次の通りである。

(1) SPOT言語によって良い構造をしたプログラムの作成がやり易い。

(2) 使い易いマクロ機能によりトップダウンプログラミングがやり易い。

(3) プログラムの作成，修正，開発管理がTSS機能を使って容易にできる。

(4) 理解し易いデータ表現や制御文によって，プログラムリストが高いドキュメント性を有する。

一方，実際のソフトウェア開発にSP手法を取り入れる場合，開発のどの時点でどの様にSPを進めていくかとか，更にどの様なツールが望まれるか等の種々の問題がある。我々はSPOTシステムの開発を通してトップダウン，GOTOフリー等のSPの実験を行ったり，SPOT自身をツールとして使い，システムをSPOT言語ですべて記述してみて，その結果からSPOTがSP手法にどの程度役立つかとか，更に望まれる機能は何か等についての評価を行った。

以下に，SPOTシステムの機能とその開発過程に於いてなされたSPOT機能の評価について報告する。

2. SPOTシステム

ここではSPOTシステム全体の概要とSPOT言語の基本的な特色と思われる機能について述べる。

2.1 システム概要

(1) SPOTシステムの概要

SPOTシステムはSP指向型言語（SPOT言語）とそのプロッサ，マクロ展開処理システム，プログラ

ム保存管理システム，コマンドシステム及びその他のサービス機能からなる。図1にシステムの概要を示す。

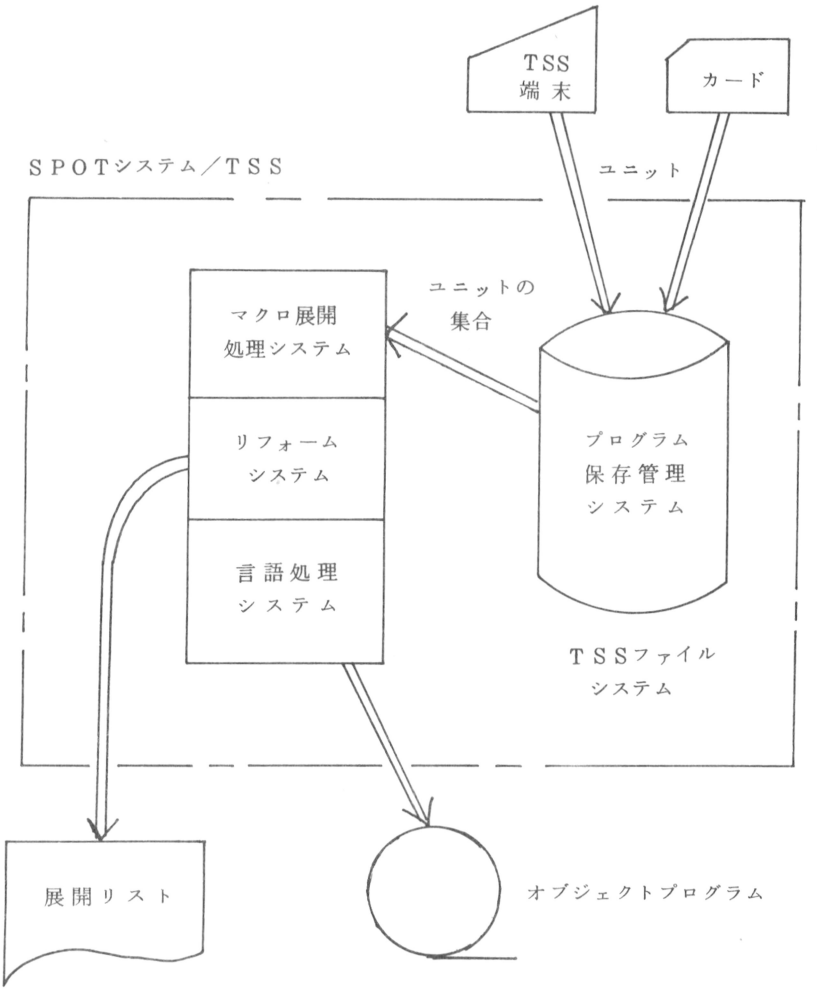


図1　SPOTシステム概要

SPOT言語で書かれたプログラムはユニット単位でプログラム保存管理システムに保存される。1つのユニットは手続き又はマクロを構成している。システムはこれらユニットの集合を入力してマクロ展開の後，言語プロセッサによってオブジェクト（BPL[2]の手続き）を作成する。この時マクロ展開されたソースリストを読み易い形になおして（段づけ等）出力する。

システムはNEAC－TSS上で使えるのでユニットの作成，修正，登録，リスト出力，マクロ展開変換等が端末から会話的にできる。サービス機能としてREFORMコマンドがある。これによって端末からフリーフォーマットで入力されたプログラム（ユニット）は見易い形に段づけされ，DO－END対応，構造体データの宣言等が見易くなり，プログラムリストのドキュメント性が増す。

(2)　SPOTで作成されたプログラムの構成

図2にSPOTで作成されたプログラム（SPOTプログラム）の構成例を示す。SPOTプログラムはモジュールの階層構造をなす。モジュールは1つの手続きユニットと，それから直接間接に呼ばれるいくつか

のマクロユニットからなる。モジュールはマクロ展開の範囲を規定するものであって，言語処理の単位となる。ユーザはユニット単位にプログラムの作成，修正を行う。手続きユニットおよびマクロユニット（SMACRO ユニット）は，そのユニットの仕様や入出力データ宣言（インタフェイスの仕様となる）等を含む DATA BLOCK と処理を記述する PROGRAM BLOCK からなる。ユニットの概要や他ユニットとの関係等は，そのユニットの DATA BLOCK を見るだけでわかる様になっている。

トップダウンプログラミングに於いてマクロはオーバヘッドのないモジュール化を可能にするものとして有用である。一方いわゆるテキストマクロはプログラムの構文と全く関係しないので展開されたところで元のプログラムの構造をこわす恐れがある。このため SPOT では，その呼び出しが手続き呼び出し同様文法的な＜文＞としての意味をもつマクロとして SMACRO がとり入れられた。SPOT では SMACRO がトップダウンプログラミングに於いて中心的な役割りを果たす。SMACRO の他に通常のテキストマクロも用意されている。

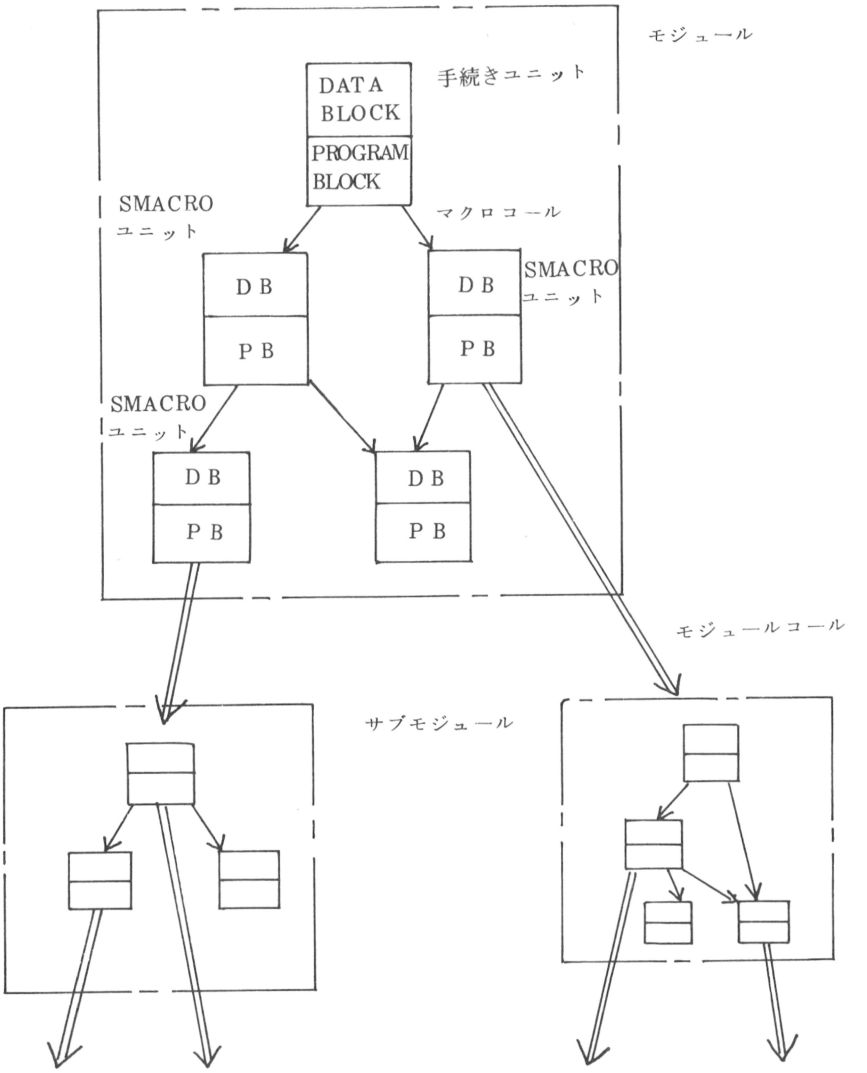


図２　ＳＰＯＴ プログラムの階層

(3) プログラム保存管理

SPOT プログラムはすべてユニットとして作成されシステムに管理される。各ユニットは論理的にまとまった機能を果し，それぞれの機能を示す名前（ユニット名）がつけられ，その名前が他のプログラム中のユニットコールで使われる。各ユニットは SPOT ファイルとして実現され，それぞれファイル名（ファイル管理上便利な名前で 10 字以内）がつけられる。ユニット名とファイル名の対応はユーザが定義しシステムに登録する。マクロ展開処理では，プログラム中のマクロユニット名から対応するファイル名を調べ，それによってマクロ本体の読み出しを行う。

開発過程で作成されるダミーユニットやデバッグ用のユニット等も一元的に管理するために，1つのユニット名に対して複数個のファイルを対応させることができる。各ファイルはA（available）かN（non-available）かの性質をもち，同一ユニット名に対応するファイルにはAが高々1つしかなく他はすべてNである。ユニット名からファイル名を得る場合，このAのファイル名（もしあれば）が使われる。これによって，目的に応じて必要なファイルを使うことができる（必要なファイルをAにすればよい）。

(4) SPOT コマンド

SPOT コマンドは NEAC－TSS の通常のコマンドと同じレベルで使われる。SPOT システムを使う場合，SPOT コマンドを入力した後で SPOT の各機能に対応するサブコマンドを入力する。サブコマンドとしては次の様なものがある。

1) DECLARE：ユニット名とファイル名を対応させて登録する。
2) AVAIL　：ファイルの availability の変更。
3) RENAME　：ユニット名，ファイル名の変更。
4) EDIT　　：ユニットの作成，修正。
5) PURGE　：ユニット，ファイルの消去
6) OUTPUT　：ユニットリストの出力
7) REFORM　：ユニットを見易い形に段づけして編集しなおす。
8) COMPILE：マクロ展開，チェック，BPL への変換および展開リスト出力。
9) UASK　　：指定されたファイル名から対応するユニット名を出力。
10) FASK　　：指定されたユニット名から対応するファイル名を出力。

2.2 SPOT 言語

SPOT 言語は SP を実行するのに適した各種機能をもつ高級言語として新たに設計された言語であり，SPOT トランスレータによって実験の便宜上 BPL（PL/I サブセットのシステム記述言語）に変換される様になっている。ここでは特に SPOT 言語の特長となる機能について述べる。プログラムの単位はユニットとなっており，ユニットは DATA BLOCK と PROGRAM BLOCK からなる。

(1) DATA BLOCK

DATA BLOCK はユニットの仕様（コメントで書く），シノニム宣言，タイプ宣言，各種変数の宣言およびサブユニットの宣言からなる。

i) シノニム

シノニムは最も単純なテキストマクロであるが他のマクロと異なりユニットを構成しない。ある名標をシノニムとしてその値（文字列）と共に宣言すれば，以後その名標が現れた時その値の文字列に置きかわ

る。

```
SYNONYM : MAXLENGTH = '132' ;
```

上の宣言によって，以後プログラム中では 132 のかわりに MAXLENGTH が使える。シノニムの文字列中に又シノニムが現れてもよい。

ii）タイプ

タイプは Pascal[3] の ＜Scalar type＞ とほぼ同じ機能をもつ。タイプ宣言は新しい属性名とその属性のとり得る値（名標）の列の組みからなる。

```
TYPE : ELEM( IDENTIFIER, CONSTANT, OPERATOR, ……) ;
```

上の宣言の結果，変数の属性として ELEM が使える。

```
OUTPUTVAR : 1 ELEMENT ,
            2 MARK   ELEM,
            2 BODY   CHAR (6) ;
```

プログラム中では従来マーカとしての意味しかもたなかった数値のかわりにタイプで宣言した ELEM 属性の値（IDENTIFIER, CONSTANT 等）が使える。

```
IF ELEMENT . MARK = OPERATOR THEN ……
```

iii）各種変数の宣言

変数の宣言はそのユニットのインタフェイスの記述ともなる。SPOT ではこの点を特に明確にするため，変数の宣言を次の 4 つの種類にわけた。

a）PARAMETER宣言

手続きパラメタの宣言を行う。パラメタで VARY 属性のついていないものはそのモジュール内で値の変更ができない。これによって不注意によるパラメタの値の変更が防げる。その他の宣言の仕方は PL／I の DCL 文とほぼ同様である。

```
PARAMETER : P1 BIN(6), P2 CHAR(10) VARY ;
```

b）INPUTVAR宣言

そのモジュールの入力変数（外部変数でそのモジュールの入口での値が保証されているもの）の宣言を行う。入力変数はそのモジュール内で値を変更することができない。

c）OUTPUTVAR 宣言

そのモジュールの出力変数（外部変数でそのモジュールの中で valid な値がセットされ，その出口での値が保証されているもの）の宣言を行う。

d）LOCALVAR 宣言

そのモジュールにローカルな変数の宣言を行う。

この様な区別により，変数の使われ方のシステマティックなチェックができる。例えば INPUT 変数は

① そのモジュールに入る前に値がセットされていること，

② そのモジュール内で値が変らないこと，

③ そのモジュールがコールするサブモジュールで INPUT 変数としてのみ使われていること，

の 3 点がチェックされれば正しい使い方がされているかどうかがある程度判定できる。

ⅳ) サブユニットの宣言

プログラム中で使われるモジュールやマクロはすべてその名前が宣言されねばならない。これによって関連ユニットのコール関係やシステムの階層構造がDATA BLOCKに明記され，わかりやすくなる。

(2) PROGRAM BLOCK

PROGRAM BLOCKは実行文の列からなり，BPLの文がほとんどそのまま使え，かつ各種SP指向制御文が使える。以下にSPOT特有の文についてのみ述べる。

ⅰ) SMACROコール

SMACROはプログラム中<文>が現われる場所でのみコールされる。SMACROのコールはSMACROユニット名(前後に°印をつけた任意の文字列)の出現による(手続きコールもAlgolの様に手続き名の出現による)。SMACROがコールされると，そのマクロユニットのDATA BLOCKが処理された後，PROGRAM BLOCKの本体がDO文とEND文で囲まれてコールされた位置に挿入される。マクロにはcall by name方式のマクロパラメタが使える。

```
°SCAN INPUT UNTIL('','')° ;
```

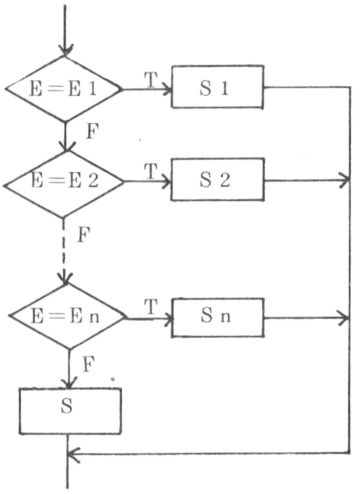
上の文は','を実パラメタとしたSMACROコールの例である。

ここで"SCAN INPUT UNTIL"がSMACROユニット名となる

ⅱ) CASEグループ

いくつかの文から条件に合った文を選択して実行する場合CASE文が使われる。

```
CASE E ;
 (E1):S1;
 (E2):S2;
 ..............
 (En):Sn;
 ELSECASE
      S ;
ENDCASE ;
```



ここでE，E1，E2，…，Enは式，S，S1，…，Snは文(又は文グループ)を示す。ELSECASE項はなくてもよい。E1，…，Enが論理式の場合，Eを省略するとtrueが書かれているのと同じ効果になる。E1，…，Enが数値定数又はTYPE値の場合，RANGE指定をつけることにより効率のよいオブジェクトができる(IF文でなく計算型GOTO文の様になる)。RANGE指定はCASE文のEに続いて，Eのとり得る値の範囲を指定する。

```
CASE E RANGE(I,J);
```

ここでI，Jはそれぞれ下限，上限値を示す定数(数値又はTYPE値)である。

ⅲ）REPEATグループ

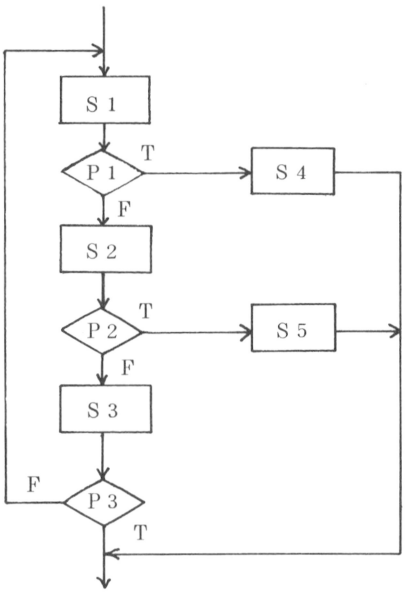
繰り返し指定条件により文のグループを繰り返し実行する。繰り返し指定条件としてはPL／Ｉの繰り返しDO文で指定できる項の他に，ＴＩＭＥＳが使える。

```
REPEAT E TIMES ;
```

```
REPEAT I＝1 TO E ;と同じ。但し，Ｉはシステム変数が使われる。
```

ループの途中からの脱出としてＥＸＩＴが使える。又，脱出後の後処理指定としてＥＸＩＴＣＡＳＥ文がある。

```
REPEAT ;
 S1 ;
 EXIT(L1)WHEN(P1);
 S2 ;
 EXIT(L2)WHEN(P2);
 S3 ;
UNTIL(P3);
EXITCASE
 (L1):S4 ;
 (L2):S5 ;
ENDEXITCASE ;
```



ここでＰ１，…，Ｐ３は論理式，Ｌ１，Ｌ２はＥＸＩＴＣＡＳＥレーベルと呼ばれる名標である。

ＲＥＰＥＡＴ文の前にＲＥＰＥＡＴグループ名を指定すれば，ＥＸＩＴ文でそのグループ名を書くことによってネストしたループの中から指定したＲＥＰＥＡＴグループを出ることができる。

この時ＥＸＩＴＣＡＳＥレーベルを併記することによって，指定したＲＥＰＥＡＴグループに対応する指定した後処理（次の例では，ＭＡＩＮＬＯＯＰに対応するＦＩＬＥＥＮＤの後処理）を行なうことができる。

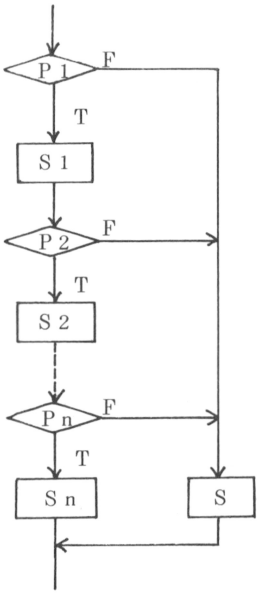
```
(MAINLOOP):REPEAT WHILE(NOERROR);
    ..................
    REPEAT ;
        ..................
        EXIT MAINLOOP(FILEEND)WHEN(EOP);
        ..................
    ENDREPEAT ;
    ..................
ENDREPEAT MAINLOOP;
EXITCASE
    ..............
    (FILEEND); ..........
ENDEXITCASE ;
```

ここでENDREPEAT文はUNTIL（false）と同じ意味をもつ。

iv）SEQUENCEグループ

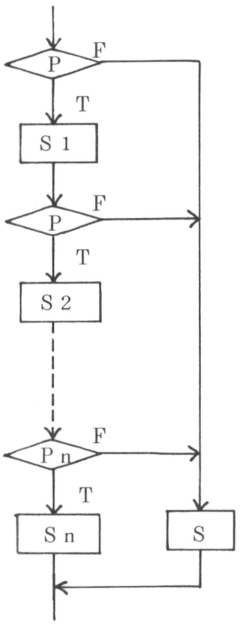
条件が成立する限り，グループに含まれる文を順次実行する。

```
SEQUENCE WHILE ;
  (P1) : S1 ;
  (P2) : S2 ;
     ..............
  (Pn) : Sn ;
ELSESEQ
        S  ;
ENDSEQUENCE ;
```



ここでP1，…，Pnが同じ論理式の場合はまとめてWHILEの後に書ける。

```
SEQUENCE WHILE(P) ;
    S1 ;
    S2 ;
   ..........
    Sn ;
  ELSESEQ
    S  ;
ENDSEQUENCE ;
```



```
TM¥EXPRESS..
 PROC OPTIONS(MAIN),.

DATABLOCK..
  /%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%/
  /% PROBLEM..                                                       %/
  /%  INPUT EXPRESSION, CHECK SYNTAX AND GENERATE OBJECT            %/
  /% CODE                                                            %/
  /%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%/
 SYNONYM..
  TRUE                                         #%%%1%%B%,
  FALSE                                        #%%%0%%B%..
 TYPE..
  ELEM                                         (
                                               IDENTIFIER,
                                               CONSTANT,
                                               OPERATOR,
                                               SEPARATOR,
                                               TEMPORARY¥VAR
                                               )  ,.
 OUTPUTVAR..
  1 ELEMENT                                    ,
    2 MARK                                     ELEM,
    2 BODY                                     CHAR(6),
  OPERATIONABLE                                BIT(1),.
 SMACRO..
  °PROLOGUE°,
  °INITIALIZATION°,
  °CHECK SYNTAX°,
  °TEST OPERATIONABILITY OF PRIOR OPERATOR°,
  °EPILOGUE°,.
 WMACRO..
  °ELEMENT IS END¥OF¥EXPRESSION OPERATOR°,
  °END OF INPUT FILE°,.
 SUBMODULE..
  GET¥ONE¥ELEMENT,
  PUSH¥OPERATOR,
  PUSH¥OPERAND,
  OPERATE¥PRIOR¥OPERATOR,.
ENDDATABLOCK,.

PROGRAMBLOCK..
  °PROLOGUE°,.
  REPEAT ,.
    °INITIALIZATION°,.
    REPEAT,.
      GET¥ONE¥ELEMENT,.
      °CHECK SYNTAX°,.
      CASE ELEMENT.MARK RANGE(IDENTIFIER,OPERATOR),.
        (IDENTIFIER,CONSTANT)..
         PUSH¥OPERAND,.
        (OPERATOR)..
         DO,.
          REPEAT,.
            °TEST OPERATIONABILITY OF PRIOR OPERATOR°,.
            EXIT WHEN(NOT OPERATIONABLE),.
            OPERATE¥PRIOR¥OPERATOR,.
          ENDREPEAT,.
          PUSH¥OPERATOR,.
         END,.
      ENDCASE,.
    UNTIL(°ELEMENT IS END¥OF¥EXPRESSION OPERATOR°),.
  UNTIL(°END OF INPUT FILE°),.
  °EPILOGUE°,.
ENDPROGRAMBLOCK,.

ENDPROC TM¥EXPRESS,.
EOP    1.264

READY 16-32-29
X
```

図3　手続きユニットの例

（端末文字の制限より　＝は#，'は%，：は‥，；は，．となっている）

```
  100        °TEST OPERATIONABILITY OF PRIOR OPERATOR°..
  200         SMACRO,.
  300
  400        DATABLOCK..
  500         TYPE..
  600          °OPERATOR TYPE O¥TYPE°,
  700          PRIORITY                                  (
  800                                                    LOWEST2,
  900                                                    LOWEST1,
 1000                                                    MIDDLE2,
 1100                                                    MIDDLE1,
 1200                                                    HIGHEST
 1300                                                    ),.
 1400         OUTPUTVAR..
 1500          OPERATIONABLE                             BIT(1),
 1600          PRIOR¥OPERATOR¥TYPE                       O¥TYPE,.
 1700         LOCALVAR..
 1800          CURRENT¥OP¥PRIORITY                       PRIORITY,
 1900          PRIOR¥OP¥PRIORITY                         PRIORITY,
 2000          PRIORITY¥TABLE(END¥OF¥EXPRESSION)         PRIORITY INIT(
 2100              /¥BEGIN¥OF¥EXPRESSION¥/ LOWEST2,
 2200              /¥PREFIX¥MINUS,--PLUS¥/ HIGHEST, HIGHEST,
 2300              /¥MINUS, PLUS         ¥/ MIDDLE2, MIDDLE2,
 2400              /¥MULTIPLE, DIVIDE    ¥/ MIDDLE1, MIDDLE1,
 2500              /¥EXPONENT            ¥/ HIGHEST,
 2600              /¥LEFT¥PAR,RIGHT¥PAR  ¥/ LOWEST1, LOWEST1,
 2700              /¥END¥OF¥EXPRESSION   ¥/ LOWEST2 ),.
 2800         SUBMODULE..
 2900          GET¥OPERATOR,
 3000          POP¥OPERATOR,.
 3100        ENDDATABLOCK,.
 3200
 3300        PROGRAMBLOCK..
 3400          CURRENT¥OP¥PRIORITY#PRIORITY¥TABLE(OPERATOR¥TYPE),.
 3500          GET¥OPERATOR,.
 3600          PRIOR¥OP¥PRIORITY#PRIORITY¥TABLE(PRIOR¥OPERATOR¥TYPE),.
 3700          IF OPERATOR¥TYPE#LEFT¥PAR THEN
 3800            OPERATIONABLE#FALSE,.
 3900          ELSE
 4000            CASE,.
 4100              (CURRENT¥OP¥PRIORITY LT PRIOR¥OP¥PRIORITY)..
 4200               OPERATIONABLE#TRUE,.
 4300              (CURRENT¥OP¥PRIORITY EQ PRIOR¥OP¥PRIORITY)..
 4400               CASE CURRENT¥OP¥PRIORITY,.
 4500                 (HIGHEST)..
 4600                  OPERATIONABLE#FALSE,.
 4700                 (LOWEST1,LOWEST2)..
 4800                  DO,.
 4900                   OPERATIONABLE#FALSE,.
 5000                   POP¥OPERATOR,.
 5100                  END,.
 5200                ELSECASE
 5300                  OPERATIONABLE#TRUE,.
 5400               ENDCASE,.
 5500              ELSECASE
 5600               OPERATIONABLE#FALSE,.
 5700            ENDCASE,.
 5800        ENDPROGRAMBLOCK,.
 5900
 6000        ENDSMACRO °TEST OPERATIONABILITY OF PRIOR OPERATOR°,.
EOP    1.363

READY 15-27-13
¥
```

図 4　SMACRO ユニットの例

3. ＳＰＯＴ機能の評価

ここではＳＰＯＴシステムの開発を通して得られたＳＰＯＴの各種機能の評価のうち，特にＳＰに関するものについて述べる。

3.1 コントロール文

(1) ループの脱出としてＥＸＩＴは効果的でよく使われた。特に無限ループとＥＸＩＴ文の組み合せは便利である。又多重ＥＸＩＴ文（ネストしたループからの脱出）は使用頻度は少ないが必要であった。

(2) ＥＸＩＴＣＡＳＥ文はＥＸＩＴ文と共にプログラムを複雑にする制御変数（flag 変数等）を減少させる効果があった。ＥＸＩＴＣＡＳＥ 項目は高々２項くらいであったがよく使われた。

(3) ＣＡＳＥ文はＴＹＰＥ の機能と相まって特に使いやすくなった。

```
CASE ELEMENT．MARK RANGE（IDENTIFIER，OPERATOR）；
（IDENTIFIER，CONSTANT）：PUSH¥OPERAND；
（OPERATOR）：……
  ……
ENDCASE；
```

ＥＬＳＥＣＡＳＥもよく使われた。ＣＡＳＥ文によって多重ＩＦ文等の複雑な使用は大巾に減少した。

(4) ＳＥＱＵＥＮＣＥ文はあまり使わなかった。条件をＷＨＩＬＥの後に書くＳＥＱＵＥＮＣＥは各ＳＥＱＵＥＮＣＥ項目（文グループ）の範囲がリストからは不明瞭で，条件判定のところがわかりにくいので一部構文を考えなおす必要があろう。

(5) ＳＰＯＴのコントロール文によって，ＧＯＴＯ文は次の２つの場合以外には使用しなかった。

① エラー処理時の非常脱出部，

② 従来のプログラムやＯＳとのインタフェイス部でどうしても必要なところ。

3.2 ＤＡＴＡ ＢＬＯＣＫの機能

(1) ＤＡＴＡＢＬＯＣＫの記述は，従来，プロジェクトによりプログラミング規則としてコメントで書かれていた部分が言語仕様に入ったので，リストのドキュメント性が増した。但し，コーディングでの煩しさが増すが，ＳＰＯＴ のソースリストは詳細仕様書ともなり，別に仕様書を作成する場合に比べればそれほど苦にはならなかった。

(2) データブロックの分離により，ユニットの仕様，インタフェイスの記述（宣言）等が集中され調べやすくなった。

(3) インタフェイス変数の宣言の仕方（ＩＮＰＵＴ，ＯＵＴＰＵＴ，ＰＡＲＡＭＥＴＥＲ等）は，その有効範囲等で現在のＳＰＯＴの仕様では種々不便な点があり充分とはいえないが，正しい値をセットする前にその値を参照する様なエラー（実際によくあったバグの１つ）のチェックに役立った。特にイニシャライズのやり忘れに対して有効なチェックとなった。

3.3 マ ク ロ 機 能

(1) トップダウンプログラミングを効果的に進めるには多重マクロの機能は必須であった。ＳＰＯＴ 作成では，文法的に文としての意味をもった ＳＭＡＣＲＯ がマクロ機能の中心として使われた。又，１ページプログラミング，機能の独立化（いわゆるモジュール化）を進めていく場合，ＳＭＡＣＲＯが効果的に多く使われた（１モジュールあたり平均６個強の ＳＭＡＣＲＯが使われた）。更に複雑なシンタックスマクロ等の必要は

ほとんど感じなかった。

(2) マクロパラメタは SMACRO では便利であったが，使用頻度は低かった。又，あまり自由な使い方はせず，＜式＞と規制してもよかった。

(3) マクロ名は充分長く，又スペースを許したのでプログラムが見易くなった。多くのマクロで 30 字以上の名前が使われた。前後の ° 印は，かえって見易くする効果があった。

(4) 単なるテキストマクロでは，簡単で手軽なシノニムが便利であった。その他，宣言をまとめる等の目的でテキストマクロユニットが使われたが，プログラム構造にそった使い方しかしなかった。

3.4 SPOT プログラムのドキュメント性

(1) 各種制御文により GOTO 文のないプログラムが書けリストを上から順に読むだけで理解できる様になった。又 CASE 文では同レベルの条件が並ぶような場合は少々プログラムが長くても理解し易い。

(2) シノニムやタイプの使用により論理レベルが高くなり理解し易くなった。

(3) マクロ名が充分長いのでそれから処理内容の大略が理解でき，下位プログラムを見ずに 1 ユニットのプログラムが理解できる。従って他と独立にその意味がわかるユニットを 1 ページくらいの大きさで書ける様になった。

(4) データブロックはプログラムの管理に有効と思われる。プログラムの変更がどこまで影響を及ぼすかが比較的容易に把握できる。

(5) リフォームされたプログラムリストは大変見易い。適当に空行を入れておけば，GOTO 文のないプログラムではフローチャートと同じ位に見易い（慣れればフローチャートより見易い）。SPOT レベルの言語およびマクロ機能とプログラムリストのリフォーム機能があれば，詳細フローチャートは必要なくなる。SP 用言語ではフローチャート自動作成よりリフォーム機能が望まれる。

3.5 その他の SPOT システムの評価

(1) SPOT 作成過程では SPOT のプログラム保存管理機能は使えなかったが，開発中ダミーユニットをデバッグ用に作成するため同一ユニット名をもつファイルを複数個管理する必要があり，ファイルの availability 機能は充分有効であると思われる。

(2) SPOT では TSS 端末からプログラムの修正ができるので，特に REFORM コマンドは必須であった。又展開リストがリフォームされるのも必須である。

(3) デバッグについては，従来よりモジュール化がうまくなされ，プログラム自身のドキュメント性が高いので机上デバッグが有効であった。

4. おわりに

SP 支援システム SPOT とその評価について述べた。SPOT 開発を通して，SP 手法が実際的なシステムの開発にもかなり有効であることが確められた。

一方 SP 方式の留意点，問題点として，次の様な事項が上げられる。

(1) トップダウンプログラミングに於いて上位のプログラム中に，論理的に下位の表現を書いてしまうことが多い。これによりプログラムは理解しにくくなる。

(2) 上位プログラムでのデータの扱い方がむずかしく，従来のトップダウン，GOTO フリーからなる SP だけでは不充分であった。又，データの refinement 方式が問題になった。

(3) トップダウンプログラミングに於いて，プログラムの分割の仕方に個人差がかなり出た。又フローチャートレスでプログラムを作成するには，慣れが必要であった。

このうち(1)と(2)については，抽象化の概念[5]が非常に有効であった。又，(3)より誰でもSPでやれば良いプログラムが作成できるという訳にはいかず，充分な教育と訓練が必要であるといえよう。

今回定量的に充分評価できなかった問題として次の様なものがあげられる。

(1) SPによるシステムの実行およびメモリ効率はどうなるか。

(2) SPOTで作成されたプログラムはどの程度保守し易くなるか。

これらについては今後のSPOTの使用，保守を通して評価していくつもりである。

従来のプログラミングシステムは，人間と計算機のコミュニケーションのみを重視し，人間が計算機に指令する手段としてのみ開発されてきたが，これからの大規模システムの開発を考えると，更に人間と人間のコミュニケーションを考慮した。ドキュメント性の高いシステムが必要となってくるであろう。

今後は，更に信頼性の高いシステム開発のために，SPを含む広範なソフトウェア開発手法と，それに適した各種機能を含む総合的な支援システムの研究開発が望まれよう。

最後に，本研究を進めるに当って，便宜と御指導を頂いた当社，中研・村上研究部長，藤野研究部長代理に深謝します。

文　献

〔1〕 Dahl，Dijkstra，Hoare “Structured Programming” 1972， Academic Press

〔2〕 小久保，佐谷“コンパイラ記述言語BPL” 情報処理 Vol．11 №6 1970

〔3〕 Wirth “The Programming Language Pascal” Acta Informatica 1， 1971

〔4〕 紫合他 “SPOTシステム開発に於けるストラクチャードプログラミングの評価”
情報処理学会 第15回大会 1974

〔5〕 下村他“抽象化手法によるソフトウェア開発” 情報処理学会 第15回大会 1974

〔6〕 Shimomura， Shigo， Maejima
“SPOT：A Structured Programming Development System”
ACM 74 Anual Conf． 1974