ソフトウェア基礎論　27—12
プログラミング言語　19—12
（1988. 12. 9）

自然言語の曖昧構文解析に対する並列処理

Parallel Processing of Ambiguous Syntax Analysis for Natural Language

安留　誠吾　　青江　順一
Seigo YASUTOME　Junichi AOE
徳島大学工学部情報工学科
Tokushima University

あらまし　本論文では，ＬＲパーサの手法をもちいて構文解析を効率的に行うための並列化ＬＲパーサを提案する．提案するＬＲパージングアルゴリズムでは，次の事柄を基本とする．
1）複数のスタックを分散し，１個のプロセッサが１個のスタックを管理する．
2）１個の特別なプロセッサ（ルートプロセッサ）は，他のプロセッサのスタックとの接続関係を管理し，子プロセッサと重複する動作を実行する．
3）プロセッサ間の通信により入力記号との同期をとり，通信を制限する．
4）曖昧構文を減少するために意味処理を導入する．

Abstract　In this paper, we propose a new parallel LR parsing scheme which is especially efficient with natural language grammars. The present algorithm has the following features.
1) A one-to-one correspondence between processors and parsing stacks.
2) A special processor (the root processor) manages connections among all other processors and controls multiple actions.
3) Elaborate broadcasting technique enables us to synchronize the input scanning and restrict to the number of broadcasts.
4) A simple semantic analysis is introduced to reduce ambiguous parse trees.

1．まえがき

ＬＲ文法はknuth[16]によって提案されたが，その文法に基づくパーサは空間的効率が悪く実用的なものではなかった．その後ＬＲ文法に制限を加えたＳＬＲ文法[9]やＬＡＬＲ文法[8]が考え出され，コンパクトなパーサの生成が可能となった．その結果，ＵＮＩＸシステムのコンパイラ・コンパイラツールＹＡＣＣ[15]として実用化され，多くのコンパイラに利用されている．

また，近年の自然言語処理研究が活発化するに伴って，富田[20]のようにＬＲパーサを英文の構文解析に利用する報告がなされている．ＬＲパーサを自然言語に適用する際には，曖昧な文の処理が問題となってくる．すなわち，曖昧な文法に対するＬＲパーサでは，解析動作が唯一に決定できない場合がある．そこで，富田はこれら曖昧な構文をすべて管理する多重スタックのＬＲパーサ（パーサＴと呼ぶ）を提案し，その複数のスタックをグラフ構造スタックとして処理するＬＲパージングアルゴリズムを与えた．しかし，複数のスタックに対する還元動作は，多くの並列操作を必要とするので，還元動作の処理効率の低下は免れない．また，自然言語処理に対して，構文解析と意味解析は融合されるべきである[17]ので，還元動作に意味処理

を付加した場合，この処理効率の低下はより深刻な問題となってくる．

従って，本論文では，このパーサＴの複数のスタック管理を効率的に行うための並列化ＬＲパーサ（パーサＭと呼ぶ）を提案する．提案するＬＲパージングアルゴリズムでは，次の事柄を基本とする．

1）複数のスタックを分散し，１個のプロセッサが１個のスタックを管理する．
2）１個の特別なプロセッサ（ルートプロセッサ）は，他のプロセッサのスタックとの接続関係を管理し，子プロセッサと重複する動作を実行する．
3）プロセッサ間の通信により入力記号との同期をとる．但し，通信を行うのは，次の場合に限られる．
   a）あるプロセッサにおける動作が曖昧となったとき．
   b）ルートプロセッサでの還元動作が，子プロセッサのスタックに影響するとき．

提案された並列アルゴリズムは，次の特徴をもつ．

1）スタックの管理が簡単になる．
2）プロセッサと解析結果が１対１に対応するので，意味処理の導入も容易になる．

以下，2.では，文脈自由文法，ＬＲ（１）オートマトンについて述べ，3.では，パーサＴについて説明する．そして，4.で提案する並列アルゴリズムを与え，最後に，ＬＲ解析時の意味解析のタイミングについての考察を加える．

2．諸定義

2.1　文脈自由文法

文脈自由文法（context free grammar）を

$$G=(V_N,\ V_T,\ P,\ S)$$

で表す．ここで，V_N，V_Tはそれぞれ非終端語，終端語の有限集合であり，ＰはＡ→αなる生成規則の有限集合である．但し，$A\in V_N$であり，αは非終端語と終端語からなる記号列である．ＳはV_Nの要素で始記号と呼ばれる．

$V=V_N\cup V_T$として，Ｖ上のすべての有限長の記号列の集合をV^*で表し，V^+はV^*から空集合εを除いた集合を表す．そして，Ａ，Ｂ，Ｃ$\in V_N$；ａ，ｂ，ｃ$\in V_T$；Ｘ，Ｙ，Ｚ$\in V$；α，β，γ$\in V^*$なる記号列を使用する．

以後，添え字＊，＋はその関係の反射推移閉包（reflexive and transitiveclosure），推移閉包（transitive closure）を表すものとする．

2.2　ＬＲ（１）オートマトン

文法ＧのＶN，ＶT，Ｐにそれぞれ Ｓ’，＄，Ｓ’→Ｓ＄なる要素を加え，Ｓ’を始記号とする文法を以後使用する．但し，Ｓ’と＄は生成規則Ｓ’→Ｓ＄にのみ使用される．Ｐの要素はＳ’→Ｓ＄を第０番目として適当に順序づけられているものとし，第ｐ番目の要素を

$A_p\rightarrow\alpha_p$あるいは$A_p\rightarrow X_{p1}X_{p2}\cdots X_{pnp}$で表す．但し，$n_p\geqq 0$であり，$n_p=0$は$A_p\rightarrow\varepsilon$を表す．

文法Ｇに対するＬＲ（１）オートマトンを定義するために，ＬＲ（１）項とその集合（ＬＲ（１）状態と呼ばれる）の閉包（closure）を導入する．

文法ＧのＬＲ（１）項とは，[Ａ→α・β，ａ]のようなＰの任意の生成規則Ａ→αβの右辺に１個の・を付けたものと，先読みと呼ばれる終端語もしくは”＄”からなる．このとき，文法ＧのＬＲ（１）状態Ｓの閉包closure(s_j)は次のように得られる．

1．closure(s_j)をs_jに初期設定する．

2．[Ａ→α・Ｂβ，ａ]$\in$closure(s_j)であってＢ→γ$\in$Ｐかつｂ$\in$FIRST(βα)ならば，[Ｂ→・γ，ｂ]をclosure(s_j)に加える．そして，この手順を新たに加える項がなくなるまで繰り返す．但し，FIRST(α)とは，αから導出される文字列の先頭の終端記号の集合である．

これらの状態間の遷移を表すＧＯＴＯ関数を次のように定義する．

$$GOTO(s_j,X)=closure(\{[A\rightarrow\alpha X\cdot\beta,a]\mid [A\rightarrow\alpha\cdot X\beta,a]\in s_j\})$$

文法Ｇに対するすべての状態の集合Ｋは次のように求めることができる．

$$K=\{s_0=closure(\{[S'\rightarrow\cdot S\$,\dashv]\})\}$$

と初期設定し，以下の操作を新たな状態がＫに加えられなくなるまで繰り返す．

任意の$s_j\in K$とＸ$\in$Ｖに対して，ＧＯＴＯ（s_j，Ｘ）が空集合でなく，Ｋに入ってなかったらそれをＫに加える．

ここで，s_0は初期状態であり，{[Ｓ’→Ｓ＄・，⊣]}なる状態は最終状態である．そ

して，このＧＯＴＯ関数により定義されるオートマトンを文法Ｇに対するＬＲ（１）オートマトンＭと呼ぶ．以後，状態ｓ$_i$に対応する状態番号をｊで表す．従って，０は初期状態番号を表す．

オートマトンＭの[A→α・aβ,a]∈ｓ$_i$なる状態ｓ$_i$はシフト状態（shift state）と呼ばれ，[A_p→α_p・,b]∈ｓ$_i$なる状態ｓ$_i$は還元状態（reduce state）と呼ばれる．

ＡＣＴＩＯＮ（ｊ，ａ）は，状態ｓ$_i$と入力ａに対する動作の集合を表し，各動作は次のように定義される．

1) [A→α・aβ,b]∈ｓ$_i$ かつ GOTO(ｓ$_i$,a)=ｓ$_j$ ならば，{"sh j"}∈ACTION(j,a).
この動作は，１つの文字列を入力からパージングスタックへプッシュし，状態ｓ$_j$へシフトすることを表す．

2) [A_p→α_p・,b]∈ｓ$_i$ ならば，{"re p"}∈ACTION(j,a).
この動作は，ｐ番目の生成規則を用いてスタック上の内容を還元し，その生成規則番号ｐを出力することを表す．

3) [S'→S・$,┤]∈ｓ$_i$ かつ a="$" ならば，{"acc"}∈ACTION(j,a).
この動作は，入力文字列が受理されたことを表す．

4) [A→α・bβ,a]∈ｓ$_i$ かつ a≠b ならば，{"error"}∈ACTION(j,a).
この動作は，入力文字列が受理されなかったことを表す．

ＡＣＴＩＯＮ（ｊ，ａ）が２つ以上の動作を持つとき，状態ｓ$_i$は曖昧状態（ambiguous state）と呼ばれる．

シフト状態と還元状態と曖昧状態の集合をそれぞれK_s，K_r，K_aで表す．

また，オートマトンＭの状態ｓ$_i$が次の動作矛盾の少なくとも１つをもつとき，ｓ$_i$は不適合状態（inadequate state）と呼ばれる．

1．シフト動作と還元動作による矛盾．
2．異なる規則による還元動作の矛盾．

不適合状態が入力記号の先読み動作により解決できるとき，ＬＡＬＲ（１）パーサがオートマトンＭから構成できる．

［例１］次の生成規則を持つ文法G_1に対する解析表を表１に示す．なお”＊”で始まる文字列は終端語である．

1．S→NP VP
2．S→S PP
3．NP→*n
4．NP→ADJ *n
5．NP→NP PP
6．PP→*prep NP
7．VP→*v NP
8．ADJ→*adj
9．ADJ→*adj ADJ

表１　解析表

	ACTION(j,a)					GOTO(j,A)				
State	*adj	*n	*v	*prep	$	NP	VP	PP	ADJ	S
0	sh5	sh8				2			4	1
1				sh7	acc			6		
2			sh10	sh7			8	9		
3			re3	re3	re3					
4		sh11								
5	sh5	re8							12	
6				re2	re2					
7	sh5	sh8				13			4	
8				re1	re1					
9			re5	re5	re5					
10	sh5	sh8				14			4	
11			re4	re4	re4					
12		re9								
13			re6	sh7,re6	re6			9		
14				sh7,re7	re7			9		

解析表の左部分（action table）の空白（blank space）は，"error"を表す．

解析表の右部分（goto table）は，還元動作後に，進むべき状態を決定するための情報を持つ．

解析表において，ACTION(13,*prep)={"sh7","re6"}，ACTION(14,*prep)={"sh7","re7"}，であるので，状態S_{13}，S_{14}は，曖昧状態である．

3．　グラフ構造スタックによるＬＲ解析アルゴリズム

自然言語の持つ曖昧性のためにＬＲ解析表上には曖昧状態が存在するので，解析時に曖昧状態が出現すると動作数だけスタックをコピーしなければならない．しかし，コピーされたスタックの中には，他のスタックと重複した部分が多く存在するので，この重複した部分をまとめてパーサＴでは，グラフ構造スタックを使用する．以下，パーサＴの説明を行う．

3.1　グラフ構造スタック

グラフ構造スタックの定義を行う前に若干の用語を準備しておく．

有向グラフ（directed graph）を，節（node）

の集合Nと辺（edge）の集合Eの順序対（N, E）で定義する.

例えば＜v, w＞∈Eは，節vを始節（inital node）とし，節wを終点（terminal node）とする辺であると呼び，節vを節wの先祖（predecessor）と呼び，節wを節vの子孫（successor）と呼ぶ．また，辺＜v_{i-1}, v_i＞（但し，$1 \leqq i \leqq n$）が存在するならば，（v_0, v_1, ..., v_n）をv_0からv_nへの長さnのパス（path）と呼ぶ.

パス（v_0, v_1, ..., v_n）において，$v_0 = v_n$ならばこのパスをサイクル（cycle）と呼び，サイクルをもたないグラフを非サイクル（acyclic）と呼ぶ.

節vに入る辺の個数を節vの入次数（in-degree）と呼び，節vから出る辺の個数を節vの出次数（out-degree）と呼ぶ．また入次数が0である節をトップまたはルート（root）と呼び，出次数が0である節を葉（leaf）と呼ぶ.

グラフ構造スタックΓは，非サイクルであり，唯一の葉v_0は状態番号0でラベル付けされている．この特別な節v_0をΓの底（bottom）と呼ぶ．v_0からの距離が偶数である節を状態節（state node）と呼び，状態番号がラベル付されている．v_0からの距離が奇数である節を記号節（symbol node）と呼び，文法記号がラベル付けされている．状態節を○印で，記号節を□印で表す．以後，状態節をv, w, u∈Γで表し，記号節をx, y, z∈Γで表す.

3.2 パージングアルゴリズム

富田[20]に基づいたパージングアルゴリズムは，2.2で定義したACTIONと$GOTO(s_j, A_p) = s_i$ならば$goto(j, A_p) = i$となる関数gotoを解析表としてもち，以下の関数及び変数を使用する.

G: 文法．すなわち生成規則の集合.

$a_1 \ldots a_n$: 長さnの入力文字列.

Γ: グラフ構造を持つスタック.

STATE(v): 節vにラベル付けされた状態番号を返す.

SYMBOL(x): 節xにラベル付けされた記号を返す.

SUCCESSORS(v): 節vの先祖の節の集合を返す.

RESULT: RESULTがTRUEならば，文が受理されたことを表し，FALSEならば，文が受理されなかったことを表す結果フラグである.

AC_SET: 処理すべきΓの活性化節（active node）の集合.

RE_SET: 還元すべき節vと適用される生成規則番号pの組<v,p>の集合.

<v,p>∈RE_SETは，vを出発節とするパスに"re p"を適用すべきであることを意味する.

SH_SET: シフトすべき節の集合.

要素は<v,s>で表され，vはシフトすべき節，sは次に進むべき状態番号である．<v,s>∈SH_SETは，節vに"sh s"を適用すべきであることを意味する.

以下にLRアルゴリズムを記述する．但し，入力記号列$a_1 \ldots a_n$の後には，端を表すa_{n+1}="$"が付加されているものとする.

```
PARSE(G,a1...an+1)
  begin
    RESULT:=FALSE;
    s0でラベル付けされた節v0でΓを初期化する;
    AC_SETを{v0}で初期化する;
    for i:=0 to n do PARSEWORD(i);
    return RESULT
  end

PARSEWORD(i)
  begin
    RE_SET:=φ ; SH_SET:=φ ;
    repeat
      if (AC_SET≠φ) then ACTOR;
      elseif (RE_SET≠φ) then REDUCER;
    until (RE_SET=φ and AC_SET=φ);
    SHIFTER;
  end
```

```
ACTOR
  begin
    AC_SETから要素vを除去;
    λ:=ACTION(STATE(v),a_{i+1});
    if (λ="acc ") then RESULT:=TRUE;
    if (λ="sh j") then
        SH_SETに<v,j>を追加;
    if (λ="re p") then
        RE_SETに<v,p>を追加
  end

REDUCER
  begin
    RE_SETから要素<v,p>を除去;
    節vから距離2*n_pにある節をwとする;
    それぞれgoto(STATE(w),A_p),A_pでラベル
      付けされた節u,xをΓに作成;
    辺<u,x>,<x,w>をΓに作成;
    AC_SET:={u}
  end

SHIFTER
  begin
    SH_SETから要素<v,j>を除去;
    それぞれj,a_{i+1}でラベル付けされた節w,
      xをΓに作成;
    辺<w,x>,<x,v>をΓに作成;
    AC_SET:={w}
  end
```

PARSEは，RESULTの初期化；Γの初期化；AC_SETの初期化をする．入力記号ごとにPARSEWORDで解析を進める．

PARSEWORDは，RE_SET，SH_SETの初期化；それぞれが空になるまでACTOR，REDUCERを繰り返した後，SHIFTERを実行する．

ACTORは，AC_SETから要素vを除去し，ACTION(STATE(v),a_{i+1})の内容に従う動作をを実行する．

REDUCERは，RE_SETから要素<v,p>を除去した後，図1の処理を実行する．そして，AC_SETに{u}をセットする．

SHIFTERは，SH_SETから要素<v,s>を除去した後，図2の処理を実行する．そして，AC_SETに{w}をセットする．

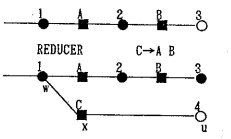

図1　REDUCERの説明図

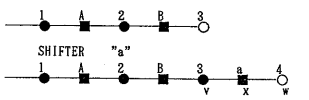

図2　SHIFTERの説明図

4. LR解析の並列化アルゴリズム

4.1 アルゴリズムの概要

パーサTには，次の特徴がある．

1) 曖昧状態に於ける複数の動作に対してシフト動作は，高々1個である．
2) 還元動作の処理がシフト動作の処理より複雑である．

この理由により，複雑な還元動作の処理は，子プロセッサに分散させ，ルートプロセッサは，シフト動作とこれら子プロセッサの管理を行うものとする．図3にシステムの構成を示す．

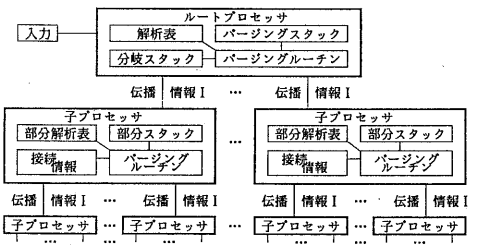

図3　システムの構成

複数のスタック処理を効率化するために，このような並列処理は有効であるが，提案されるべき並列化アルゴリズムでは，スタック内容の接続関係を保持し，且つ入力記号の同期を効率的に実現する必要がある．また，子プロセッサでも解析表が必要となるが，子プロセッサは還元動作のみを実行するので，このaction tableは一部だけの情報でよい．従って，各子プロセッサの持つべき部分解析表は，次のように定義できる．

［定義１］

$K' = \{ s_i \mid s_i = GOTO(s_j, A), s_i \in K \}$

とするとき，$K' \times V_T$上のaction tableと$K \times V_N$のgoto tableの組を部分解析表と定義する．

本論文で提案するパーサは，次の状況において子プロセッサに還元動作が伝播する．

1) ルートプロセッサにおいて曖昧状態となったとき．

2) ルートプロセッサでの還元動作が，子プロセッサの部分スタックに影響するとき．

1)の場合，新しい子プロセッサを登録し，子プロセッサに還元動作を実行させるための情報を通信し，そして，ルートプロセッサは，シフト動作が存在するならば，そのシフト動作を実行し，存在しないならば還元動作を実行する．

2)の場合，子プロセッサに還元動作を実行させるための情報を通信し，ルートプロセッサも還元動作を実行する．1)の場合，子プロセッサに通信する情報を伝播情報Ｉと呼び，以下のように定義する．

I=(FLAG,ACTION_SET,X,k,t(d_s,d_s+k))

FLAG: 制御フラグ．

ACTION_SET: 動作の集合．

X: 現在の入力記号．

k: 通信スタックの大きさ．

t(d_s,d_s+k): 子プロセッサに通信されるべき通信スタック．

k>0: d_s～d_s+kで決定される．

k=0: 部分スタックは空になる．

ルートプロセッサは，全ての子プロセッサの部分スタックとのつながりを管理するための分岐スタック（branch stack）をもち，その要素は以下のように定義される．

要素＜q, d_q＞

q: 子プロセッサの番号．

d_q: パージングスタックの深さ．

この分岐スタックにより次の特徴が得られる．

a) ルートプロセッサから子プロセッサへのパージングスタック内容の通信の簡素化．

b) 複数のプロセッサにおける冗長なシフト，還元動作の除去．

各子プロセッサでは，ルートプロセッサからの伝播情報Ｉをもとに以下の処理を行う．

1) 既に存在する子プロセッサに伝播情報Ｉを通信する．

2) 更に新しい子プロセッサq_{new}を登録し，その子プロセッサに伝播情報Ｉを通信する．

3) 還元動作を実行する．

4) エラーが検出されれば，親プロセッサに知らせる．

従って，子プロセッサは，自分の直接の子プロセッサを管理するためにプロセッサ番号を要素とする集合PRO_SETを持っている．

4.2 ルートプロセッサの解析アルゴリズム

以下にルートプロセッサの解析アルゴリズムを表す．但し，入力記号列$a_1 \ldots a_n$の後には，端を表すa_{n+1}="$"が付加されているものとし，次の変数，スタック，関数を利用する．

d_0: ルートプロセッサのパージングスタックの深さ．

q: 子プロセッサの番号．

d_q: プロセッサ番号qのパージングスタックの深さ．

B_STACK: 分岐スタック．

W_STACK: 作業用のスタック．

q_{new}: 新しく登録されるべきプロセッサ番号．

TRANS(q,I): プロセッサ番号qに伝播情報Iを通信する手続き．

SHIFT(l,X): パージングスタックP_STACKにX,lをプッシュする手続き．

REDUCE(p): パージングスタックを$2*n_p$ポップし，そのトップの要素をsとするとき，パージングスタックにA_p, GOTO(s,A_p)をプッシュする手続き．

```
PARSE(G,a1...an+1)
  begin
    RESULT:=FALSE;
    パージングスタックを0で初期化する;
    分岐スタックを初期化する;
    for i:=1 to n+1 do PARSEROOT(ai);
    return RESULT
  end
```

```
PARSEROOT(X)
  begin
    ACTION(sj,X)をACTION_SETにセットする
    if      (ACTION_SET={"acc"})   then
        RESULT:=TRUE;
    elseif (ACTION_SET={"error"}) then
        exitする;
    elseif (ACTION_SET={"sh l"})  then
        SHIFT(l,X);
    elseif (ACTION_SET={"re p"})  then
        PRE_REDUCE(p,X);
    else AMBIGU(ACTION_SET,X);
  end
```

PARSEは，RESULT，パージングスタック，分岐スタックを初期化し，入力記号ごとにPARSEROOTを繰り返す．
PARSEROOTは，ACTION(s_j,X)をACTION_SETにセットし，ACTION_SETの内容により処理を振り分ける．

次に，PRE_REDUCE(p,X),AMBIGU(ACTION_SET,X)について示す

```
PRE_REDUCE(p,X)
  begin
    k:=dø-2*np;
    forever
      begin
        分岐スタックB_STACKをポップしその
          要素を<q,dq>とする;
        if (k>dq) then
          begin
            <q,dq>を分岐スタックB_STACKに
              プッシュする;
            break
          end
        TRANS(q,(1,{"re p"},X,0,*));
        スタックW_STACKにqをプッシュする
      end
    REDUCE(p);
    repeat
      スタックW_STACKをポップしその要素を
        qとする;
      <q,dø>を分岐スタックB_STACKにプッシュする
    until(スタックW_STACKが空)
  end

AMBIGU(ACTION_SET,X)
  begin
    if ("sh l"∈ACTION_SET) then
        FLAGを0にセットし，ACTION_SETから"sh l"を除去する;
    else FLAGを1にセットする;
    BROADCAST(FLAG,ACTION_SET);
    if (FLAG≠1 or r≠1) then
        NEW_BROAD(FLAG,ACTION_SET);
    if (FLAG=1) then SHIFT(l,X);
    else REDUCE(p1,X);
  end
```

PRE_REDUCE(p,X)は，還元動作が他のプロセッサのスタックに影響するかどうかを調べ，影響する場合は，そのプロセッサに必要な情報を通信する．
AMBIGU(ACTION_SET,X)は，ACTION_SETの内容によりFLAGをセットし，ACTION_SETから"sh l"を除去し，BROADCASTを呼び，ACTION_SETの要素が"re p"だけでないときNEW_BROADを呼ぶ．そして，FLAGが1ならばシフト動作を実行し，1でないならば，還元動作を実行する．

次に，BROADCAST(FLAG,ACTION_SET),
NEW_BROAD(FLAG,ACTION_SET)について示す．

```
BROADCAST(FLAG,ACTION_SET)
  begin
    repeat
      分岐スタックB_STACKをポップし，その要素を<q,dq>とする;
      TRANS(q,(FLAG,ACTION_SET,X,dø-dq
        ,t(dq,dø)));
      スタックW_STACKにqをプッシュする
    until(分岐スタックが空);
    repeat
      スタックW_STACKをポップしその要素
        をqとする;
      <q,dø>を分岐スタックB_STACKにプッシュする
    until(スタックW_STACKが空)
  end
```

```
NEW_BROAD(FLAG,ACTION_SET)
    for i:=FLAG+1 to r do
      begin
        <q_new,d_0>を分岐スタックB_STACKに
          プッシュする;
        TRANS(q_new,(-1,{"re p_i"},X,
          d_0-2*n_pi,t(0,d_0-2*n_pi)));
      end
```

BROADCAST(FLAG,ACTION_SET)は，既に存在しているプロセッサに伝播情報を通信し，分岐スタックを変更する．

NEW_BROAD(FLAG,ACTION_SET)は，r-FLAG個の新しいプロセッサを分岐スタックに登録し，それぞれのプロセッサに伝播情報を通信する．

4.3 子プロセッサの解析アルゴリズム

以下にプロセッサqにおける還元動作の解析アルゴリズムを示す．但し，次の変数，関数を利用する．

PRO_SET={$q_1,q_2,\ldots,q_m$}：直接の子を管理するための集合．

ERROR(q)：親プロセッサのPRO_SETからqを除去する手続き．

REDUCE(p)：アルゴリズム1と同じ．

TRANS(q,I)：アルゴリズム1と同じ．

SEMI_REDUCE(p)：手続きREDUCE(p)からパージングスタックを$2*n_p$ポップする処理を除いた手続き．

```
PARSES(I)
  begin
    if (FLAG≠-1) then
        for i:=1 to m do TRANS(q_i,(FLAG,
          ACTION_SET,X,k,t(d_s,d_s+k)));
    通信スタックt(d_s,d_s+k)を部分スタック
      に付加する;
    if ((FLAG≠1 or r≠1) or FLAG≠-1)
      then NEW_BROAD(FLAG,ACTION_SET);
    if (FLAG=0) then exit;
    else SEMI_REDUCE(p);
    forever PARSESON(1,X);
    return RESULT;
  end
```

```
PARSESON(1,X)
  begin
    ACTION(1,X)をACTION_SETにセットする;
    if     (ACTION_SET={"acc"})   then
        exit;
    elseif (ACTION_SET={"error"}) then
        ERROR(q);
    elseif (ACTION_SET={"sh 1'"}) then
        exit;
    elseif (ACTION_SET={"re p"})  then
        REDUCE(p);
    else AMBIGU(ACTION_SET,X);
  end
```

PARSESは，FLAGが-1でないならば，集合PRO_SETの要素に伝播情報を通信する．そして，通信スタックt(d_s,d_s+k)を部分スタックに付加しし，ACTION_SETの要素が"re p"だけでない，またはFLAGが-1でないならばNEW_BROADを呼ぶ．そして，FLAGが0ならばexitし，0でなければ還元動作を実行する．そして，PARSESONを呼び続けEXITに出会うとRESULTを返す．

PARSESONは，ACTION(s,X)をACTION_SETにセットし，ACTION_SETの内容により処理を振り分ける．

次に，AMBIGUSON(ACTION_SET,X)について示す．

```
AMBIGUSON(ACTION_SET,X)
  begin
    if ("sh 1'"∈ACTION_SET) then
        FLAGを0にセットし，ACTION_SETか
          ら"sh 1'"を除去する;
    else FLAGを1にセットする;
    if (FLAG≠1 or r≠1) then
        NEW_BROAD(FLAG,ACTION_SET);
    if (FLAG≠1) then REDUCE(p_1);
  end
```

AMBIGUSON(ACTION_SET,X)は，ACTION_SETの内容によりFLAGをセットし，ACTION_SETから"sh 1'"を除去し，ACTION_SETの要素が"re p"だけでないならばNEW_BROADを呼ぶ．そして，FLAGが1でないならば，還元動作を実行する．

次に，NEW_BROAD(FLAG,ACTION_SET)について示す．

```
NEW_BROAD(FLAG,ACTION_SET);
    for i:=FLAG+1 to r do
      begin
        qnewを集合PRO_SETに追加する;
        TRANS(qnew,(-1,{"re pi"},X,
          dq-2*npi,t(0,dq-2*npi)));
      end
```

NEW_BROAD(FLAG,ACTION_SET)は，r-FLAG個の新しいプロセッサを集合PRO_SETに登録し，それぞれのプロセッサに伝播情報を通信する．

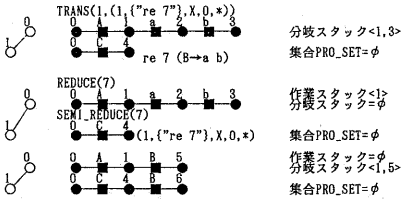

図４　ルートプロセッサにおいて還元動作が子プロセッサに影響する説明図

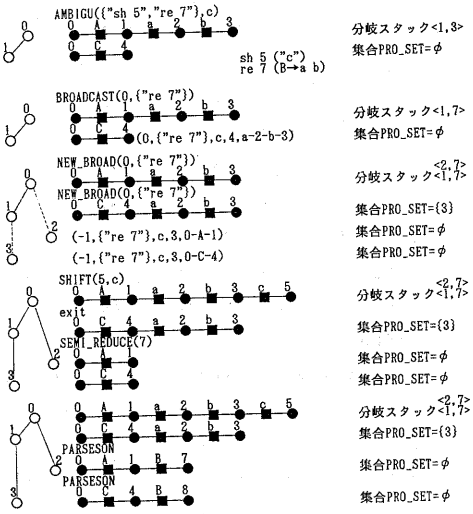

図５　ルートプロセッサにおいて曖昧状態になった例の説明図

［定理１］パーサＴとパーサＭの受理する言語のクラスＬ（Ｔ）とＬ（Ｍ）は等価である．

（証明略）

5.　意味処理の付加

自然言語においては，構文解析の途中に早期意味解析を付加することにより不適当な構文解析結果を削除することができる[17]．本論文におけるＬＲ解析では，プロセッサとパージングスタックが１対１に対応しているので，１個のプロセッサが１個の解析結果を保持していることを意味する．この特徴により，意味処理の付加は極めて容易になる．

また，ＬＲ解析後に意味処理を実行したのでは，余分なＬＲ解析を実行したこととなる．そこで次の手順で意味処理を実行する．

1）曖昧状態となったスタック上の深さを保持しておく．

2）1)で保持した深さを越えて還元動作が実行された後，意味処理を実行する．

つまり，曖昧状態となったことは，そのときの入力単語に曖昧さが含まれていることを意味している．しかし，その時点ではその入力単語が，いずれの句または単語を修飾しているかが判定できないので，その状態を越えて還元動作を実行した後に限られる．意味処理の後，意味的不整合の生じたプロセッサは，以後解析は不要となる．

図６，７で，この意味処理の導入を説明する．

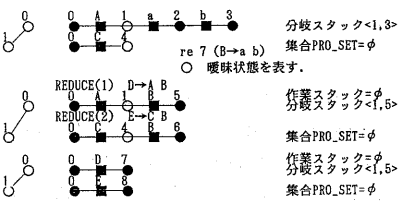

図６　意味処理実行のタイミング

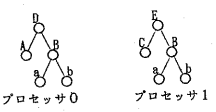

図７　解析結果の木構造

図６の解析結果を木構造で表すと図７のようになる．このように，"a"で曖昧状態となり，その曖昧状態を還元動作により越えた時に始めて"a"の意味関係が判明する．よってこの時点で意味処理が実行可能となる．

6. まとめ

以上，本論文では，富田の提案したパーサTの複数のスタック管理を効率的に行うための並列化ＬＲパーサを提案した．提案したＬＲパージングアルゴリズムは，１個のプロセッサが１個のスタックを管理し，重複する動作の実行を削除するように設計されている．また，プロセッサ間の通信は，あるプロセッサにおける動作が曖昧となったときと，ルートプロセッサでの還元動作が，子プロセッサのスタックに影響するときに限られている．よって提案されたＬＲパージングの並列化により，１個のプロセッサが１個の解析結果を保持していることになり，意味処理の付加が容易になる．

実際にインプリメントする際には，ＬＲ解析の途中で解析結果が受理されないと確認されたスタックをもつプロセッサに新しくスタックを割り当てる手法などが導入されなければならない．今後，意味処理がＬＲ解析中に効率よく実行されるために，解析結果の保持の仕方，意味処理のための知識表現の導入を検討しなければならない．

参考文献

(1) Aho,A.V., Ullman,J.D.: "The Theory of Parsing,Translation and Compiling": Prentice-Hall (1972).

(2) Aho,A.V., Johnson,S.C., Ullman,J.D.: "Deterministic parsing of ambiguous grammars": Comm.ACM18,8 (1975).

(3) Aho,A.V., Ullman,J.D.: "Principles of Compiler Design": Addison-Wesley (1977).

(4) Bochmann,G.V.: "Semantic Evaluation from Left to Right": Comm.ACM 19,2 (1976).

(5) Cohen,J., Roth,M.S.: "Analyses of Deterministic Parsing Algorithms": Comm.ACM 21,6 (1978).

(6) Cohen,R.: "A Computational Theory of the Function of Clue Words in Discourse": Proceedings of COLING84 (1984).

(7) Cohen,R.: "Interpreting Clue in Conjunction with Processing Restrictions in Arguments and Discourse": AAAI-87 (1987).

(8) DeMemer,F.L.: "Practical Translations for LR(k) Languages": Ph.D.dissertation,M.I.T.,Cambridge, Mass (1969).

(9) DeMemer,F.L.: "Simple LR(k) grammars": Comm.ACM 14,7 (1971).

(10) Eiselt,K.P.: "Recovering from Erroneous Inferences": AAAI-87(1987).

(11) Finn,G.D.: "Extended Use of Full Productions in LR(1) Parser Applications": Comm.ACM 28,9 (1985).

(12) Hovy,E.H.: "Interpretation in Generation": AAAI-87 (1987).

(13) Hunt III,H.B., Szymanski,T.B., Ullman,J.D.: "On the Complexity of LR (k) Testing": Comm.ACM 18,12 (1975).

(14) Hunt III,H.B., Szymanski,T.G., Ullman,J.D.: "Operations on Sparse Relations": Comm.ACM 20,3 (1977).

(15) Johnson,S.C.: "YACC--Yet Another Compiler Compiler": Technical Report CSTR 32, Bell Laboratories (1975).

(16) Knuth,D.E.: "On the translation of Language from left to right": Inf.& Control.,8,6 (1965).

(17) Mellish,C.S.: "Computer Interpretation of Natural Language Descriptions": Ellis Horwood Limited (1985).

(18) Mickunas,M.D., Modry,J.A.: "Automatic Error Recovery for LR Parsers": Comm.ACM 21,6 (1978).

(19) Rich,E., Barnett,J., Wittenburg,K., Wroblewski,D.: "Ambiguity Procrastination": AAAI-87 (1987).

(20) Tomita,M.: "Efficient Parsing for Natural Language": Kluwer Academic Publishers,(1986).

(21) 青江，山本，原田，島田：”弱順位関数の一構成法”：信学論（Ｄ）Vol.J60-D No.1 (1977).

(22) 中田：”コンパイラ”：産業図書（株）(1981).

盛光印刷所