ソフトウェア工学 26－2
（1982. 10. 26）

ライブラリモジュールを用いたプログラムの半自動的詳細化

藤田米春　斎直人　西田富士夫
（大阪府立大学　工学部）

1．まえがき

近年，要求仕様技術やドキュメンテーションの技術が活発に研究されている．それらの中には，ＰＳＬのような制限された自然言語で書かれた仕様書を，処理システムがいろいろな観点から解析し，半自動的に，プログラム化可能な仕様にまとめて行くものなどがある．(3)

他方，プログラムの自動合成は古くから研究されて来ているが，正確に仕様定義された非常に簡単なプログラムの実行部分しか合成できず，データ構造の決定などやアルゴリズムの選択などの実際的な問題については，まだ解決されていないものが多い．(2)

さて，現在までのところ，具体的なプログラミング言語レベルでのサブルーチンの共用を除くと，異なった種類の仕事は異なったプログラムで処理されるのが普通である．したがって従来莫大な数のプログラムが個別的に作成されて来た．しかしながら，通常作成される大部分のプログラムは，基本的に共通の手順や関数から作られていることが多い．したがって，その様な共通に用いられる手順や関数を，具体的に展開するときに加工しながら展開可能な形にして，ライブラリとして蓄え，これを用いてトップダウン的にプログラムに詳細化して行くことにすれば，能率的かつ柔軟に仕様をプログラムに展開できると考えられる．

このような考え方に従って，この報告では，プログラム仕様からライブラリモジュールを用いて半自動的に仕様を詳細化し，プログラムに変換する方法を述べる．プログラムの仕様は，手順的表現，入出力表現またはパスカル風の表現を用いて記述する．これらの表現は，格ラベルを用いて係受け関係を明確にしており，これにより人間にも機械にも取扱かい易いものとなっている．

現在拡張中の実験システム（SAPRE：Semi-Automatic Program Refinement and Expansion System）は，ライブラリモジュールを自ら探索するか，またはユーザの指示により，ライブラリモジュールを選び，これを仕様や詳細化途中のモジュールに適用して，具体的なプログラムに変換する．各ライブラリモジュールには，その手順をやや詳細に示すOP部がある．OP部は，詳細化に関してユーザの選択を促す選択文を含んでいる．詳細化の途中経過や結果は，一種のHIPO図のような形でも表示することができ，ユーザと機械の間の対話をしやすくしている．

ライブラリモジュールをプログラム仕様に適用するときに，2階の述語論理における単一化(1)を，特殊化したものを用いる．

2．プログラムの仕様

プログラムの仕様記述はいろいろな表現を用いて行うことができるが，仕様の内容により，それらの表現の適・不適が変って来る．たとえば，入出力関係が計算式のように数学的に明確に定義された関数を計算する場合や，方程式などのように入力と出力の関係が陽に与えられる場合には，入出力関係による仕様記述が適当であるが，ファイルの更新，合併や客へのオンラインでの応答を行う事務用の

システムなどの場合には，手順的な表現の方が表現が容易であり，適切な場合が多い．このような観点から，この報告では，仕様の表現として，入出力表現，手順的表現および種々の制御構造を手順的表現で表わしたパスカル風表現を適宜用いることにする．なお，これらの表現は種々の形式をとれるが，ここではその基本的なものについて示すことにする．

2．1　手順的表現

手順的表現の形式は，

処理名（格ラベル：対象，---，格ラベル：対象）　(1)

の形をしており，どのような対象をどのように処理するかを大まかに記述している．これは，PASCAL などの procedure や function と類似の形をしているが，(1)式の「格ラベル：対象」には procedure や function では引数として現れない中間的な変数や条件なども含む．格ラベルとして標準的なものを次に示す．

OBJect	：手順の対象要素．
SOurce	：手順の対象要素が属するデータベースの名前．
GOal	：手順の適用結果を格納すべき場所の名前．
KEY	：対象を処理するために用いられるキーパラメータ．
USE	：対象を処理するために用いられるデータベースの名前．
CONDition	：対象を処理するための条件．
LOC	：対象が存在する位置．
PART	：二項演算子の第2引数など．

[例1] 与えられた配列 r(1..m) に対して EQ(t(i,1..m), r(1..m)) を満たす部分配列 t(i,1..m) を配列 t(1..n, 1..m) から見つけ，これを z(1..m) に格納する手続の手順的表現は，次のようになる．

search(SO：t(1..n,1..m), OBJ：t(i,1..m), GO：z(1..m),
　　COND：EQ(t(i,1..m), r(1..m)))．

この手順的表現は，人間の理解にも機械の処理にも比較的容易な妥協的な仕様の一つの形式的表現として導入したものである．同じ内容を表わす異なる記述がいくつか存在する場合には，変換規則により，標準的な表現に変換する．

2．2　入出力表現

入出力表現は次のような入出力条件の対である．

(IN：P(x), OUT：Q(x,z))　(2)

ここに P(x) はリテラルの連言，Q(x,z) は連言標準形か

$$\bigwedge_i \left(\left(\bigvee_j P_{ij} \right) \rightarrow Q_i \right)$$

の形をとるものとし，手順的表現と同様に格名を引数に前置するものとする．

[例2] (IN：Exist(OBJ：mark(1..5000, 1..8), LOC：MARK(1..5000, 1..8)),
　　OUT：Exist(OBJ：mark(1..5000, 1..9), LOC：MARK(1..5000, 1..9))
　　∧ ∀i∈(1..5000)(MARK(i,9) = sum(MARK(i,2..8))))

ここに，Exist(OBJ：x, LOC：X) は対象 x が場所 X に在ることを表わす．

2．3　パスカル風表現

PASCAL のようなプログラミング言語がもつ制御構造は，仕様の記述においても有用な場合が多い．これは，手順的表現や入出力表現よりも小回りがきき，直接的にプログラムを書く方が仕様記述が能率的で容易な場合があることによる．そこで，仕様記述の弾力性を増大させる観点から，反復や条件分岐を表す仕様記述のための表現として，次に示す(3)，(4)，(5)式などの表現を用いる．これらは，手順的表現と同様の形式を用い，表現形態の統一性を図るとともに仕様の機械による解読を容易にし，他のコンパイラ言語などのターゲット言語によるプログラム展開を容易にする．

do(FOR：i:=l, TO：n, OP：spec)　(3)
do(WHILE：P, OP：spec)　(4)
if(IF：P, THEN：spec1, ELSE：spec2)　(5)

ただし，"spec" は手順的表現，入出力表現，パスカル風表現のどれかである．(3)，(4)，(5)式は，それぞれ，for文，while文，if文を表す．

上記の表現からPASCAL以外のコンパイラ言語へ変換するには，内部表現の変換表およびその言語の生成規則を用いて行うことができる．

仕様の系列が実行可能であるためには，手順的表現のGOal格以外の各引数や入出力表現の入力条件が，仕様の入力データや先行する手順的表現のGOal格や入出力表現の出力条件によってあらかじめ決定されていなければならず，システムはこのような必要条件が満たされているかどうかをチェックする．

[例3]「ソースプログラムを表す文字列が配列TEXT(1..1000)に格納されている．これを左から走査して区切記号の表DELIM(1..10)に格納されている記号が現れたらその位置をzに入れる」．この仕様は次のようにsearchモジュールを用いて書ける．ただし走査の始点をi_0とする．

```
Exist(OBJ：source-program(1..1000), LOC：TEXT(1..1000));
do(FOR：i:=i0, TO：1000,
    OP：search(SO：DELIM(1..10), OBJ：DELIM(j), GO：y,
                COND：DELIM(j)=TEXT(i) );
        if(IF：y≠φ, THEN：z:=i; go(OBJ：lab)) )
lab:
```

3．ライブラリモジュールと単一化

3．1　ライブラリモジュール

プログラム仕様を半自動的に詳細化するために，一般的によく用いられるプログラムモジュールを，変形加工可能な形にしてこれを階層的な仕様記述によって表したモジュールのライブラリを作成しておく．これらのモジュールは，統計処理，テーブル処理，入出力操作などに分類されてライブラリに格納される．

各モジュールの仕様記述は，2．で述べた三種類の表現と，型，OP部などから成り，入出力表現か手順的表現を用いて検索可能である．この入出力表現と手順的表現を「見出し部」と呼ぶことにする．

型は，モジュールで用いられる変数が単純変数か複合変数か，また，複合変数ならば配列かレコードかなどを表わす．

OP 部は，見出し部で表わされたモジュールの機能の詳細化されたものを表わし，この部分は，入出力表現，手順的表現，パスカル風表現によって，より下位のモジュールを組合わせて書かれている．また，詳細化に関してシステムがユーザにいくつかの展開方法の一つを選択させる場合には，OP 部に，

$?select(t,(t_1,\cdots,t_n))$

の形の表現を設け，t の値として $t_1,\cdots,t_n$ のどれを選ぶかを問合わせるようにする．表１に，テーブル操作のモジュールの一部を示す．

(1) PROC	add(OBJ:x(m1..m2), GO:z(n1..n2,m1..m2))
IN	Exist (OBJ:{x(m1..m2), z(n1..n2,m1..m2)}, LOC:{x(m1..m2), z(n1..n2,m1..m2)})
OUT	EQ(x(m1..m2), z(cp(z), m1..m2))
TYPE	x(m1..m2), z(n1..n2,m1..m2):array of integer or character
OP	copy(SO:x(m1..m2), OBJ:x(m1..m2), GO:z(cp(z)+1, m1..m2)); inc(cp(z))
(2) PROC	search(SO:x(n1..n2,m1..m2), OBJ:x(*,m1..m2), GO:z(m1..m2), COND:P(x(*,m1..m2), r(m1..m2)))
IN	Exist (OBJ:{x(n1..n2,m1..m2), r(m1..m2)}, LOC:{x(n1..n2,m1..m2), r(m1..m2)})
OUT	Exist (OBJ:z(m1..m2), LOC:z(m1..m2)) ∧P(x(*,m1..m2), r(m1..m2))→EQ(x(*,m1..m2), z(m1..m2))
TYPE	x(n1..n2,m1..m2), z(m1..m2), r(m1..m2):array of integer or character
OP	?select(q_1, (sequential: do(FOR:JJ:=n1, TO:n2 OP: if(IF:P(JJ, m1..m2), r(m1..m2)), THEN:copy(SO:x(JJ, m1..m2), OBJ:x(JJ, m1..m2), GO:z(m1..m2)); go(OBJ:lab))); lab:) (binary:?select(q_2, (i:GT≙GT at i-th column, *:GT≙lexicographic)) yy1:=n1; yy2:=n2; JJ:=(yy1+yy2)/2; do(WHILE:not(JJ=yy1 ∨ JJ=yy2), OP: if(IF:P(x(JJ, m1..m2), r(m1..m2)), THEN:copy(SO:x(JJ, m1..m2), OBJ:x(JJ, m1..m2), GO:z(m1..m2)); go(OBJ:lab), ELSE:if(IF:GT(x(JJ, m1..m2), r(m1..m2)), THEN:yy2:=JJ, ELSE:yy1:=JJ)); JJ:=(yy1+yy2)/2); lab:))

表１．テーブル操作モジュールの一部

表1において，addは，表に新しいレコードを追加する操作であり，copyは配列を他の配列に写す操作である．また，searchは2次元配列の中から一つの行を探し出す操作である．このモジュールのOP部ではユーザにsequential(逐次)探索かbinary(二分)探索かの選択を要求する選択文(?select)が設けられており，展開に際してユーザの指示を求める．

3.2 単一化

各ライブラリモジュールは，プログラム仕様の広い範囲にわたって適用できるように，一般的な形で表現され，多くの場合，関数変項や述語変項を含む．したがって，このようなライブラリモジュールによりプログラム仕様から詳細化するためには2階の論理における単一化が必要である．本報告では，プログラム仕様を定項からなるものとみて，単一化をライブラリモジュールからプログラム仕様またはその子孫の方向に限定して，単一化の木が閉じるようにするとともに，単一化手順を簡単化し，能率化している．2階述語論理における単一化は，単なる代入と異なり，代入可能性の判定やλ-表現を含む式の簡約化などの処理が必要である．また，一般に，単一化の木は閉じないので，上記のような特殊化は2階の述語論理の単一化を実用化するためには，本質的な効果がある．

いま，2個の対象α,βを単一化するものとする．αは$a_1\cdots a_k\cdots a_n$なる記号列で，次のような一つのライブラリモジュールの表現に含まれる式とする．

(PROC: proc(x,z), IN: P(x), OUT: Q(x,z), OP: op(x,z)) (6)

同様にβは$b_1\cdots b_k\cdots b_m$なる記号列で，プログラムの仕様かその子孫に含まれているものとする．

a_k, b_kをα，βの中の最左端の不一致記号とするとき，簡単化された単一化手順は次のように述べられる．

(1) a_kが個体変項のとき，次の代入により1階の述語論理の単一化を行なう．

$$\theta=\{a_k\leftarrow b_k\cdots b_{k+l}\} \quad (7)$$

(ただし$0\leqq l\leqq m-k$で，$b_k\cdots b_{k+l}$はβの部分対象である．)

(2) a_kがαの部分対象の先頭のp位の関数または述語変項のとき．

(a)射影

a_kを先頭とするαの部分対象と，a_kの第i引数のタイプが等しいとき代入

$$\theta=\{a_k\leftarrow\lambda u_1\cdots u_p\cdot u_i\} \quad (1\leqq i\leqq p) \quad (8)$$

により第i引数に射影する．

(b)模倣

b_kをq位の関数または述語定項とする．このとき，b_kを先頭とするβの部分対象を，代入

$$\theta=\{a_k\leftarrow\lambda u_1\cdots u_p\cdot b_k h_1\cdots h_q\cdot\} \quad (9)$$

(ただし，$h_i=f_i u_1\cdots u_p$で$1\leqq i\leqq q$ かつf_iはαにもβにも現れないp位の関数または述語変項とする．)

により模倣する．

上記の手順を有限回適用すれば，必ず単一化は終了することが証明されるが，ここでは略す．

[例4] (a) α: $f(x,y)$ と β: $-(+(x,y),/(x,y))$ は，$f\leftarrow\lambda uv\cdot -(+(u,v),/(u,v))\cdot$ により単一化される．

(b) 2個の論理式

α : $P(x(*, l_1..l_2), r(l_1..l_2))$ と

β : $\bigwedge_{j=j_1}^{j_2} EQ(room(*, j), req(j))$

とは，次の代入を逐次行なうことにより単一化される．

$P \leftarrow \lambda u_1(v_1..v_2)\, u_2(v_1..v_2) \cdot \bigwedge_{j=v_1}^{v_2} EQ(u_1(j), u_2(j)) \cdot$,

$l_1 \leftarrow j_1$, $l_2 \leftarrow j_2$,

$x \leftarrow \lambda u_1 u_2 \cdot room(u_1, u_2) \cdot$,

$r \leftarrow \lambda u \cdot req(u) \cdot$

ここに $u_1(v_1..v_2)\, u_2(v_1..v_2)$ は $u_1(v_1)..u_1(v_2)\, u_2(v_1)..u_2(v_2)$ の簡略表現であり，$\bigwedge_{j=j_1}^{j_2} Q(j)$ は SAPRE では Λ(FOR : $j := j_1$, TO : j_2, OBJ : $Q(j)$) のように表わす．

4. 詳細化と変形

プログラムの仕様は，ライブラリモジュールを用いて対話方式で変形され，詳細化される．SAPRE は，与えられたプログラム仕様またはその一部に適用可能なモジュールを演算の種類，手順的表現，入出力表現，部分対象の型などをキーとして探索する．

4. 1 詳細化

与えられた仕様とライブラリモジュールの単一化を試み，単一化可能なモジュールが見つかった場合には，このモジュールの OP 部に単一化に伴なう代入をほどこした結果を詳細化結果とする．もし OP 部に選択文がある場合には，ユーザに選択を促し，その指示に従った詳細化を行なう．

[例5] SAPRE に，仕様 *search* (SO : TABLE(1.. n, 1.. m), OBJ : TABLE(i, 1.. m), GO : z(1.. m), COND : TABLE(i, 1) = r(1)) が与えられた場合，表1の *search* モジュールに対して，代入 $x \leftarrow$ TABLE, $n1 \leftarrow 1$, $n2 \leftarrow n$, $m1 \leftarrow 1$, $m2 \leftarrow m$, $P \leftarrow \lambda u v \cdot (u \dot{\leq} v) \cdot$ を行うとともに，SAPRE は選択文により，"sequential" か "binary" を選択することをユーザに要求し，"binary" が選択された場合，述語 "GT" の選択をすることをユーザに要求する．ただし，"$\dot{\leq}$" は第 i 列の要素で比較することを表わすものとする．この要求に対して "1" を選択した場合，

```
yy1:= 1; yy2:= n; JJ := (yy1 + yy2)/2;
do(WHILE: not(JJ = yy1 ∨ JJ = yy2),
   OP: if (IF: TABLE(JJ, 1) = r(1),
           THEN: copy (SO: TABLE(1..n, 1..m), OBJ: TABLE(JJ, 1..m),
                       GO: z(1..m) ); go(OBJ: lab),
           ELSE: if (IF: TABLE(JJ,1) > r(1),
                     THEN: yy2:= JJ, ELSE: yy1 := JJ) );
       JJ:= (yy1+ yy2)/2
   lab: )
```

と詳細化される．

4. 2 分割

プログラムの仕様の一部分と単一化可能な部分を含むモジュールがあるときに

は，仕様を部分的に変形したり，分割したりして，そのモジュールが適用可能な形に書換えて，仕様の詳細化が試みられる．このための変換規則として，次の2種類がある．

(a) $\dfrac{\begin{array}{l}(\mathrm{IN}: P_1(x), \mathrm{OUT}: Q_1(x,z), \mathrm{OP}: pr_1(\mathrm{OBJ}: x, \mathrm{GO}: z)) \\ (\mathrm{IN}: P_2(x'), \mathrm{OUT}: Q_2(x',z'), \mathrm{OP}: pr_2(\mathrm{OBJ}: x', \mathrm{GO}: z')) \\ P_1(x) \wedge Q_1(x,z) \rightarrow P_2(x')\end{array}}{\begin{array}{l}(\mathrm{IN}: P_1(x), \mathrm{OUT}: Q_2(x,z'), \mathrm{OP}: pr_1(\mathrm{OBJ}: x, \mathrm{GO}: z), \\ \qquad\qquad pr_2(\mathrm{OBJ}: \{x,z\}, \mathrm{GO}: z'))\end{array}}$

(b) $\dfrac{\begin{array}{l}(\mathrm{IN}: P(x), \mathrm{OUT}: Q_2(x,z_1), \mathrm{OP}: pr_1(\mathrm{OBJ}: x, \mathrm{GO}: z_1)) \\ (\mathrm{IN}: P(x), \mathrm{OUT}: Q_2(x,z_2), \mathrm{OP}: pr_2(\mathrm{OBJ}: x, \mathrm{GO}: z_2)) \\ z_1 \neq z_2\end{array}}{\begin{array}{l}(\mathrm{IN}: P(x), \mathrm{OUT}: Q_1(x,z_1) \wedge Q_2(x,z_2), \mathrm{OP}: \{pr_1(\mathrm{OBJ}: x, \mathrm{GO}: z_1), \\ \qquad\qquad pr_2(\mathrm{OBJ}: x, \mathrm{GO}: z_2) \})\end{array}}$

上の図式は，横線の上の前提条件から，横線の下の結論が導かれることを示す．与えられた仕様またはその子孫の出力表現が，ライブラリモジュールの出力表現を含んでいることがわかると，分割規則の横線の下の結論を仕様に適用して，規則の前提で表される部分問題に，仕様を分割する．

[例6] 仕様として

(IN: Exist(OBJ: v(1..n), LOC: x(1..n)),
OUT: z = sum{(x(i) - average(x(1..n)))**2 | i∈(1..n)}/n)

が与えられ，もし平均値の計算がfunctionとして与えられていないときには，規則(a)により，次のように分割される．

(IN: Exist(OBJ: v(1..n), LOC: x(1..n)), OUT: u1 := average(x(1..n)),
OP: average'(OBJ: x(1..n), GO: u1))
(IN: Exist(OBJ: (v(1..n), average(v(1..n))), LOC: (x(1..n), u1)),
OUT: z = sum{(x(i) - u1)**2 | i∈(1..n)}/n,
OP: z := sum{(x(i) - u1)**2 | i∈(1..n)}/n).

5．プログラムへの展開と例

プログラムの仕様は，プログラムへ展開可能ないくつかのモジュールに詳細化され，大域的に最適化された後に，ユーザにより指定されたプログラミング言語に展開される．展開されたプログラムは，宣言部分と実行部分とからなる．宣言部分は，詳細化の過程で指定された型や各選択文の応答結果を用いて作成される．実行部分は，プログラミング言語の内部表現の変換規則などを用いて生成される．

次に，コンパイラの字句解析部分の仕様から字句解析プログラムを生成する例について述べる．

[例7]「text(1..end)に与えられたソースプログラムを左から読み，トークンを切出し，切出されたトークンをリテラル，予約語，アイデンティファイヤに分類して各々の表に格納するとともに，中間表現を作成する．」

この仕様を格表現を用いて書きなおせば，

```
ps := 1;                                          ps : point of scanning
do(WHILE: not(text(ps) = EOF)
   OP: find(SO: text(ps..end),
            OBJ: text(left..right),
            COND: First-token(text(left..right)),
            GO: token(1..token-end));
       classify(OBJ: token(1..token-end),
                CLASS: {literal, reserved, identifier},
                GO: class);
       union(OBJ: (token(1..token-end), class), PART: class-table,
             GO: class-table);
       add(OBJ: (code(token(1..token-end)), table-pointer),
           GO: intermediate-table))
```

となる．システムは上記の find，classify，union，add あるいはこれらと同義のモジュールをライブラリから探し，単一化可能かどうかを調べる．単一化可能であれば対応するモジュールの OP 部に単一化の代入を行なった結果によって詳細化を行ない，さらにこの OP 部を同様にして詳細化する．単一化可能なモジュールがなければ，上の各手順について格ラベルや対象の型などがほぼ一致するモジュールをユーザに提示し，これか仕様に条件の付加や変形などを行なって単一化可能な形にした後，上と同様に詳細化を行なう．変形などによってもライブラリモジュールと単一化可能にすることができない場合には，仕様の各手順的表現を，ライブラリモジュールをベースとしてユーザが詳細化する．本例の場合，union，add は，PART 格，GO 格の対象のデータの形を決定することにより，ライブラリモジュールと単一化可能となり，find については，

```
find(SO: text(ps..end), OBJ: l, COND: Start-of-token(text(l)),
     GO: left)
find(SO: text(left..end), OBJ: r, COND: End-of-token(text(r)),
     GO: right)
copy(OBJ: text(left..right), GO: token(1..token-end))
```

と分解することにより，copy はライブラリモジュールと単一化可能となり，find は search を小し変形すればこれと単一化可能となる．したがってトークンを見つける手順はライブラリモジュールにより表現できる．同様の手順により classify もライブラリモジュールにより表現できる．

このようにして，すべての手順がライブラリモジュールにより表現された後，具体的なプログラミング言語に変換する．

6．むすび

現在のシステムは，プログラム仕様の分割規則の適用が，状態の評価やプランなしに行われるので，むだが多い．また，モジュールの変形などの処理をもっと自動化する必要がある．現在これらについて検討中である．

文 献

(1) T. Pietrzykowski : A complete mechanization of second-order type theory., JACM, Vol.20, No.2, pp.333-365 (april 1973).

(2) J.L. Darlington : Automatic Synthesis of SNOBOL programs, in Computer oriented learning processes, J. C. Simon (ed.), pp.443-453, Nordhoff-Leyden (1976).

(3) D. Teichrow and E.A. Hershey : PSL/PSA : a computer-aided technique for structured documentation and analysis of information processing systems., IEEE Trans, SE-3, 1, pp.41-48 (1977).

(4) D.R. Barstow : Knowledge-based program construction., book, Elsevier North Holland, Inc. (1979).

(5) J. Foisseau et al. : Program development with or without coding., IFIP, pp.327-330 (1980)

(6) Y. Fujita and F. Nishida : A Program Synthesis by Hierarchical Definition System., IECE of Japan, Trans, J61-D, No.2, pp.103-110 (1978).

(7) 黒木, 西田, 藤田 : プログラムの半自動的詳細化., 情報処理学会第23回全国大会. P.399 (1981).

Appendix

例7の字句解析プログラムを, ライブラリモジュールを用いて展開した結果の一部を次に示す. プログラムの実行部分は約110行であり, この生成に要した時間はACOS-700 TSS LISP で約60秒(GC時間を除く)であった.

```
BEGIN
 JJ6:=1 ;
 WHILE NOT (TEXT[JJ6] = NIL)
  DO BEGIN
      FOR I2:=JJ6 TO 10000
       DO BEGIN
           FOR JJ13:=1 TO 50
            DO IF ==(TERMINALT[JJ13,1..(I2 - I2) + 1],TEXT[I2])
                THEN BEGIN
                       JJ02:=JJ13 ;
                       GOTO 100
                      END ;
          JJ02:=0 ;
     100 : IF JJ02 = 0
            THEN Y3:=0
            ELSE Y3:=TERMINALT[JJ02,JJ22] ;
           IF Y3 = 0
            THEN MONITOR(ERROR) ;
           IF NOT (Y3 = SPACE)
            THEN GOTO 110
          END ;
 110 : L1:=I2 ;
       FOR JJ17:=1 TO 50
        DO IF ==(TERMINALT[JJ17,1..(L1 - L1) + 1],TEXT[L1])
           THEN BEGIN
                  JJ03:=JJ17 ;
                  GOTO 120
                 END ;
       JJ03:=0 ;
 120 : IF JJ03 = 0
        THEN Y4:=0
        ELSE Y4:=TERMINALT[JJ03,JJ23] ;
       IF Y4 = BREAK
        THEN BEGIN
               I2:=L1 ;
              GOTO 130
             END ;
       FOR I2:=L1 + 1 TO 10000
        DO BEGIN
            FOR JJ21:=1 TO 50
             DO IF ==(TERMINALT[JJ21,1..(I2 - I2) + 1],TEXT[I2])
                THEN BEGIN
```