

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パーシ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。L

R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パーシ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パーシ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 $A \rightarrow \alpha$ 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、 $A, B, C \in V_N$; $a, b, c \in V_T$; $X, Y, Z \in V$; $\alpha, \beta, \gamma \in V^*$ 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 transitive closure), 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 V_N 、 V_T 、 P にそれぞれ S' 、 $\$, S' \rightarrow S\$$ 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 $S' \rightarrow S\$$ にのみ使用される。 P の要素は $S' \rightarrow S\$$ 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 $A_p \rightarrow X_{p1}X_{p2}\cdots X_{pn_p}$ で表す。但し、 $n_p \geq 0$ 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、 $[A \rightarrow \alpha \cdot \beta, a]$ のような P の任意の生成規則 $A \rightarrow \alpha \beta$ の右辺に1個の $\cdot$ 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 $\text{closure}(s_j)$ は次のように得られる。

1. $\text{closure}(s_j)$ を s_j に初期設定する。
2. $[A \rightarrow \alpha \cdot B\beta, a] \in \text{closure}(s_j)$ であって $B \rightarrow \gamma \in P$ かつ $b \in \text{FIRST}(\beta\alpha)$ ならば、 $[B \rightarrow \cdot \gamma, b]$ を $\text{closure}(s_j)$ 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、 $\text{FIRST}(\alpha)$ 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。

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

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。

$$K = \{s_a = \text{closure}(\{[S' \rightarrow \cdot S\$, \mid]\})\}$$

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、 $\text{GOTO}(s_j, X)$ が空集合でなく、 K に入っていなかったらそれを K に加える。

ここで、 s_a は初期状態であり、 $\{[S' \rightarrow S\$, \mid]\}$ 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_j 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 $[A \rightarrow \alpha \cdot a \beta, a] \in s_j$ なる状態 s_j はシフト状態(shift state)と呼ばれ、 $[A_p \rightarrow \alpha_p \cdot, b] \in s_j$ なる状態 s_j は還元状態(reduce state)と呼ばれる。

ACTION(j, a)は、状態 s_j 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) $[A \rightarrow \alpha \cdot a \beta, b] \in s_j$ かつ $GOTO(s_j, a) = s_i$ ならば、 $\{ "sh\ i" \} \in ACTION(j, a)$ 。
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_i へシフトすることを表す。
- 2) $[A_p \rightarrow \alpha_p \cdot, b] \in s_j$ ならば、 $\{ "re\ p" \} \in ACTION(j, a)$ 。
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) $[S' \rightarrow S \cdot \$, \$] \in s_j$ かつ $a = "\$"$ ならば、 $\{ "acc" \} \in ACTION(j, a)$ 。
この動作は、入力文字列が受理されたことを表す。
- 4) $[A \rightarrow \alpha \cdot b \beta, a] \in s_j$ かつ $a \neq b$ ならば、 $\{ "error" \} \in ACTION(j, a)$ 。
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_j 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_j が次の動作矛盾の少なくとも1つをもつとき、 s_j 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. $S \rightarrow NP\ VP$ | 6. $PP \rightarrow *prep\ NP$ |
| 2. $S \rightarrow S\ PP$ | 7. $VP \rightarrow *v\ NP$ |
| 3. $NP \rightarrow *n$ | 8. $ADJ \rightarrow *adj$ |
| 4. $NP \rightarrow ADJ\ *n$ | 9. $ADJ \rightarrow *adj\ ADJ$ |
| 5. $NP \rightarrow NP\ PP$ | |

表1 解析表

State	ACTION(j,a)					GOTO(j,A)				
	*adj	*n	*v	*prep	\$	NP	VP	PP	ADJ	S
0	sh5	sh3				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	sh3				13			4	
8			re1	re1						
9			re5	re5	re5					
10	sh5	sh3				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 $\langle v, w \rangle \in E$ は、節 v を始節 (initial 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 $\langle v_{i-1}, v_i \rangle$ (但し、 $1 \leq i \leq n$) が存在するならば、 $(v_0, v_1, \dots, v_n)$ を v_0 から v_n への長さ n のパス (path) と呼ぶ。

パス $(v_0, v_1, \dots, v_n)$ 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けされている。 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 $\bigcirc$ 印で、記号節を $\square$ 印で表す。以後、状態節を $v, w, u \in \Gamma$ で表し、記号節を $x, y, z \in \Gamma$ 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。

$a_1 \dots a_n$: 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 $\langle v, p \rangle$ の集合。

$\langle v, p \rangle \in RE_SET$ 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 $\langle v, s \rangle$ 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。 $\langle v, s \rangle \in SH_SET$ 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 $a_1 \dots a_n$ 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($G, a_1 \dots a_{n+1}$)

begin

RESULT := FALSE;

s_0 でラベル付けされた節 v_0 で Γ を初期化する;

AC_SET を $\{v_0\}$ で初期化する;

for $i := 0$ to n do PARSEWORD(i);

return RESULT

end

PARSEWORD(i)

begin

RE_SET := $\emptyset$; SH_SET := $\emptyset$;

repeat

if (AC_SET $\neq \emptyset$) then ACTOR;

elseif (RE_SET $\neq \emptyset$) then REDUCER;

until (RE_SET = $\emptyset$ and AC_SET = $\emptyset$);

SHIFTER;

end

ACTOR

begin

AC_SETから要素 v を除去;

$\lambda := \text{ACTION}(\text{STATE}(v), a_{i+1});$

if ($\lambda = \text{"acc"}$) then RESULT:=TRUE;

if ($\lambda = \text{"sh j"}$) then

SH_SETに $\langle v, j \rangle$ を追加;

if ($\lambda = \text{"re p"}$) then

RE_SETに $\langle v, p \rangle$ を追加

end

REDUCER

begin

RE_SETから要素 $\langle v, p \rangle$ 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($\text{STATE}(w), A_p$), A_p でラベル

付けされた節 u, x を Γ に作成;

辺 $\langle u, x \rangle, \langle x, w \rangle$ を Γ に作成;

AC_SET:={ u }

end

SHIFTER

begin

SH_SETから要素 $\langle v, j \rangle$ 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 $\langle w, x \rangle, \langle x, v \rangle$ 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($\text{STATE}(v), a_{i+1}$)の内容に従う動作を実行する.

REDUCERは, RE_SETから要素 $\langle v, p \rangle$ 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 $\langle v, s \rangle$ 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.

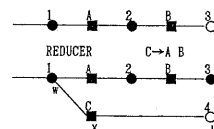


図1 REDUCERの説明図

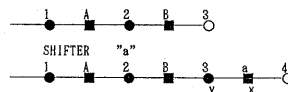


図2 SHIFTERの説明図

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.

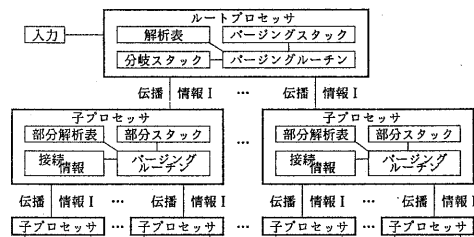


図3 システムの構成

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]

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

とすると、 $K' \times V_T$ 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。

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。

$k>0$: $d_s \sim d_s+k$ 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 $\langle q, d_q \rangle$

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。

- 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。
- 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。

- 既に存在する子プロセッサに伝播情報Iを通信する。
- 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。
- 還元動作を実行する。
- 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 $a_1 \dots a_n$ 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(I, X): パージングスタックP_STACKにX, Iをプッシュする手続き。

REDUCE(p): パージングスタックを $2 \times n_p$ ポップし、そのトップの要素をsとすると、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, $a_1 \dots a_{n+1}$)

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(a_i);

return RESULT

end

PARSEROOT(X)

```
begin
  ACTION( $s_j$ , 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(1, 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:= $d_a-2*n_p$ ;
  forever
    begin
      分岐スタックB_STACKをポップしその要素を<q,  $d_q$ >とする;
      if ( $k>d_q$ ) then
        begin
          <q,  $d_q$ >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
  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,  $d_a$ >を分岐スタックB_STACKにプッシュする
```

ユする

until(スタックW_STACKが空)

end

AMBIGU(ACTION_SET, X)

```
begin
  if ("sh l"  $\in$  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  $\neq$  1 or  $r \neq$  1) then
    NEW_BROAD(FLAG, ACTION_SET);
  if (FLAG=1) then SHIFT(1, X);
  else REDUCE( $p_1$ , 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,  $d_q$ >とする;
    TRANS( $q$ , (FLAG, ACTION_SET, X,  $d_a-d_q$ ,  $t(d_q, d_a)$ ));
    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,  $d_a$ >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_a>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_a-2*n_p_i, t(0, d_a-2*n_p_i)));

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={q₁, q₂, ..., 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, l): アルゴリズム1と同じ。

SEMI_REDUCE(p): 手続きREDUCE(p)からパー
ジスタックを2*n_pポップする処理を除い
た手続き。

PARSESON(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(I, X);

return RESULT;

end

PARSESON(I, X)

begin

ACTION(I, 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 l'}) then

exit;

elseif (ACTION_SET={"re p"}) then

REDUCE(p);

else AMBIGU(ACTION_SET, X);

end

PARSESONは、FLAGが-1でないならば、集合PRO_SE
T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、
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 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;

if (FLAG ≠ 1 or r ≠ 1) then

NEW_BROAD(FLAG, ACTION_SET);

if (FLAG ≠ 1) then REDUCE(p_i);

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 l'"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(FLAGS, 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(FLAGS, 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。

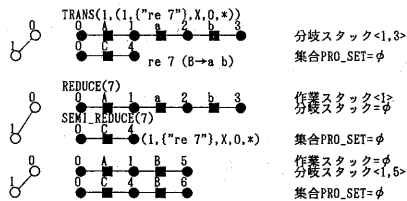


図4 ルートプロセッサにおいて還元動作が子プロセッサに影響する説明図

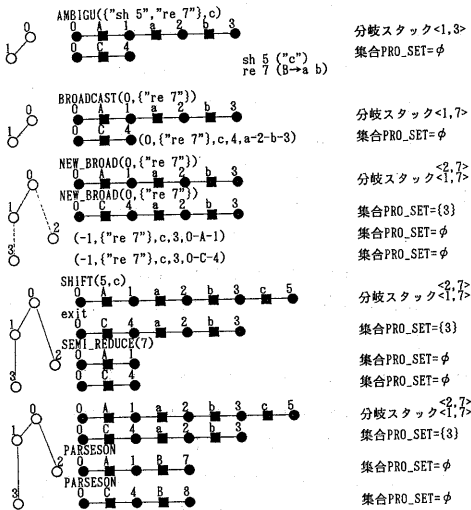


図5 ルートプロセッサにおいて曖昧状態になった例の説明図

〔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、プロセッサとパーキングスタックが1対1に対応しているの、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。

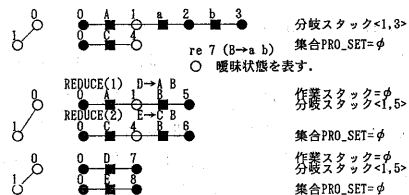


図6 意味処理実行のタイミング

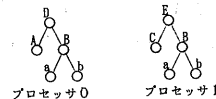


図7 解析結果の木構造

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パーシ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、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パーシ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.ACM 18, 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) DeMeyer, F.L.: "Practical Translations for LR(k) Languages": Ph.D.dissertation, M.I.T., Cambridge, Mass (1969).
- (9) DeMeyer, 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