B 5　トランスレータ記述システムの開発

西野秀毅，小島富彦，霜田忠孝，梅野英典，吉村一馬　（日立製作所中央研究所）

1.　はじめに

コンパイラの生産性をあげるために，コンパイラ・コンパイラのように言語の構文と意味と計算機の仕様を入力して，コンパイラを自動的に作ろうとする方法から，記述し易い言語を開発し，それを用いて，作成しようという方法まで，色々の試みがなされて来た。

私達は人と計算機で協力してコンパイラを作る，トランスレータ記述システムの方法を選んだ。この方法が，すべてを自動的に作成する方法にくらべて，無理な点がすくなく，語彙解析，構文解析，意味処理のすべてにわたり，能率のよいものを作りやすいと判断したからである。

このシステムの主要部分は語彙解析プログラム作成の部分と，構文解析プログラム作成部分とからなっている。

語彙解析のプログラムはコンパイラの構文解析の部分に比べて，演算時間がかかるので処理能率をよくし，コンパクトにまとめることが要請される。そこで，語彙解析プログラムの機能の分析を行ない，語彙解析記述言語の形にまとめた。この記述言語は６５種類の命令からなっている。

構文解析プログラムは，Backus 記法（ＢＮＦ）で書かれた文法を入力すると，構文解析表を自動的に生成するプログラムシステムを使って作成する。このシステムは語彙解析や意味解析の情報を利用したり，機械語の生成に必要でない情報を省くことで，アルゴリズムの簡約化をはかっている。手法はトップダウン型の解析が容易なＬＬ(k)文法[1]に基づくものと，ボトムアップ型の解析を行ない，広い言語族が扱かえるＬＲ(k) 文法[5]に基づくものを選んだ。

2.　システムの概説

我々のトランスレータ記述システムは語彙解析記述言語ＬＸ１とＬＬ(k)文法の構文解析プログラム作成を対象とするＬＡＳＡＬシステムと，ＬＲ(k)文法の構文解析プログラム作成を対象とするＬＲ(k)アナライザⅡよりなる。

ＬＡＳＡＬシステムはＢＮＦで書かれた文法を入力するとフロイドプロダクション型の構文解析プログラムを作成し，次いでその機械語を作り出す。

一方，ＬＲ(k)アナライザⅡはＢＮＦで書かれた文法を入力すると，その構文解析手続きを決定性プッシュダウンオートマトンの形に構成し，それを「表」にした構文解析表を出力するものである。

この２つのシステムでは，いづれも語彙解析，意味処理の情報がスムーズに処理出来るような工夫がされている。すなわち，ＬＡＳＡＬシステムではＦｌｏｙｄ　Ｐｒｏｄｕｃｔｉｏｎ　言語のレベルで，ＬＲ(k)アナライザⅡではＢＮＦのレベルでアクション・ルーチン名が書けるようになっており，意味処理との橋渡しがなり自然に行えるようになっている。

3.　ＬＸ１言語

3.1　概説

我々は語彙解析プログラムの記述のために，ＬＸ１言語を発案した。この言語によって，語彙解析プログラムはマクロなレベルで記述でき，その結果，(1)プログラムの記述速度は従来の２～３倍になり，(2)ドキュメントはプログラムだけでよくなった。しかも，処理速度の向上とエリアの縮小をも達成することができた（このことに関する実験結果は第４章で示す）。

3.2　ＬＸ１の主な機能

ＬＸ１言語はアセンブラ言語と同じ形式で記述され，宣言文と実行文とからなる。以下，代表的機能だけを取り上げて説明する。

3.2.1　宣言文

(1)　ＫＩＮＤ文

これは，文字の文字クラスを定義するために使われる。次の例では英字の文字クラスを４と定義している。

ＫＩＮＤ　　４，Ａ，Ｂ，Ｃ，……，Ｘ，Ｙ，Ｚ

(2)　ＣＨＯＰＴＡ文（ＣＨＯＰ　ＴＡＢＬＥ）

これは，ＣＨＯＰ文によって連続的に読み込まれる文字セット情報を与えるテーブルを定義する。次の例は英数字を連続読みしたいときＩＤＮＴという名前を使って記述できるようにする為の宣言文である。

ＣＨＯＰＴＡ　ＩＤＮＴ，Ａ，Ｂ，Ｃ，……，Ｘ，Ｙ，Ｚ，１，２，……，９，０

(3)　ＳＫＩＰＴＡ文（ＳＫＩＰ　ＴＡＢＬＥ）

これは連続的に読みすてる文字セット用のテーブルを定義するものである。形式はＣＨＯＰＴＡと同じ。

3.2.2　実行文

(1)　ＧＥＴＲ文（ＧＥＴ　ＲＥＣＯＲＤ）

ソースプログラムの次の一行を読み込む。

(2)　ＲＥＡＤＣ文（ＲＥＡＤ　ＣＨＡＲＡＣＴＥＲ）

ソースプログラムの次の一文字を読み込む。

この文の実行の結果，読み込まれた文字の文字クラスと文字コードが得られる。

(3)　ＳＴＡＴ文（ＳＴＡＴＩＳＩＺＥ）

今読み込んだ文字の文字クラスの値に従って，ジャンプ先きを変える。次の例は文字クラスが４，５，６，７の時，それぞれＡＬＰＨＡＢＥＴ，ＤＥＬＩＭＩ，ＮＵＭＢＥＲ，ＢＬＡＮＫにジャンプする。

ＳＴＡＴ　ＡＬＰＨＡＢＥＴ，ＤＥＬＩＭＩ，ＮＵＭＢＥＲ，ＢＬＡＮＫ

(4)　ＳＴＲＤ文（ＳＴＯＲＥ　ＡＮＤ　ＲＥＡＤ）

読み込み文字をトークン形成エリアにストアし，さらにその後，ＲＥＡＤＣ文に等しい処理をする。

(5)　ＳＷＴ文（ＳＥＡＲＣＨ　ＡＮＤ　ＷＲＩＴＥ　ＴＯＫＥＮ）

トークン形成エリア内の文字列を持って名前表を検索し，登録されていないときは登録し，登録番地を得る。

(6)　ＬＩＳＴ文（ＬＩＳＴ　ＴＯＫＥＮ）

ＳＷＴ命令で求まった番地とトークンの種別をトークンリストエリアにリストする。（構文解析プログラムはここからＳＣＡＮする。）

(7)　Ｂ文（ＢＲＡＮＣＨ）

ブランチ命令

(8)　ＣＨＯＰ文（ＣＨＯＰ）

指定されたテーブル（ＣＨＯＰＴＡ参照）に従って，レコードエリアの文字を連続読みして，これらの文字列をトークン形成エリアにストアする。なお連続読みしているときレコードエリアの終りの記号が現われた時，次のレコードを読み込み，レコードエリアポインタのリセット等が必要になってくる。このような作業を我々は割り込みルーチンで処理することにした。

3.2.3　文字クラスの役割

ＬＸ１言語で扱われる文字は，文字コードと文字クラスの２通りの扱われ方をする。文字コードとはその文字自身の内部コードである。文字クラスとは通常４から３１までの数で表わし，文字の用法を区別する。

語彙解析の処理速度はソースプログラムの文字数にほぼ比例している。それ故，語彙解析の処理能率は一文字当りの処理能率に大きく左右される。文字クラスはこの文字の処理能率を上げる上で重要な役割をはたしている。すなわち，文字クラスによって，

(1)　特殊な文字は一回の判定で区別できるようになり，

(2)　一行の区切りとか，ソースプログラムの終り等の実際の文字がないものでも，文字として統一的に扱えるようになり，

(3)　文字セット内に文字が属しているかどうかも一回の判定で区分けできるようになった。

図3.1にＬＸ１コーディング例を示す。これは四則演算を行うプログラムに対する語彙解析プログラムである。

```
                KIND      4，A，B，C，D，……X，Y，Z
                KIND      5，PLUS，MINUS，STAR，SLNT，EQUA
                          SCOL，RP，LP
                KIND      6，0，1，2，3，4，5，6，7，8，9
                KIND      7，BLNK
                CHOPTA    IDNT，A，B，C，……X，Y，Z，1，2，3
                          4，5，6，7，8，9，0
                CHOPTA    DIGIT，0，1，2，3，4，5，6，7，8，9
                SKIPTA    SPACE，BLNK
                GETR
                READC
BRANCH          STAT      ALPHABET，DELIMI，NUMBER，BLANK
DELIM           STRD
TBL             SWT
                LIST
                B         BRANCH
ALPHABET        CHOP      IDENT
                B         TBL
NUMBER          CHOP      DIGIT
                B         TBL
BLANK           SKIP      SPACE
                B         BRANCH
```

図 3.1　　ＬＸ１コーディング例

4. LASALシステム

4.1 概要

ＬＡＳＡＬシステムは語彙解析プログラム，構文解析プログラムを半自動的に作成するシステムである。（図4.1参照）

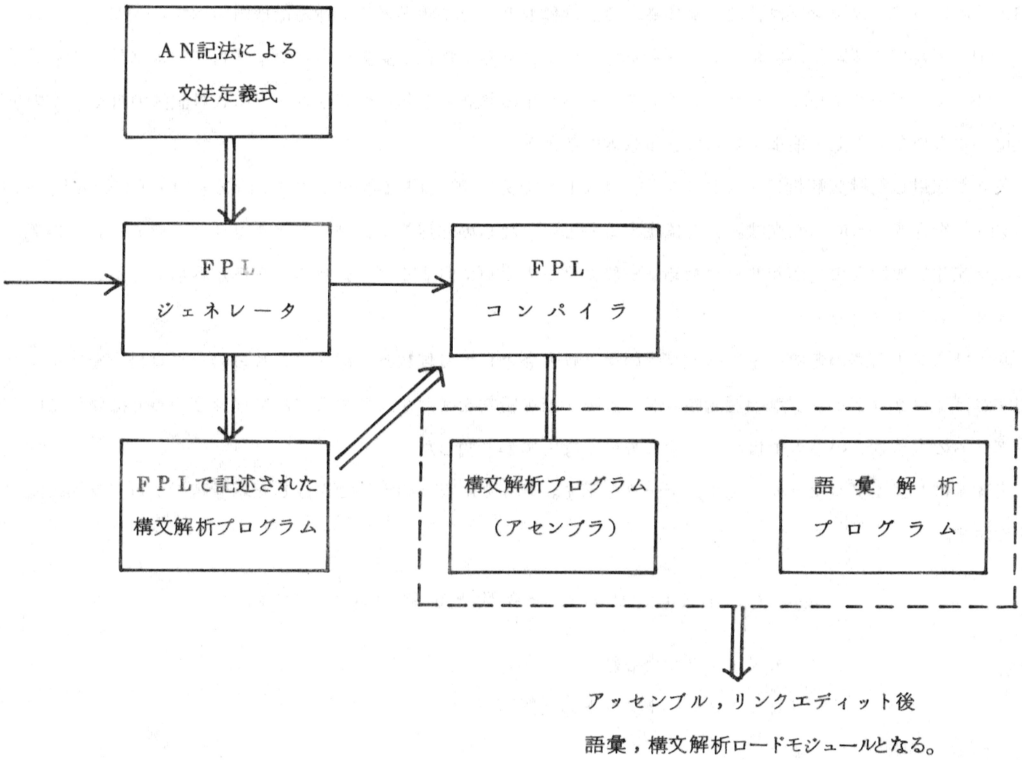


図 4.1　ＬＡＳＡＬシステム

ＬＡＳＡＬシステムにおいては語彙解析プログラムと，構文解析プログラムとでは，その作成法が本質的に異っている。語彙解析プログラムの作成には語彙解析アルゴリズムを語彙解析記述言語（ＬＸ１）で記述し，それをＬＸ１プロセッサで処理する方式をとる。一方，構文解析プログラムは文法の定義（言語の構文の定義）を記述した後は，ＥＰＬジェネレータ，ＥＰＬコンパイラにより自動的に構文解析プログラムが生成される。語彙解析プログラムの作成に対して問題向き記述言語方式を選んだのは次の理由による。

(1)　語彙解析アルゴリズムにおいては処理速度が重要である。

(2)　語彙の定義から自動的に語彙解析プログラムを作成する方式においてはまだ能率のよい方法が示されていない。（ＭＩＴのＡＥＤ　ＲＷＯＲＤシステムでは手作りの約2.7倍の時間がかかると報告されている。）

(3)　語彙解析アルゴリズムの定式化を行ってみると，約１０個程度の基本機能で多くの部分をカバーしており，これらに対してできる限りの最適化を施すことにより効率向上が期待できる。

(4)　上で述べた基本機能を中心として言語にまとめ上げ，これで記述してみると容易に書け，かつ読みやすさにお

いてもすぐれている事がわかった。

この問題向き記述言語方式に対して，構文解析プログラム作成の方は問題記述方式と考えられる。即ちユーザは自分の対象としている言語の文法をＡＮ記法（ＡＬＧＯＬ　Ｎの文法定義に使われた記法）で記述する。

ＡＮ記法で表わされた構文生成規則は，ＦＰＬジェネレータにより，構文解析アルゴリズム記述言語であるＦＰＬプログラム[3]（Ｆｌｏｙｄ Ｐｒｏｄｕｃｔｉｏｎ Ｌａｎｇｕａｇｅ）に変換される。さらにこのＦＰＬプログラムはＦＰＬコンパイラによってアッセンブラ言語に変換される。次にＦＰＬを中間語として選んだ理由を示す。

(1) アルゴリズム記述におけるフレキシビリティに富み，修正，変更が容易で，プログラムは読みやすい。

(2) 上の理由により，ユーザとのインタフェース用の言語となる。つまりユーザにＡＮ記法で書くことの出来ない，又は困難な事をＦＰＬで追加，修正できる余地を与える。

我々が採用した構文解析アルゴリズム[3]はＡＬＬ(k)文法[2]（ＬＬ(k)文法にｄｉｒｅｃｔ ｌｅｆｔ ｒｅｃｕｒｓｉｖｅ ｒｕｌｅ を許すようにした文法。）を認識するあと戻りなしの回帰トップダウンアルゴリズムである。このアルゴリズムは構文生成規則との対応が明瞭なため，一般に文法の変更に対するプログラムの修正が容易である。

4.2　ＬＸ１プロセッサ

我々はＬＸ１言語の各機能をアセンブラのマクロ命令として表現した。従ってこれらのマクロ命令をアセンブラで表現すれば，マクロアセンブラの展開機能がプロセッサの役割をはたす。この方式は処理速度の点では優れているが，プログラムの大きさという点ではかならずしも良くない事が判明した。

表4.1にＬＸ１プロセッサに関するデータを示す。表4.2に我々の手法で作成した語彙解析プログラムに関するデータを示す。

表 4.1　ＬＸ１マクロの命令数と定義のステップ数

ＬＸ１マクロ命令数 （ＬＸ１ステートメントの種類）	６５
ＬＸ１マクロ定義のステップ数	８５０ステップ

表 4.2　得られた結果

	FORTRAN ＪＩＳ ７０００レベル	FORTRAN ＪＩＳ ７０００レベル
プログラムの大きさ		
(1) ＬＸ１ステップ数	５６６	２２９
(2) マクロ展開の結果	１７３５８バイト	１２６２４バイト
語彙解析速度	(イ) 1.2msec	
	(ロ) ≈1.3msec	

注 (イ)の例題はＡＡＡ＝ＢＢＢ＋ＣＣＣ＋ＤＤＤに対するもの。同一データに対するFORTRAN Gコンパイラの全コンパイル時間は１２msec である。(370-155)

(ロ)ほとんどすべてのタイプのステートメントを含んだデータに対する一行当りの語彙解析時間

```
1890        SG      ACT(L018  52  L017R);;
1900
1910    L017R   ::  /*   F RCRSV PART   */
1920        **      ACT(* L018  53  L017R);;
1930        SG      ACT(RETURN);;
1940
1950    L018    ::  /*   P   */
1960        I       ACT(* 54  RETURN);;
1970        (       ACT(* L015  L01802)<ERROR  36>;;
1980
1990    L01802  ::  /*   <P>::=( <AE> )   */
2000        )       ACT(* 55  RETURN)<ERROR  37>;;
```

図 4.3　四則演算に対するFPLプログラム例

入力文法の制限事項として，AN記法で記述される文法はALL(k)文法でなければならないというものがある。ここではまずLL(k)文法の定義を与え，それをもとにして，ALL(k)文法を定義する。

G＝（N，Σ，P，S）は次の条件が成り立つ時LL(k)文法という。

『ある整数kに対して，次の2つの最左導出がある時は$\beta=\tau$となる。

(1)　$S \overset{*}{\underset{lm}{\Rightarrow}} wA\alpha \underset{lm}{\Rightarrow} w\beta\alpha \overset{*}{\underset{lm}{\Rightarrow}} wx$

(2)　$S \overset{*}{\underset{lm}{\Rightarrow}} wA\alpha \underset{lm}{\Rightarrow} w\tau\alpha \overset{*}{\underset{lm}{\Rightarrow}} wy$

ただしk：x＝k：y，およびw，x，yは終端記号列』　なおk：xはxの先頭の記号列で，長さがkである。

この定義に従うとdirect left recursiveルールを含んだ文法はLL(k)にならない。たとえばE::＝E＋Tのルールを含んだ文法はLL(k)にならない。ALL(k)文法はdirect left recursiveルールで定義されるすべての非終端記号に対して，

(1)　$A ::= A\alpha_1 \mid A\alpha_2 \mid \cdots\cdots \mid A\alpha_m$　とした時（m個以上Aのdirect left recursionルールはないとする。），ここでAを含んだ任意の2つの導出を考える。

i)　$S \overset{*}{\underset{P}{\Rightarrow}} \beta A \tau$

ただしこの導出においてAのdirect left recursiveルールは適用していないとする。

ii)　$S \overset{*}{\underset{P}{\Rightarrow}} \beta' A \tau' \Rightarrow \beta' A\alpha_i \tau'$

この時

$FIRST_k(\tau) \frown FIRST_k(\alpha_i \tau') = \phi$

ここで$FIRST_k$の定義を示す。

$FIRST_k(\tau) \equiv \{ x \mid \tau \overset{*}{\Rightarrow} x\delta,\ \mid x \mid = k, x \in \Sigma^* \}$

4.3 FPLジェネレータ[4)]

FPLジェネレータへの入力はAN記法を基にして，これに少し追加，修正をしたものである。図4.2に入力の例を示す。

```
BEGIN
   <AE> :: = ≪ +|− ≫ <T> (( +|− )) . . . ;;
   <T>  :: = <F> (( *|〃/〃 )) . . .        ;;
   <F>  :: = <P> (( ** )) . . .            ;;
   <P>  :: = I|〃 ( 〃 <AE> 〃 ) 〃          ;;
END
```

注　≪ および ≫は〔と〕に相当する。
((および)) は{と}に相当する。
〃で囲むことによりANの予約語でなくなる。

図 4.2　四則演算をAN記法で記述した例

FPLジェネレータの出力は既に述べたごとく入力文法の構文解析アルゴリズムをFPLで表現したものである。(図4.3参照)。

```
1700   L015     ::  /* AE */
1710      +     ACT (* L016 44 L015R) ;;
1720      −     ACT (* L016 45 L015R) ;;
1730      SG    ACT (L016 46 L015R) ;;
1740
1750   L015R    ::  /*  AE RCRSV PART  */
1760      +     ACT (* L016 47 L015R) ;;
1770      −     ACT (* L016 48 L015R) ;;
1780      SG    ACT (RETURN) ;;
1790
1700   L016     ::  /*  T  */
1810      SG    ACT (L017 49 L016R) ;;
1820
1830   L016R    ::  /*  T RCRSV PART  */
1840      *     ACT (* L017 50 L016R) ;;
1850      /     ACT (* L017 51 L016R) ;;
1860      SG    ACT (RETURN) ;;
1870
1880   L017     ::  /*  F  */
```

⑵ すべてのｉ，ｊ（$1 \leq i, j \leq m$）に対して，

$FIRST_k(\alpha_i) \cap FIRST_k(\alpha_j) = \phi$　　ただし $i \neq j$

次にＦＰＬジェネレータの処理手続を示す。

⑴ ＡＮ記法で書かれた文法生成規則を単純な生成規則（Ｐｒｏｄｕｃｔｉｏｎ ｒｕｌｅ）に変換する。

⑵ 単純な生成規則から各非終端記号のすべての特徴連系の集合を求める。

注　（この集合は２種類に分類される。これを仮に第１種特徴連系集合，第２種特徴連系集合と名ずけ，$F^1(A, i)$，$F^2(A, j)$と書くことにする。$w \in F^1(A, i)$とは次の事を意味する。$A :: = \alpha_1 \mid \alpha_2 \mid \cdots \mid \alpha_n$，かつ$|w| = k$，$A \neq 1 : \alpha_j$（$1 \leq j \leq n$）の時$w \in FIRST_k(\alpha_i)$および$w \notin FIRST_k(\alpha_j)$（$i \neq j$）が成り立つ。つまり先頭の記号列を見て$\alpha_i$と他を区別するもっとも短い終端記号列である。
$F^2(A, i)$はＡの左回帰ルールがある時定義される。
$A :: = \alpha_1 \mid \alpha_2 \mid \cdots\cdots \mid \alpha_n \mid A\alpha_1' \mid A\alpha_2' \mid \cdots\cdots \mid A\alpha_m'$とした時，$w \in F^2(A, i)$とは$\{\alpha_1', \alpha_2' \cdots\cdots \alpha_m'\}$に関する第１種特徴連系である。）

⑶ 特徴連系集合よりＦＰＬプログラムを生成する。

ＥＰＬプログラムには４種類の文の集合がある。これらを⑵で示した記述を使って述べる。

㈠ Ａの定義式α_iのどのｉを選択すべきかを決める。

㈡ ｉが決定された後はα_iの構成要素の確認を行う。もし必要があればサブゴールを設定する。

㈢ Ａに左回帰規則がある時，α_iの確認後今のルーチンを呼び出していた所に戻るか，Ａの次のどのα_j'を選択するかを決める。

㈣ ㈡と同じくα_j'の構成要素の確認を行う。

⑷ ＦＰＬの最適化

ｉ．$F^1(A, 1)$，$F^1(A, 2)$ ……，$F^1(A, n)$において要素の数が一番多いものを$F^1(A, j)$とする。ここで$1 : \alpha_j$が非終端記号であればコンパイル時に$a \in F^1(A, i)$（ただし$i \neq j$）と解った時$a \in F^1(A, j)$と仮定して構文解析を進める文を生成する。

ⅱ．上の⑶で示された㈠と㈡は重複する所がある。これをなくすために出来る限り㈠で㈡の仕事を行う。

図4.3はＳＭＡＬＬ　ＡＬＧＯＬの構文規則に対するＦＰＬジェネレータの出力の一部分である。この部分は四則演算を処理する所である。１７００行目のＬ０１５はラベルであり：：はラベル区切り記号である。／＊と＊／はＰＬ／Ｉと同じくコメントを囲む記号である。１７１０行目の＋はスタックの先頭と比較される。結果がｔｒｅｅの時はＡＣＴ（　）内のルーチンを実行する。このルーチンは次のように実行される。＊は１文字スタックに読み込む操作を示す。ｉｄｅｎｔｉｆｉｅｒの場合サブルーチン呼び出しと，ジャンプがありうる。この区別は次のように行われる。identifierの次に'）'があればジャンプ，これ以外の記号があればサブルーチンの呼び出しになる。整数は意味処理ルーチン名である。ＲＥＴＵＲＮはサブルーチン呼び出しに対する戻りを意味する。１９７０行目における'＜'と'＞'で囲まれた部分は，スタックの先頭と'（'を比較して結果がｆａｌｓｅの場合実行される部分である。

次にＬＡＳＡＬによる構文解析プログラムの作製についてこの適用結果を述べる。ここではＦＯＲＴＲＡＮ ＪＩＳ ３０００の構文規則に対して，どのような修正を行ってＬＡＳＡＬシステムの入力にしたかをまず述べる。ここでは代表的な２つの方法を述べる。

⑴ 共通項くくり出し型

次のような構文生成規則を考えてみる。$A :: = \omega\gamma_1 \mid \omega\gamma_2$　ここでωから導出される記号列に長さがｋ以上の

ものがあればＡＬＬ(k)の条件を満さない。このような場合はＡ：：＝ ωＢとＢ：：＝ r_1 ｜ r_2 とすればよい。この方法で解決されるものが最も多かった。

(2) 言語拡張型

ＡＬＬ(k)文法で生成できない言語に対しては，この言語を拡張する。ただしこの拡張された言語はＡＬＬ(k)文法で生成されるものでなければならない。ＦＯＲＴＲＡＮはＡＬＬ(k)文法で生成できないが，次の構文生成規則の存在がその原因の一つである。

＜論理一次子＞：：＝ （ ＜論理式＞ ）

＜論理一次子＞：：＝ ＜算術式＞＜関係子＞＜算術式＞

拡張された言語においては，もとの文法で許されない文がある。このチェックは意味処理で行われる。

ＦＯＲＴＲＡＮ ＪＩＳ ３０００のＦＰＬによる構文解析プログラムは約３２０ステップであった。処理速度に関しては，構文解析的に複雑である次のタイプのプログラムの時間をＦＯＲＴＲＡＮ Ｇコンパイラの全コンパイル時間と比べると約５％であった。

ＡＡＡ＝ＢＢＢ＋ＣＣＣ＋ＤＤＤ

残されている最も大きな最適化の余地は，出現頻度の大きい順にトークンの比較を行うようＦＰＬプログラムを改善することである。さらに算術式に対しては特別の先き読みを行う事が有効である。

5. ＬＲ(k)アナライザⅡ

5.1 機能の概説

ＬＲ(k)アナライザⅡは，先に述べたようにプログラミング言語の文法をＢＮＦで記述したものから，その言語の構文解析ルーチン（またはトランスレータ）を，「表」の形で自動生成するプログラムである。この表を構文解析表と呼ぶ。（図5.1参照）。

このシステムは，ＳＬＲ(k)[6]，あるいはＬＡＬＲ(k)[7]パーサの構成アルゴリズムを，計算機で扱いやすい形に直した手法を用いてパーサを生成し，それに最適化処理を施した表を作成する。

構文解析表に，その表のインタプリタを合わせればパーサとなるが，このパーサは，一種の決定性プッシュダウンオートマトンであり，構文解析表が，このオートマトンの遷移表の役割を果す。この表では表わせない処理を行ないたいときは，アクションルーチン（意味処理，あるいは特殊処理ルーチン）を作り，表には，アクションルーチンの先頭番地を入れておけばよい。（図5.2参照）

表の各行が，構文解析時の基本オペレーションを表わすので，このオペレーション（6

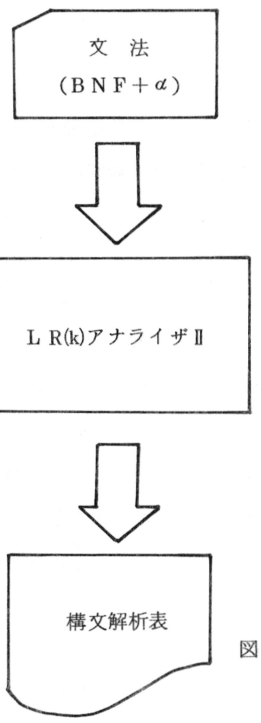


図5.1 ＬＲ(k)アナライザⅡの概要

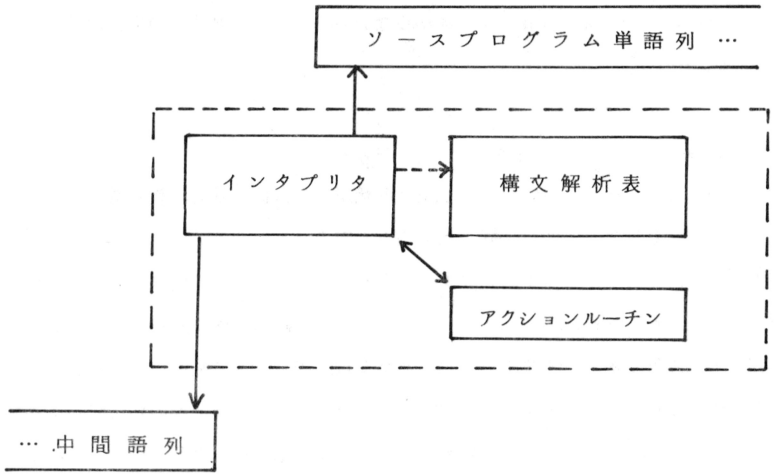


図 5.2　LR(k)パーサの構成

～10種類）をマクロ命令であると見なせば，インタプリティブ方式でなく，直接実行方式のパーサを作成するのにも構文解析表を役立てることができる。

5.2　構文解析オートマトンの生成

入力文法はBNFで記述するが，構文規則の他に，還元の際にパーサが出力する中間語を構文規則の直後に書いたり，アクションルーチン名を構文規則の右辺に挿入したりすることができるようにした。これは，意味処理や特殊処理を行なうタイミングを考え易くすることをねらったものである。また，能率のよいパーサを生成させるために，意味処理のない構文規則を，BNF中で明示することができるようにした。これにより，生成されるパーサに無用な還元を行なわせないようにした。

図5.3は，入力文法の記述例である。((　　))で囲まれている記号が出力記号であり，≪　　≫で囲まれているのが，アクションルーチン名である。！印が付けられている構文規則は，意味処理を行なわない規則である。

なお，BNF中で，空記号を使用してもよい。

```
<GOAL>::=<S>;
<S>     ::=<PRIMARY>!|<E>=<E>  (( = ))
<E>     ::=<TERM>!|<E>+<TERM>  (( + ))
<TERM>::=<TERM>*<PRIMARY>  (( * ))|
            <PRJMARY>!
<PRIMARY>::=( <E> )!|IDENTIFIER ≪ OUT ID ≫|
            CONSTANT≪ OUT CONST ≫
```

図 5.3　入力文法Gの記述

LR(k)アナライザⅡは，入力文法に関する情報を整理した後，特性有限状態オートマトンを作る。計算機での処理を簡単にさせるため，各構文規則の右辺の記号の間に，ほぼ機械的に状態名を挿入して，特性オートマトンを作るように

した。[9]（従って，configuration setは作らない）。つまり，P番目の構文規則

$$A \rightarrow a_1\ a_2 \cdots\cdots a_n$$

に対しては，

$$A\ \ a_1\ s_1\ a_2\ s_2 \cdots\cdots a_{n-1}\ s_{n-1}\ a_n\ \#_p$$

とする。ここでs_i（$i=1,\cdots\cdots,n-1$）は，読込み状態，$\#_p$は，還元状態である。

このとき，明らかに同じ状態に対しては，同じ状態名を対応させる。

次に，この特性オートマトンを，プッシュダウンオートマトンに変換する。状態の作り方より，このプッシュダウンオートマトンは，不適合状態（inadequate states：入力記号の先読みが必要となる状態）以外にも，決定性でない状態が存在する可能性もある。そのような場合には，状態の統合・分割・消去を実施して，決定性に直す。

この方式は，LR(0) machine の構成法が単純であることと，オートマトンの各状態が，構文規則中のどの部分に対応するかが明瞭であるという利点とがある。特性オートマトンは印字出力されるので，これを見れば，状態全体を一目で見渡すことができ，パーサが生成されたとき，それの動きがつかみ易くなる。

入力文法がLR(0)文法でないとき（つまり不適合状態が存在するとき），LR(k)アナライザⅡは，不適合状態に対し，まず1記号先までの単純先読み集合（simple look ahead sets）を計算する。入力文法がSLR(1)文法にならないときは，SLR(1)でない不適合状態に対してのみ，LALR(k)手法を使って，必要とあれば，最高3記号先までの先読み集合を計算する。

始めからLALR(k)手法を適用しない理由は，SLR(k)手法の方が，先読み状態の個数が少なくなるからである（LALR(k)手法では，不適合状態だけでなく，それをアクセスする状態にも依存して先読み集合が決まるため）。

図5.3の文法Gは，任意のkに対し，SLR(k)文法にならないが，LALR(1)文法になる例である（構文規則＜S＞→＜PRIMARY＞を取り除けばSLR(1)文法になる）。

LR(k)アナライザⅡは，この文法Gから，図5.4に示すプッシュダウンオートマトンを生成する（実際には遷移図の形で出力する）。

図5.4での記法の意味は，以下の通りである。

(i) Ⓐ ……Aという名前の状態

Ⓐ ……状態Aは，スタックに積まれる状態名である。

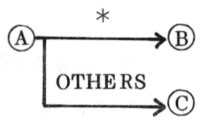
(ii) Ⓐ —c→ Ⓑ ……状態Aで，入力記号cが現われたら，それを読込んで状態Bに遷移する。

(iii) Ⓐ —TOP i→ Ⓑ ……状態Aで，スタックの一番上に乗っている状態名を見たとき，それがs_iであるときには，状態Bに遷移する（スタックのトップの状態名を見ることをlook backと呼ぶ）。

(iv) Ⓐ —OUT c / POP i→ Ⓑ ……状態Aでは，記号cを出力し，スタックをi個ポップアップして，状態Bに遷移する（出力記号のないときもある）。

(v) Ⓐ ——▽ S_i ……状態Aから出ている遷移は，状態S_iから出ている遷移と全く同じである。

なお，"OTHERS"は，「その他」の意味である。例えば



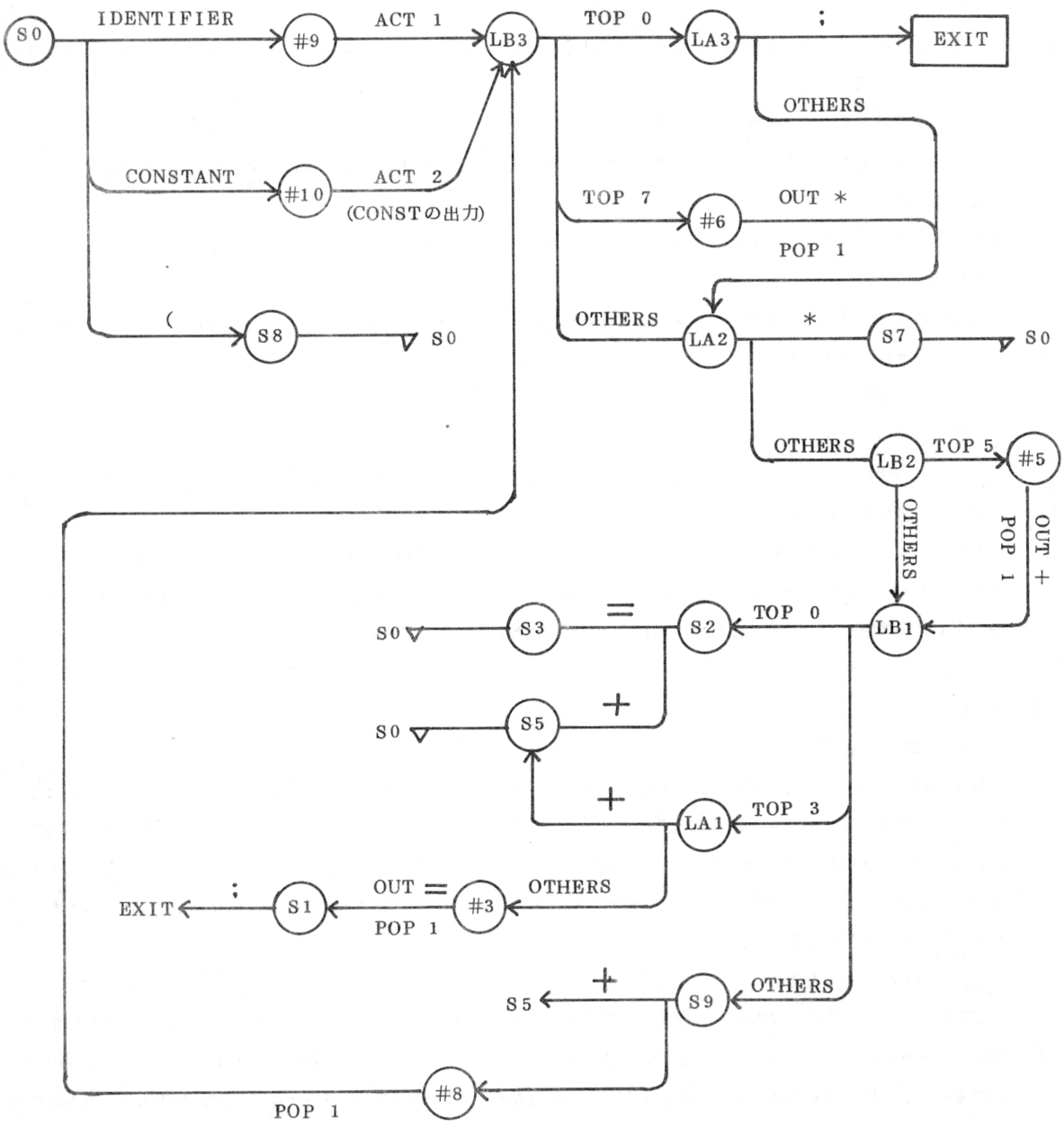


図 5.4　文法Gの構文解析オートマトン

（初期状態：S0，最終状態：EXIT）

は，「状態Aで，入力記号＊を見たとき，それを読込んで状態Bに移る。その他の記号を見たときは，読込まないで直ちに状態Cに遷移する」ことを表わす。

次に，ＬＲ(k)アナライザⅡが行なう最適化処理を４つ述べる。

(1)　１つの読込み状態（入力記号を読込む状態）から出ている終端遷移群が，他の読込み状態から出ている遷移群に含まれるときには，遷移群の共有を行なう。

(2)　look　backにより遷移先を決定するとき，look　back表が小さくて済むような順序で，スタックのトップの状態名を調べる。

(3)　意味処理やスタック操作のない還元状態（入力文法中で明示されたものに限る）を消去する。

(4)　１文字の先読みを行なうときには，可能な限り，読込みと分岐という動作で済ますようにし，入力記号の仮読みは行なわないようにする。

図5.3の入力文法Gは，ＬＡＬＲ(1)文法であっても，図5.4のパーサでは，入力記号の仮読みは，全然行なわないで済んでいる。

実用のコンパイラの構文解析ルーチンとして使うときには，ＬＲ(k)アナライザⅡによる自動最適化の他に，人手によってもパーサを最適化している。この段階で行なう処理は，以下のようなものである。

(1)　算術式の刈込みを行なう。

(2)　言語の特性を用いて構文解析表を小さくする。

(3)　パーサをシミュレートして，動作解析し，出現頻度の大きい記号列が速く構文解析されるようにする。

5.3　長所および問題点

ＦＯＲＴＲＡＮ，ＡＬＧＯＬ，ＳＮＯＢＯＬ４，ＰＣＬ（制御用計算機言語），その他いくつかの問題向き言語などの文法をＬＲ(k)アナライザⅡに通してみた。このうち，ＦＯＲＴＲＡＮについては，ＨＩＴＡＣ１０用常駐コンパイラ[8]や，小型計算機用のコンパイラなどで，ＬＲ(k)パーサが実際に使われている。

ＬＲ(k)アナライザⅡおよび生成されるパーサについて，その長所，問題点に関して述べる。

長　所

(1)　文法の広さ

ＬＡＬＲ(k)文法は，広い範囲の言語族を生成するので，言語の構文をＢＮＦで記述する際に，文法上の制約をあまり気にしないで書くことができる。ＦＯＲＴＲＡＮ（ＪＩＳ水準７０００）の場合で構文規則数は１７０個，ＡＬＧＯＬ（ＪＩＳ水準７０００）の場合では１５０個で文法を記述することができた。そして，ＦＯＲＴＲＡＮもＡＬＧＯＬも，ほぼＳＬＲ(1)文法になった（ただし，key 語は予約語とした）。これまでの実例で，ＳＬＲ(1)文法でないものもあったが，それもＬＡＬＲ(1)文法にはなった。

(2)　構文解析表の大きさ

構文解析表の大きさは，表のデータ構造や出力すべき中間語の形などにより変わるが，大まかに言えば，構文規則の個数を６～９倍したバイト数程度であり，かなり小さい。ＦＯＲＴＲＡＮ７０００の場合で１４１６バイト，SNOBOL４の場合（構文規則数６７個）で５５６バイトであった（なお，この数値は，リバースポリッシュを基本とする中間語を設定したときのものである）。

(3)　パーサの能率

パーサの能率については，まだ評価を終えていないが，ＳＬＲ(1)パーサを用いて作られたＨＩＴＡＣ１０のFORTRANコンパイラの場合では，１ステートメントあたりのコンパイルタイムは１９ｍｓｅｃ であった。ＬＲ(k)の手法に基づい

て生成されたパーサが実用的なものになることが言える。

なお，一般に，ＬＲ(k)構文解析手法では，オートマトンでの還元状態に来たとき，どの構文規則を適用するべきかということは，すでに分かっているので，還元処理が速い。また，ＬＲ(k)アナライザⅡで作りだすパーサには，無駄な還元状態がないので，能率は良くなっていると考えられる。

問　題　点

(1)　ＬＲ(k)アナライザⅡでは，遷移先を決定するために必要な状態名だけを，スタックに積んでいるが，それでもなお，スタック状態を少なくする余地が残されている。それは，現在のところ，意味処理の等しい還元状態を統合していないからである。入力文法中で形式的な記述を与えることにより，還元状態の統合を行なうようにすれば，状態数の減少，およびスタック動作の回数の減少により，パーサの能率の向上をもたらすことができる。この点は改善したい所である。

(2)　＜式＞の刈込みの必要性

ＬＲ(k)構文解析手法では，Ａ＝Ｂのような単純な式の場合，Ｂを読込んだ後，次の入力記号が，＊＊，＊，／，＋，－でないかどうかを，オートマトンのパスを遷移しながら，演算子順位の高いものから順にチェックして行くので能率が落ちる。しかし，現実には，このような単純な式の出現頻度が大きいので，問題である。これの対策として，我々は，Ａ＝Ｂのような単純な形をした式が特別速く構文解析されるようなパスを系統的にオートマトンに組入れる（式の刈込みを行なう）という策を採った（これによって構文解析表は1.5％しか増大しないので，メモリの方は特に問題は生じない）。

最後に，ＬＲ(k)アナライザⅡはＦＯＲＴＲＡＮで書かれていて，約４５００ステートメントのプログラムである。

6.　おわりに

このシステムは現在開発を終り，ＦＯＲＴＲＡＮなどに適用し，その評価を行っている段階である。

通常のコンパイラでは語彙解析部分，構文解析部分に演算時間のそれぞれ２０％，５％程しかついやしていないので，能率の点では余り問題ないと判断している。今後は誤りのあるプログラムに対する処理や，意味処理をやりやすくし，更に使いやすくして行く予定である。

参考文献

1)　Lewis, P.M.II and Stearns, R.E. : Syntax－directed transduction. JACM 15 (July 1968), 465 － 488.

2)　Backes, S : Top － down Syntax analysis and Floyd Evans Production Language. IFIP, 1971.

3)　吉村，西野：　能率のよい構文解析システムの自動作成について　　昭４７情報処理学会大会予稿集。

4)　西野，吉村，梅野：　Ｆｌｏｙｄプロダクションプログラム自動作成システム　　昭和４８電気関係学会関西支部連合大会予稿集。

5)　Knuth, D.E. : On the Translation of Languages from Left to Right. Information and Control. Vol.8, P607～P639, Dec. 1965.

6)　DeRemer, F. L.: Practical Translation for LR(k) Languages. Ph. D. thesis. MIT, Cambridge, Mass. Oct. 1969.

7) Lalonde, W. R.: An Efficient LALR(k) Parser Generator, Technical Report CSRG − 2, Tronto, Feb. 1971.

8) 加藤，中田，小島， LR(k)アナライザとそのFORTRANミニ・コンパイラへの適用 昭47情報処理学会大会，講演番号24。

9) 中田： LR(k) languageのtranslatorの例題とその作り方，private report.

本 PDF ファイルは 1965 年発行の「第 6 回プログラミング―シンポジウム報告集」をスキャンし、項目ごとに整理して、情報処理学会電子図書館「情報学広場」に掲載するものです。

この出版物は情報処理学会への著作権譲渡がなされていませんが、情報処理学会公式 Web サイトの https://www.ipsj.or.jp/topics/Past_reports.html に下記「過去のプログラミング・シンポジウム報告集の利用許諾について」を掲載して、権利者の捜索をおこないました。そのうえで同意をいただいたもの、お申し出のなかったものを掲載しています。

過去のプログラミング・シンポジウム報告集の利用許諾について

情報処理学会発行の出版物著作権は平成 12 年から情報処理学会著作権規程に従い、学会に帰属することになっています。

プログラミング・シンポジウムの報告集は、情報処理学会と設立の事情が異なるため、この改訂がシンポジウム内部で徹底しておらず、情報処理学会の他の出版物が情報学広場 (=情報処理学会電子図書館) で公開されているにも拘らず、古い報告集には公開されていないものが少からずありました。

プログラミング・シンポジウムは昭和 59 年に情報処理学会の一部門になりましたが、それ以前の報告集も含め、この度学会の他の出版物と同様の扱いにしたいと考えます。過去のすべての報告集の論文について、著作権者（論文を執筆された故人の相続人）を探し出して利用許諾に関する同意を頂くことは困難ですので、一定期間の権利者捜索の努力をしたうえで、著作権者が見つからない場合も論文を情報学広場に掲載させていただきたいと思います。その後、著作権者が発見され、情報学広場への掲載の継続に同意が得られなかった場合には、当該論文については、掲載を停止致します。

この措置にご意見のある方は、プログラミング・シンポジウムの辻尚史運営委員長 (tsuji@math.s.chiba-u.ac.jp) までお申し出ください。

加えて、著作権者について情報をお持ちの方は事務局まで情報をお寄せくださいますようお願い申し上げます。

期間：2020 年 12 月 18 日 〜 2021 年 3 月 19 日
掲載日：2020 年 12 月 18 日

プログラミング・シンポジウム委員会

情報処理学会著作権規程
https://www.ipsj.or.jp/copyright/ronbun/copyright.html