TR - H - 164

# 単純順位文法に対する並列構文解析アルゴリズムのCM-5への実装とその性能評価

椎名　広光　　山川　栄樹
(豊橋技術科学大学)

1995. 9. 1

ATR人間情報通信研究所

〒619-02　京都府相楽郡精華町光台2-2　☎ 0774-95-1011

ATR Human Information Processing Research Laboratories
2-2, Hikaridai, Seika-cho, Soraku-gun, Kyoto 619-02 Japan
Telephone: +81-774-95-1011
Facsimile: +81-774-95-1008

# 単純順位文法に対する並列構文解析アルゴリズムの CM-5 への実装とその性能評価

椎名 広光

豊橋技術科学大学 大学院 システム情報工学専攻 増山研究室

山川 栄樹

株式会社 エイ・ティ・アール人間情報通信研究所 第六研究室

1995 年 9 月 1 日

## 要旨

構文解析は, コンパイラや自然言語処理における処理の 1 フェーズであり, 入力文字があらかじめ与えられた文法の生成する言語のクラスに属しているか否かを調べる認識処理と, 入力文字列が文法に対してもつ構造を調べる構文解析木作成処理とからなる.

本稿ではまず, 1 個の制御プロセッサを含む複数個のプロセッサと共有メモリとで構成される理論的な並列計算機のモデルにおいて, 共有メモリに対する同時書き込みが不可能である場合を考え, コンパイラで使われる文法の一つである単純順位文法を並列的に構文解析するアルゴリズムを構築する. そして, 入力文字数を $n$ とするとき, $O(n^2)$ 個のプロセッサを使用することにより, 提案した並列構文解析アルゴリズムは $O(\log^2 n)$ 時間で単純順位文法の構文解析を終了し得ることを示す.

さらに, 理論的な並列計算機を前提として考案された並列アルゴリズムを, 現実の並列計算機コネクションマシン CM-5 上に実装し, その処理時間を計測することによって実際的な性能の評価を行う.

# 1 はじめに

構文解析は, コンパイラや自然言語処理における処理の 1 フェーズであり, 入力文字があらかじめ与えられた文法の生成する言語のクラスに属しているか否かを調べる認識処理と, 入力文字列が文法に対してもつ木構造などの構造を調べる構文解析木作成処理とからなる.

コンパイラにおいては, 予約語や変数名を単位とするプログラムを読込み, プログラム全体が表す 1 つの構文解析木を作成する. 但し, コンパイラが実際に作成した構文解析木を出力することはない. これらは内部情報として記録あるいは保持され, プログラムエラーを検索したり, オブジェクトコードを出力する際のデータとして利用される. 本稿では, 入力の単位を終端記号とよび, 終端記号を並べた文字列を入力文字列として構文解析を行う. 一方, 文法における生成規則を構成する記号は, 非終端記号とよぶことにする.

ここで, 構文解析処理によって作成される構文解析木の一例を示しておこう. たとえば文法の生成規則として $\{S \to AB, A \to AA, A \to a, B \to b\}$ が定義され, 入力文字列 $aaaab$ が与えられた場合, 図 1 に示すような構文解析木が作成される. 一般に, ある入力に対して作成し得る構文解析木は一通りとは限らない. 図 1(a) に示されるように, 入力文字列の先頭に近い方から順次生成規則を適用することを最左導出とよび, 最左導出によって得られる構文解析木を最左導出木とよぶ. これに対して, 図 1(b) のように入力文字列の末尾に近い方から生成規則を適用することを最右導出とよび, 最右導出を行った結果得られる構文解析木を最右導出木とよぶ.

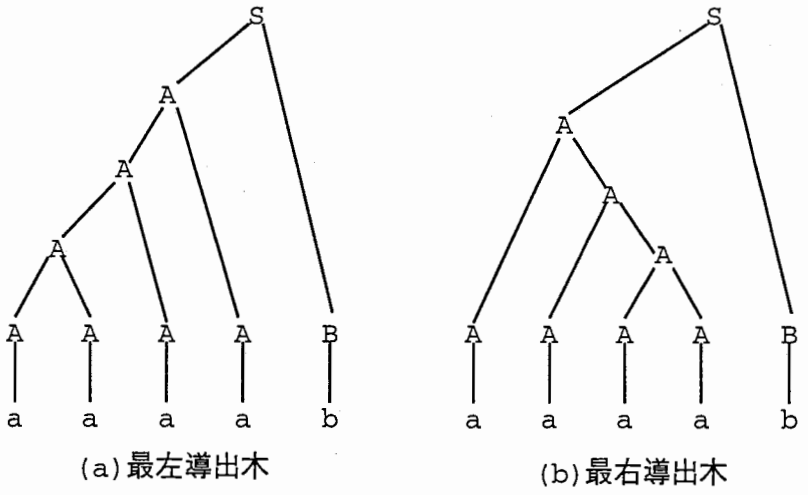


図 1: 構文解析木の例

逐次処理を用いて構文解析木を作成する方法については, 既に多くの研究が存在する. とくに, $A \to BCD$, $A \to a$ のように, 生成規則を表す $\to$ の左側が必ず 1 つの文法記号からなる文脈自由文法とよばれるクラスのなかには, 入力文字数 $n$ に比例する時間で構文解析を行うことができる特別な文法の存在が知られている. たとえば, LR 文法, LL 文法, 単純順位文法はその代表的な例であり [1] , それらは図 2 に示すような包含関係をもっている. なお, ほとんどのプログラミング言語は LR 文法または LL 文法を用いて表現することができるため, コンパイラの内部では LR 構文解析または LL 構文解析を行う場合が多い.

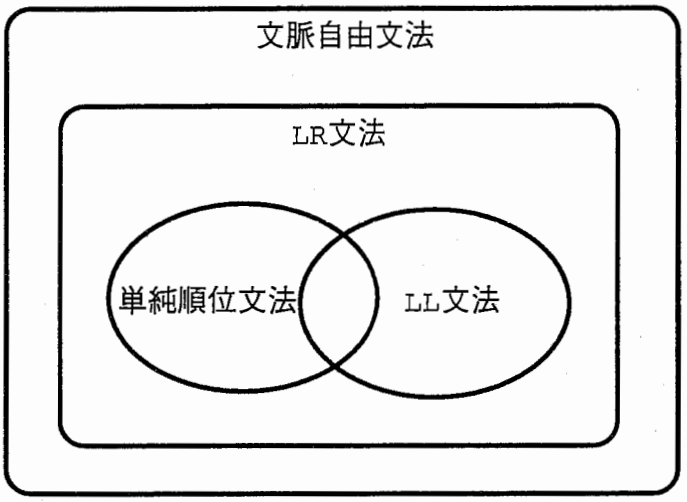


図 2: 文法の包含関係

逐次処理に基づく構文解析の研究は, 理論と実用の両面において幅広く行われているが, 並列処理を用いた構文解析については, 理論または実用のいずれか一方に焦点を絞った研究がほとんどである. 並列構文解析アルゴリズムを理論的に扱う場合, 並列計算機のモデルとして図 3 に示されるような P-RAM (Parallel Random Access Machine) を用いることが多い. P-RAM は, 1 個の制御プロセッサと複数個の並列プロセッサおよび共有メモリから構成される並列計算機で, 共有メモリからの同時読出しは可能であるものの, 共有メモリへの同時書込みができない構造をもつものを, 特に CREW P-RAM (Concurrent Read Exclusive Write P-RAM) とよぶ. 一般に, 入力文字数 $n$ の多項式個のプロセッサを用いて, $\log n$ の多項式時間で処理が終了するならば, 理論的に効率がよい並列構文解析アルゴリズムであると見なされる. 実際, 一般の文脈自由文法 [1, 3, 4, 5] に対して, $O(n^6)$ 個のプロセッサを用いて $O(\log^2 n)$ 時間で構文解析を実行する並列アルゴリズム [2] や, 左右が完全に対応した括弧のみを入力文字とするブラケット言語 (Blacket Language) およびインプット・ドリブン言語 (Input-Driven Language) [2] に対して, $n/\log n$ 個のプロセッサを用いて $O(\log n)$ 時

間で構文解析を実行する並列アルゴリズム [2] が存在する．しかし，理論的に効率がよいとされるこれらの並列アルゴリズムにおいても，一般の文脈自由文法に対する場合のように必要なプロセッサの数が現実的ではなかったり，ブラケット言語やインプット・ドリブン言語のように生成できる文法の能力が小さ過ぎたりするなどの問題点が存在する．

一方，文脈自由文法に対する構文解析のプロセスを途中から分割し，これらを並列的に処理するアルゴリズムの研究も行われている [7]．しかしながら，処理に要する時間が $O(n)$ となるため，理論的に効率のよい並列アルゴリズムとはいえない．

本稿では，文脈自由文法の部分クラスであり，コンパイラ内部の構文解析にも利用される単純順位文法を対象として，$O(n^2)$ 個のプロセッサを用いて $O(\log^2 n)$ 時間で構文解析を実行する理論的に効率のよい並列アルゴリズムを提案する．このアルゴリズムは，CREW P-RAM 上での実現を想定しているが，現実の並列計算機のほとんどは，並列プロセッサにもたせた局所メモリを相互結合網で結びつける方式をとっている．そこで，提案したアルゴリズムを実際の並列計算機 CM-5 へ実装し，分散メモリ型の並列計算機を用いた場合においても，逐次処理による場合に比べて解析に要する時間の面で十分に有効であることを示す．

なお，以下では文法の生成規則の集合を $P$，生成規則における開始記号を $S$，生成規則に現れる $S$ 以外の文法記号である非終端記号の集合を $N$，入力文字列の単位である終端記号の集合を $T$，また，文字列の開始と終了を表す記号を $\$ \notin T$ と書く．そして，単純順位文法を $G = (N, T, P, S, \$)$ で表し，入力文字列 $a_0\, a_1 \ldots a_i \ldots a_n\, a_{n+1}$ はあらかじめ与えられているものとする．ただし，$a_i \in T$, $i = 1, \ldots, n$, $a_0 = a_{n+1} = \$$ である．

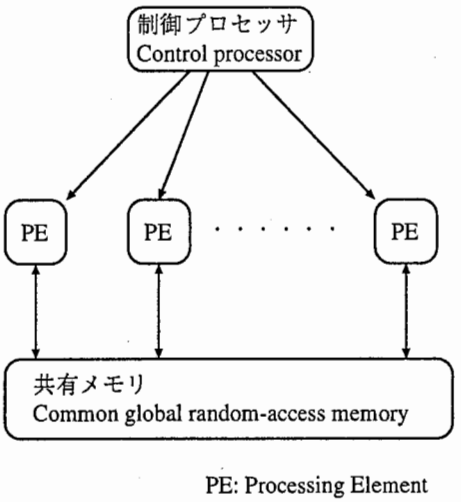


図 3: **P-PRAM の構造**

## 2 単純順位文法の定義

単純順位文法は, 構文解析を進める際に, 適用すべき生成規則を文法記号 $(N \cup T)$ 間の順位関係 (precedence relation) のみで一意的に決定できる文法である. ここで, 順位関係とは構文解析木上の任意の二つの文法記号 $X, Y$ の間に定義される関係で, $X \lessdot Y$, $X \gtrdot Y$, および, $X \doteq Y$ の 3 種類がある. $X \lessdot Y$ のとき, $X$ は $Y$ より順位が低いとよばれ, 文法記号 $Y$ の接続する枝が, 現在処理した $X$ よりも右側に伸びることを意味している. 逆に $X \gtrdot Y$ のとき, $X$ は $Y$ より順位が高いとよばれ, 現在処理した文法記号 $X$ の接続する枝が, $Y$ よりも左側に伸びていることを意味している. 一方 $X \doteq Y$ のとき, $X$ と $Y$ は順位が等しいとよばれ, $XY$ を部分にもつ生成規則が存在することを意味している. 簡単な例を図 4 に示す. なお, $X$ と $Y$ が文法上で並ぶことがないならば, これらの間に順位関係は存在しない.

以下では, $\varepsilon$ で空語すなわち長さ 0 の記号列を, $(N \cup T)^+$ で 1 個以上の有限長の記号列を表し, $(N \cup T)^* = \{\varepsilon\} \cup (N \cup T)^+$ とする. また, $\xi \in (N \cup T)^+$, $\zeta \in (N \cup T)^+$ とするとき, 記号列 $\zeta$ が $\xi$ に生成規則を 1 回, 1 回以上, および, 0 回以上適用することによって導出されることを, それぞれ $\xi \Longrightarrow \zeta$, $\xi \overset{+}{\Longrightarrow} \zeta$, および, $\xi \overset{*}{\Longrightarrow} \zeta$ と書く. 特に, 最右導出の場合は, それぞれ $\xi \underset{\mathrm{rm}}{\Longrightarrow} \zeta$, $\xi \overset{+}{\underset{\mathrm{rm}}{\Longrightarrow}} \zeta$, および, $\xi \overset{*}{\underset{\mathrm{rm}}{\Longrightarrow}} \zeta$ と記述する.

いま, 最右導出 $S \overset{*}{\underset{\mathrm{rm}}{\Longrightarrow}} \mu A v \underset{\mathrm{rm}}{\Longrightarrow} \mu \alpha v$ を考える. 但し, $A \in N$, $v \in (N \cup T)^+$ で, $X_i \in (N \cup T)$, $i = 1, \ldots, q$, に対して

$$\begin{aligned} \mu &= X_q X_{q-1} \cdots X_{k+1}, \qquad \text{または} \qquad \mu = \varepsilon, \\ \alpha &= X_k X_{k-1} \cdots X_1 \end{aligned}$$

である. また, $head_1(v)$ で, $v$ の中で最初に現れる終端記号を表す. さらに, 導出された記号列 $\mu \alpha v$ の各隣接記号の間に, 次のような順位関係があるものとする.

$$\begin{cases} X_{i+1} \text{ と } X_i \, (i > k) & \text{の間: } X_{i+1} \doteq X_i \quad \text{または} \quad X_{i+1} \lessdot X_i, \\ X_{k+1} \text{ と } X_k & \text{の間: } X_{k+1} \lessdot X_k, \\ X_i \text{ と } X_{i-1} \, (1 < i \leq k) & \text{の間: } X_i \doteq X_{i-1}, \\ X_1 \text{ と } head_1(v) & \text{の間: } X_1 \gtrdot head_1(v). \end{cases}$$

このとき, 入力文字列を左から右へたどりながら最右導出の構文解析木を求めるようなボトムアップ方式の構文解析が実行可能である. また, 記号間の順位関係に注目するとき, 一つの $\lessdot$, 幾つかの $\doteq$, 一つの $\gtrdot$ がこの順に左から並んでいる部分が見つかると, その部分 (上の例では $\alpha$) は最右導出によって最後に得られた部分であることがわかり, 適用すべき生成規則 (同じく $A \to \alpha$) を一意的に決定することができる.

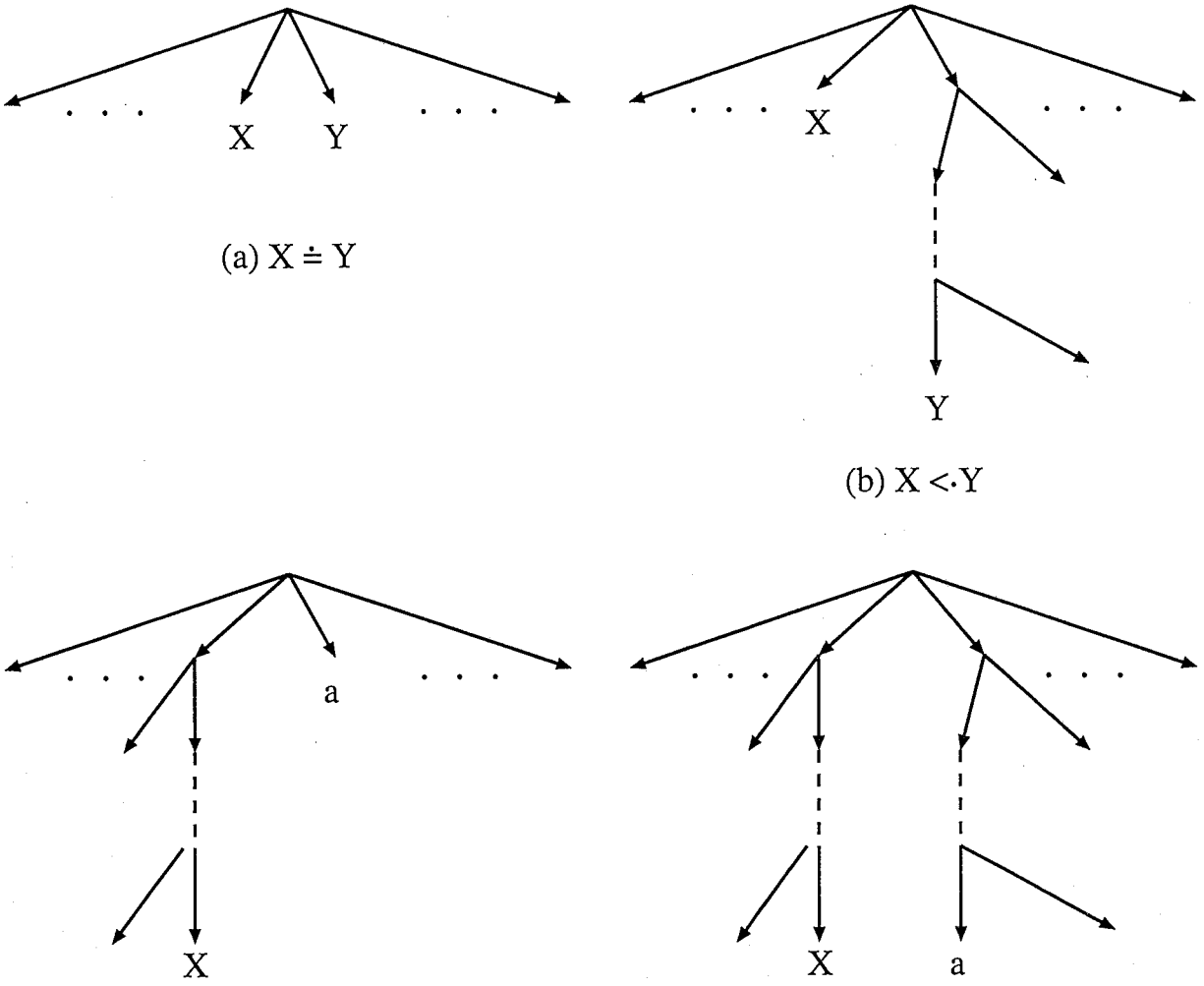


図 4: 構文解析木の導出と順位関係

順位関係 $\lessdot$ , $\doteq$, $\gtrdot$ を用いると, 単純順位文法は次のように定義される.

**定義** [5] 文法 $G = (N, T, P, S, \$)$ が次の条件を満たすとき, これを単純順位文法という. 但し, $A \in N$, $B \in N$, $X \in (N \cup T)$, $Y \in (N \cup T)$, $Z_i \in (N \cup T)$, $i = 1, \ldots, n$, $\alpha \in (N \cup T)^*$ である.

(1) $\$ \lessdot S, S \gtrdot \$$ である.

(2) $S \overset{+}{\Longrightarrow} X\alpha$ ならば, $\$ \lessdot X$.

(3) $S \overset{+}{\Longrightarrow} \alpha X$ ならば, $X \gtrdot \$$.

(4) $(A \to \varepsilon) \notin P$ (空語規則は存在しない).

(5) $A \overset{+}{\Longrightarrow} A$ なる導出は起こらない (サイクルをもたない).

(6) $X \lessdot Y, X \gtrdot Y, X \doteq Y$ の高々一つしか成り立たない.

(7) $A \to X_1 X_2 \ldots X_n$ , $B \to X_1 X_2 \ldots X_n$ ならば $A = B$ (右辺が一致し, 左辺の異なる生成規則は存在しない).

条件 (6) は **順位関係の一致性**, 条件 (7) は **逆方向決定性**とよばれる.

## 3 単純順位文法の例

ここでは, 単純順位文法の例と, それに対する順位表を示す. まず次のような単純順位文法

$$G_1 = (\{A\}, \{a, b\}, \{S \to A, A \to Aab, A \to ab\}, S, \$)$$

を考える. 文法 $G_1$ に対する順位表を, 表 1 に示す.

表 1 : 文法 $G_1$ に対する順位表

| | S | A | a | b | $ |
|---|---|---|---|---|---|
| $ | $\lessdot$ | $\lessdot$ | $\lessdot$ | | |
| S | | | | | $\gtrdot$ |
| A | | | $\doteq$ | | $\gtrdot$ |
| a | | | | $\doteq$ | |
| b | | | $\gtrdot$ | | $\gtrdot$ |

縦軸の文法記号 { $\lessdot$ , $\doteq$, $\gtrdot$ } 横軸の文法記号.

文法の生成規則を表す $\rightarrow$ の右側を, 一つの終端記号または二つの非終端記号で表した形式を, Chomsky 標準形 [4] という. 文法 $G_1$ を Chomsky 標準形に変換すると,

$$G_2 = (\{A, B, C\}, \{a, b\}, \{S \rightarrow AB, S \rightarrow CB, C \rightarrow SA, A \rightarrow a, B \rightarrow b\}, S, \$)$$

となる. なお, Chomsky 標準形に変換しても, 文法の生成能力は変化しない. すなわち, $G_2$ も単純順位文法で, $G_1$ と同一の言語を生成する. 文法 $G_2$ に対する順位表を表 2 に示す.

表 2 : 文法 $G_2$ に対する順位表

| | S | A | B | C | a | b | $ |
|---|---|---|---|---|---|---|---|
| $ | $\lessdot$ | $\lessdot$ | | $\lessdot$ | $\lessdot$ | | |
| S | | $\doteq$ | | | $\lessdot$ | | $\gtrdot$ |
| A | | | $\doteq$ | | | $\lessdot$ | |
| B | | | | | $\gtrdot$ | | $\gtrdot$ |
| C | | | $\doteq$ | | | $\lessdot$ | |
| a | | | | | | $\gtrdot$ | |
| b | | | | | $\gtrdot$ | | $\gtrdot$ |

縦軸の文法記号 { $\lessdot$ , $\doteq$, $\gtrdot$ } 横軸の文法記号.

## 4 単純順位文法に対する逐次型の構文解析

単純順位文法の生成規則の形式を Chomsky 標準形に制限すると, 逐次処理に基づく構文解析は, 一つの終端記号を生成する規則を適用する場合と, 二つの非終端記号を生成する規則を適用する場合とに分けられる. いま, 構文解析の状況を

(スタックの内容, 残りの入力, スタックの最上段の内容と次の入力との順位関係)

で表し, 1 ステップの状況変化を $\vdash$ と書くことにする. ただし, スタックには, 既に入力された文字列に対する構文解析結果を示す文法記号の列が積み上げられれているものとする. このとき, 各場合における状況の変化はそれぞれ次のようになる.

(1) 生成規則 $A_i \rightarrow a_i\,(A_i \in N,\, a_i \in T)$ を適用する場合.

まず, 生成規則を Chomsky 標準形に制限しているため, 新たな終端記号 $a_i$ が入力される直前のスタックの最上段の文法記号は, 必ず非終端記号となっていることに注意しよう. こ

れを $A_{i-1} \in N$ として, 新たな入力文字 $a_i$ との順位関係を比較すると, 最右導出において, 既に導出された記号列の最後の文法記号と新たに導出された記号列の最初の文法記号との関係から, $A_{i-1} \lessdot a_i$ であることがわかる. そこで, $a_i$ をスタックの最上段に移して, 次の入力文字 $a_{i+1}$ との順位関係を比較する. 最右導出において, 新たに導出された記号列の最後の文法記号とまだ文法規則を適用されていない記号列の最初の終端記号との関係から, $a_i \gtrdot a_{i+1}$ となることがわかる. そこで, スタックの最上段に積まれた終端記号 $a_i$ を, 生成規則 $A_i \to a_i$ に従って $A_i$ に置き換える. 状況の変化を式で記述すると, 次のようになる.

$$
\begin{aligned}
& (\,\$ \ldots A_{i-1},\ a_i\, a_{i+1} \ldots a_n\, \$,\ \lessdot\,) \\
\vdash\ & (\,\$ \ldots A_{i-1}\, a_i,\ a_{i+1} \ldots a_n\, \$,\ \gtrdot\,) \\
\vdash\ & (\,\$ \ldots A_{i-1}\, A_i,\ a_{i+1} \ldots a_n\, \$,\ ?\,).
\end{aligned}
$$

但し, $A_i$ と $a_{i+1}$ の間の順位関係は議論しないため, 記号 ? を用いている.

(2) 生成規則 $X \to YZ\,(X, Y, Z \in N)$ を適用する場合.

スタックの最上段に置かれている非終端記号を $Y$, 新たな入力文字を $a_{j+1}$ とすると, これらの順位関係は $Y \lessdot a_{j+1}$ であることがわかる. その後, 入力文字の部分列 $a_{j+1} \ldots a_k$ の処理が進み, スタックに積まれた上位 2 つの非終端記号が $Y$ と $Z$ になったと仮定する. このとき, $Z$ と $a_{k+1}$ との間には, 順位関係 $Z \lessdot a_{k+1}$ または $Z \gtrdot a_{k+1}$ が成立つ. なお, 生成規則を Chomsky 標準形に限定しているため, 順位関係が $Z \doteq a_{k+1}$ となることはない.

単純順位文法は最右導出の構文解析を行なうため, 順位関係 $Z \gtrdot a_{k+1}$ 成立っている場合には, スタックの最上段の $Z$ とその下の $Y$ との間に順位関係 $\doteq$ が成立つ. 従って, 生成規則 $X \to YZ$ を適用して, $YZ$ を $X$ で置換える. 式で記述すると, 次のようになる.

$$
\begin{aligned}
& (\,\$ \ldots\ Y,\ \ a_{j+1}\, a_{j+2} \ldots a_k\, a_{k+1}\, a_{k+2} \ldots a_n\, \$,\ \lessdot\,) \\
\vdash\ & \ldots \\
\vdash\ & (\,\$ \ldots YZ,\ a_{k+1}\, a_{k+2} \ldots a_n\, \$,\ \gtrdot\,) \\
\vdash\ & (\,\$ \ldots\ X,\ \ a_{k+1}\, a_{k+2} \ldots a_n\, \$,\ ?\,).
\end{aligned}
$$

但し, $X$ と $a_{k+1}$ の間の順位関係は議論しないため, 記号 ? を用いている.

一方, 順位関係 $Z \lessdot a_{k+1}$ が成立っている場合は, 先に $a_{k+1}$ を次の入力として受付ける.

$$
\begin{aligned}
& (\,\$ \ldots\ Y,\ \ a_{j+1}\, a_{j+2} \ldots a_k\, a_{k+1}\, a_{k+2} \ldots a_n\, \$,\ \lessdot\,) \\
\vdash\ & \ldots \\
\vdash\ & (\,\$ \ldots YZ,\ a_{k+1}\, a_{k+2} \ldots a_n\, \$,\ \gtrdot\,) \\
\vdash\ & (\,\$ \ldots YZ\, a_{k+1},\ a_{k+2} \ldots a_n\, \$,\ ?\,).
\end{aligned}
$$

但し, $a_{k+1}$ と $a_{k+2}$ の間の順位関係は議論しないため, 記号 ? を用いている.

具体的な例として, 第 3 章で示した単純順位文法 $G_2$ を取上げ, 入力文字列が $abab$ である場合の構文解析の動作状況を以下に示す.

$$
\begin{aligned}
&(\$, abab\$, \lessdot) \vdash (\$a, bab\$, \gtrdot) \\
\vdash\ &(\$A, bab\$, \lessdot) \vdash (\$Ab, ab\$, \gtrdot) \\
\vdash\ &(\$AB, ab\$, \gtrdot) \vdash (\$S, ab\$, \lessdot) \\
\vdash\ &(\$Sa, b\$, \gtrdot) \vdash (\$SA, b\$, \gtrdot) \\
\vdash\ &(\$C, b\$, \lessdot) \vdash (\$Cb, \$, \gtrdot) \\
\vdash\ &(\$CB, \$, \gtrdot) \vdash (\$S, \$, \gtrdot).
\end{aligned}
$$

また, 作成される構文解析木は, 図 5 のようになる.

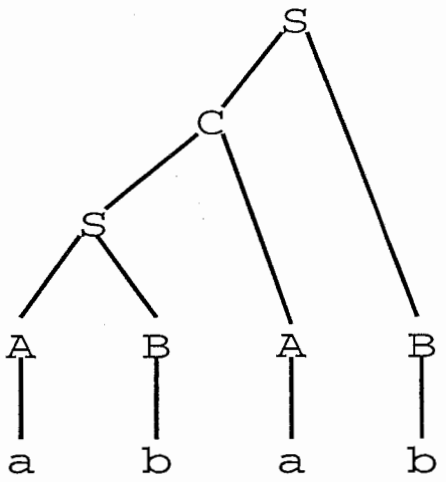


図 5: 構文解析木の例

## 5 単純順位文法に対する並列構文解析アルゴリズム

ここでは, 単純順位文法に対する並列構文解析アルゴリズムを提案する. 構文解析を行うことによって, 節点集合を $\mathcal{V}$, 枝集合を $\mathcal{E}$ とする構文解析木の候補 $\mathcal{T} = (\mathcal{V}, \mathcal{E})$ が 作成されるものとする. 提案するアルゴリズムは, 枝集合 $\mathcal{E}$ に枝の候補を順次添加しながら構文解析を進めるペブルゲーム法 [2] を拡張したものである.

以下では, 木 $\mathcal{T}$ の節点 $v$ にラベル pebble を付すことを, 「節点 $v$ を pebbled する」と言う. 通常のペブルゲーム法では, あらかじめ与えられた葉の数が $n$ の二分木に対して,

$O(n)$ 個のプロセッサを用いて次のような手続きを実行する．まず始めに，二分木のすべての葉を pebbled する．そして，以下に示す手続き activate, square, pebble を繰返す．このとき，$O(\log n)$ 時間で木 $\mathcal{T}$ のすべての節点が pebbled される．

(1) 手続き activate : ある節点 $v_1$ に対して，一方の子供の節点 $v_2$ が pebbled されているならば，$v_1$ のポインタが指す節点 $cond(v_1)$ は pebbled されていない子供 $v_3$ とする．また，節点 $v_2$, $v_3$ がいずれも pebbled されていれば，$cond(v_1)$ は $v_2$ と $v_3$ のいずれか一方とする．

(2) 手続き square : ある節点 $v$ のポインタが指す節点から更にポインタによって指示される節点が存在するならば，節点 $v$ のポインタをつけかえる．すなわち，$cond(v) := cond(cond(v))$ とする．

(3) 手続き pebble : ある節点 $v$ のポインタが指す節点 $cond(v)$ が pebbled されていれば，節点 $v$ も pebbled する．

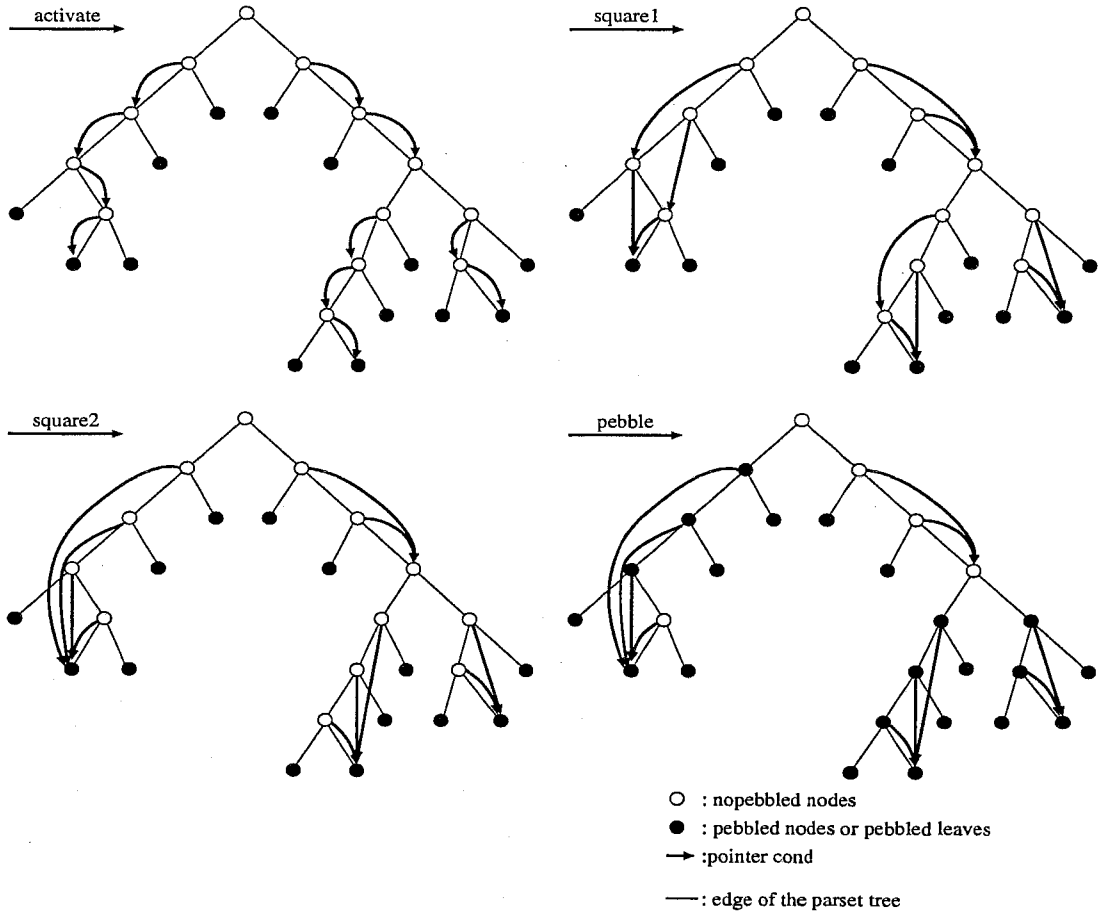


図 6: ペブルゲームの実行

但し, 手続き square は, 1 反復あたり 2 回実行する. ペブルゲーム法の 1 反復における実行の様子を, 図 6 に示す.

通常のペブルゲーム法は, 予め存在している二分木の上で実行されるが, 本稿で提案するアルゴリズムにおいては, 二分木の枝を作成しながらペブルゲーム法を実行する. また, pebbled した節点は, 構文解析が終了した部分を示す指標として用いられる. なお, 本稿で対象とする単純順位文法の生成規則は, Chomsky 標準形で表現されているものとし, 入力文字列長 $n$, 非終端記号の種類の数 $|N|$ は既知であるとする.

アルゴリズムで使用するデータは, 次の通りである.

(1) 構文解析表 $PT$ : $n \times n \times |N|$ の 3 次元配列である. 各要素 $PT[i,j,X]$ の値は, 入力文字列の $i \sim j$ 番目が構文解析できて非終端記号 $X$ が得られるときに値 pebbled を取り, 構文解析できないときに値 unpebbled を取る.

$$PT[i,j,X] = \begin{cases} \text{pebbled}, & X \overset{*}{\underset{\text{rm}}{\Longrightarrow}} a_i \ldots a_j \text{ となるとき}, \\ \text{unpebbled}, & \text{それ以外のとき}. \end{cases}$$

(2) 構文解析木の候補 $\mathcal{T} = (\mathcal{V}, \mathcal{E})$ : 節点集合 $\mathcal{V}$ は $\{1,\ldots,n\} \times \{1,\ldots,n\} \times N$ の部分集合, 枝集合 $\mathcal{E}$ は $\mathcal{V} \times \mathcal{V}$ の部分集合である.

(3) 表 $Right$ : 各要素 $Right[i,Y]$ の値は, 入力文字列の $i$ 番目の文字 $a_i$ と, ある非終端記号 $Y \in N$ に対して, 次の式で定められる.

$$Right[i,Y] = \min_{Z \in N} \{ \max_j \{ j \mid PT[i,j,Z] = \text{pebbled}, \ {}^{\exists}(X \to YZ) \in P \} \}.$$

(4) 表 $Right\_NT$ : $Right[i,Y]$ の値を求めるときに探索した非終端記号を格納する表である. 各要素 $Right\_NT[i,Y]$ の値は, 入力文字列の $i$ 番目の文字 $a_i$ と, ある非終端記号 $Y \in N$ に対して, 次の式で定められる.

$$Right\_NT[i,Y] = \arg\min_{Z \in N} \{ \max_j \{ j \mid PT[i,j,Z] = \text{pebbled}, \ {}^{\exists}(X \to YZ) \in P \} \}.$$

(5) 表 $Low$ : 各要素 $Low[j,Z]$ の値は, 入力文字列の $j$ 番目の文字 $a_j$ と, ある非終端記号 $Z \in N$ に対して, 次の式で定められる.

$$Low[j,Z] = \max_{Y \in N} \{ \min_i \{ i \mid PT[i,j,Y] = \text{pebbled}, \ {}^{\exists}(X \to YZ) \in P \} \}.$$

(6) 表 $Low\_NT$ : $Low[j,Z]$ の値を求めるときに探索した非終端記号を格納する表である. 各要素 $Low\_NT[j,Z]$ の値は, 入力文字列の $j$ 番目の文字 $a_j$ と, ある非終端記号 $Z \in N$ に対して, 次の式で定められる.

$$Low\_NT[j,Z] = \arg\max_{Y \in N} \{ \min_i \{ i \mid PT[i,j,Y] = \text{pebbled}, \ ^{\exists}(X \to YZ) \in P \} \}.$$

なお, 表 $Right$, $Right\_NT$, $Low$, $Low\_NT$ の定義において, $\max_j$ および $\min_i$ を求める際に条件を満たす $i$, $j$ が存在しない場合には, それぞれ

$$Right[i,Y] = i, \quad Right\_NT[i,Y] = Y, \quad Low[j,Z] = j, \quad Low\_NT[j,Z] = Z$$

とする. また, ペブルゲーム法の説明のところでも既に触れたように, 構文解析木の節点間のポインタを $cond$ で表す. 上で述べたデータ構造に対応して言い換えると, $cond$ は $\{1,\ldots,n\} \times \{1,\ldots,n\} \times N$ の要素から, $\{1,\ldots,n\} \times \{1,\ldots,n\} \times N$ の要素への関数となる. また, $(i,j,X) \in \{1,\ldots,n\} \times \{1,\ldots,n\} \times N$ に対して, 関数 $F_1$, $F_2$, $F_3$ をそれぞれ次のような値を返す関数として定義する.

$$F_1((i,j,X)) = i, \qquad F_2((i,j,X)) = j, \qquad F_3((i,j,X)) = X.$$

このとき, 並列構文解析の手続きは次のように記述される.

**Procedure Precedence_Parallel_Parser**

**ステップ 1: (初期化)** 構文解析表 $PT$ の各要素に, 値 unpebbled をセットする. また, 構文解析木の候補 $\mathcal{T}$ の節点集合 $\mathcal{V}$ および枝集合 $\mathcal{E}$ をそれぞれ次のように定める.

$$\mathcal{V} := \{1 \ldots n\} \times \{1 \ldots n\} \times N, \qquad E := \emptyset.$$

**ステップ 2: (構文解析木の葉の計算)** 入力文字列の各文字 $a_i$, $i = 1 \ldots n$,に対して, 次の処理を順次行う. まず, $P$ に $A_i \to a_i$ なる生成規則が存在するならば,

$$PT[i,i,A_i] := \text{pebbled}$$

とする. そのうえで, $X \to A_i\, A_{i+1}$ なる生成規則が存在するならば,

$$Right[i,A_i] := i+1, \qquad Right\_NT[i,A_i] := A_{i+1}$$

とおく. さらに, $X \to A_{i-1}\, A_i$ なる生成規則が存在するならば,

$$Low[i,A_i] := i-1, \qquad Low\_NT[i,A_i] := A_{i-1}$$

とセットする.

ステップ 3: (activate 1) $(i,j,Y)$, $1 \leq i \leq n-1$, $i \leq j \leq n-1$, $Y \in N$, に対して,

$$Z = Right\_NT[j, Y]$$

とおくとき,

条件 (1) $Y \lessdot a_{j+1}$,
条件 (2) $Z \gtrdot a_{Right[j,Y]+1}$,
条件 (3) $Y \doteq Z$,
条件 (4) $^{\exists}(X \to YZ) \in P$,

が成立っているならば,

$$cond\,((i,j,Y)) := (i, Right[j, Y], X)$$

および

$$\begin{aligned} E \;:=\; E \;&\cup\; \{(i,j,Y), (i, Right[j,Y], X)\} \\ &\cup\; \{(j+1, Right[j,Y], Z), (i, Right[j,Y], X)\} \end{aligned}$$

とする.

ステップ 4: (activate 2) $(i,j,Z)$, $2 \leq i \leq n$, $i \leq j \leq n$, $Z \in N$, に対して,

$$Y = Low\_NT[i, Z]$$

とおくとき,

条件 (5) $Z \lessdot a_{i-1}$,
条件 (6) $Y \gtrdot a_{Low[i,Z]+1}$,
条件 (7) $Y \doteq Z$,
条件 (8) $^{\exists}(X \to YZ) \in P$,

が成立っているならば,

$$cond\,((i,j,Z)) := (Low[i, Z], j, X)$$

および

$$\begin{aligned} E \;:=\; E \;&\cup\; \{(i,j,Z), (Low[i,Z], j, X)\} \\ &\cup\; \{(Low[i,Z], i-1, Y), (Low[i,Z], j, X)\} \end{aligned}$$

とする.

ステップ 5: (square 1) $(i,j,X)$, $1 \leq i \leq n$, $i \leq j \leq n$, $X \in N$, に対して,

$$cond\,((i,j,X)) := cond\,(cond\,(i,j,X))$$

とする.

ステップ 6: (square 2) $(i,j,X)$, $1 \leq i \leq n$, $i \leq j \leq n$, $X \in N$, に対して,

$$cond\,((i,j,X)) := cond\,(cond\,(i,j,X))$$

とする (ステップ 5 の繰返し).

ステップ 7: (pebble) $(i,j,X)$, $1 \leq i \leq n$, $i \leq j \leq n$, $X \in N$, に対して, 条件

$$PT[\,i,j,X\,] = \text{pebbled}$$

が成立っているならば

$$PT[\,F_1(cond((i,j,X))),\, F_2(cond((i,j,X))),\, F_3(cond((i,j,X)))\,] := \text{pebbled}$$

とする.

ステップ 8: (Right and Low) $i = 1 \ldots n$ と非終端記号 $X \in N$ のすべての組合せに対して, $Right[\,i,X\,]$, $Right\_NT[\,i,X\,]$, $Low[\,i,X\,]$, $Low\_NT[\,i,X\,]$ の各値を計算する.

ステップ 9: (終了判定) もし

$$PT[\,1,n,S\,] = \text{pebbled}$$

ならば, ステップ 10 に進む (入力文字列を受理). さもなければ, ステップ 3 へ戻る.

ステップ 10: (構文解析木の再構成) 最初の入力文字 $a_1$ に対して, $(A_1 \to a_1) \in P$ なる非終端記号を $A_1$ とする. 木 $\mathcal{T} = (\mathcal{V}, \mathcal{E})$ の節点 $(1,1,A_1) \in \mathcal{V}$ を根としてオイラーツアー法 (Euler tour technique) [2] を適用し, $\mathcal{T}$ の各節点に対して深さ優先順の順位を並列的に求める. さらに, 順位のついた節点のみを取出して, それらによって張られる $\mathcal{T}$ の部分木を, 改めて入力文字列の構文解析木とする. □

容易に確かめられるように, ステップ 1 および 3 ~ 7 は構文解析表 $PT$ の各要素 $(i,j,X)$, $X \in N$, に対して並列的に処理を実行することができる. また, ステップ 2 は入力文字列のすべての文字 $a_i$, $i = 1 \ldots n$, に対して, ステップ 8 は入力文字列の各文字 $a_i$, $i = 1 \ldots n$, と非終端記号 $X \in N$ のすべての組合せに対して, 並列的に処理を行うことができる.

## 6 単純順位文法に対する並列構文解析の例

第 3 章で取上げた Chomsky 標準形の単純順位文法 $G_2$ を例に, 並列構文解析の手続き Procedure Precedence_Parallel_Parser のステップ 2 ～ 7 が実行される様子を図 7 に示す. 但し, 入力文字列は *abab* であり, ステップ 3 ～ 8 のループを 1 回行うだけで構文解析が終了する.

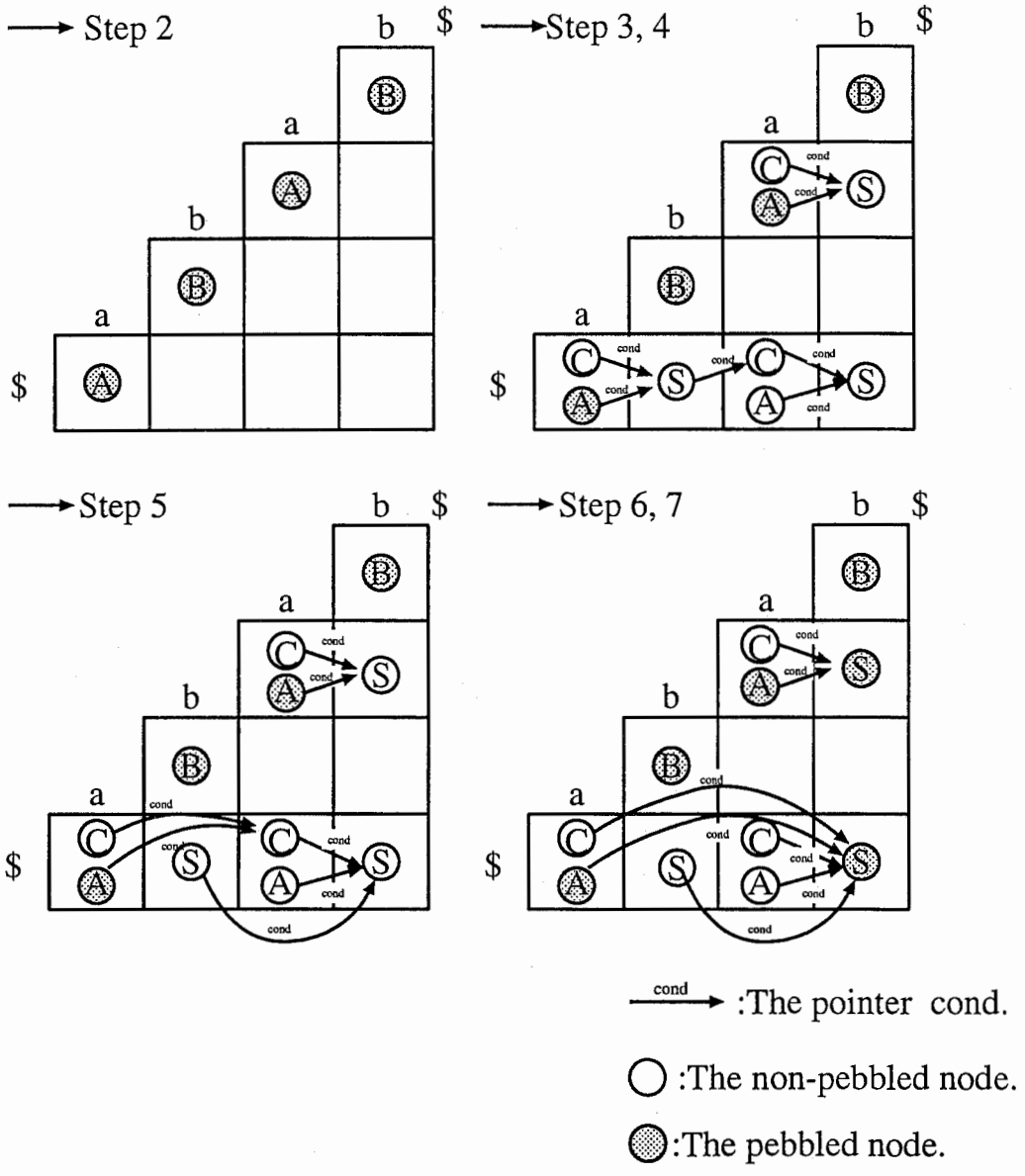


図 7: 並列構文解析の様子

# 7 並列構文解析アルゴリズムの正当性

ここでは, 第 5 章で提案した単純順位文法に対する並列構文解析アルゴリズムの正当性を証明する. まず第 7.1 節において, 並列構文解析の手続きで使用した構文解析表 $PT$ と逐次処理に基づく単純順位構文解析の関係について述べる. 次に第 7.2 節において, Procedure Precedence_Parallel_Parser を用いることにより, 単純順位文法に対する逐次型の構文解析と同じ構文解析木が得られることを示す. 最後に第 7.3 節において, 提案した並列構文解析アルゴリズムが必要とするプロセッサ数と計算時間について考える.

## 7.1 構文解析表 $PT$ と逐次処理に基づく単純順位構文解析

構文解析表 $PT$ は本来 $n \times n \times |N|$ の 3 次元配列であり, 各非終端記号に対して値 pebbled または unpebbled をとるものであるが, ここでは $PT$ を $n \times n$ の 2 次元下三角行列で表現し, その各要素が非終端記号のいずれかの値をとるものとする (例えば, 図 8 および 9 参照). さらに, 構文解析の手続きが 2 次元配列 $PT$ 上の $i$ 行 $j$ 列の要素を処理していることを, 要素の位置 $L[i,j]$ によって表す.

単純順位文法に対する逐次型の構文解析処理が, 並列構文解析アルゴリズムで使用した構文解析表 $PT$ 上で動作することを示すために, 第 4 章で述べた逐次型の処理方式を拡張して, 文法記号の他に構文解析表 $PT$ 上の位置を表す $L[i,j]$ もスタックに積むことにする. このとき, 逐次型の構文解析の状況は次のようになる.

(1) 生成規則 $A_i \rightarrow a_i\,(A_i \in N,\, a_i \in T)$ を適用する場合.

まず, 新たな終端記号 $a_i$ が入力される直前のスタックの最上段の内容 $A_{i-1} \in N$ と入力文字 $a_i$ との順位関係を比較すると, 必ず $A_{i-1} \lessdot a_i$ が成立っている. そこで, 入力 $a_i$ を位置 $L[i,i]$ と共にスタックの最上段に移して, 次の入力文字 $a_{i+1}$ との順位関係を比較する. このとき, 必ず $a_i \gtrdot a_{i+1}$ となるので, スタックの最上段に積まれた終端記号 $a_i$ を, 生成規則 $A_i \rightarrow a_i$ に従って $A_i$ に置き換える. 構文解析表 $PT$ の位置 $L$ を含めて状況の変化を式で記述すると, 次のようになる.

$$
\begin{aligned}
& (\,\$ \ldots A_{i-1},\ a_i\, a_{i+1} \ldots a_n \$,\ \lessdot\,) \\
\vdash\ & (\,\$ \ldots A_{i-1}\, L[i,i]\, a_i,\ a_{i+1} \ldots a_n \$,\ \gtrdot\,) \\
\vdash\ & (\,\$ \ldots A_{i-1}\, L[i,i]\, A_i,\ a_{i+1} \ldots a_n \$,\ ?\,).
\end{aligned}
$$

但し, $A_i$ と $a_{i+1}$ の間の順位関係は議論しないため, 記号 ? を用いている.

(2) 生成規則 $X \to YZ\,(X,Y,Z \in N)$ を適用する場合.

スタックの上に位置 $L[i,j]$ と非終端記号 $Y$ が新たに積まれた場合を考える. このとき, 非終端記号 $Y$ と新たな入力文字 $a_{j+1}$ の間には, 順位関係 $Y \lessdot a_{j+1}$ が成立っている. その後, 入力文字の部分列 $a_{j+1} \ldots a_k$ の処理が進み, スタックに積まれた上位 2 つの非終端記号が $Y$ と $Z$ になったと仮定する. なお, $Z$ をスタックに積む際には, 位置 $L[j+1,k]$ が一緒に積まれる. このとき, 非終端記号 $Z$ と次の入力 $a_{k+1}$ との間には, 順位関係 $Z \lessdot a_{k+1}$ または $Z \gtrdot a_{k+1}$ のいずれかが成立つ.

順位関係 $Z \gtrdot a_{k+1}$ が成立っている場合は, スタックの最上段の $Z$ とその下の $Y$ との間に順位関係 $\doteq$ が成立つ. 従って, 生成規則 $X \to YZ$ を適用して, $YZ$ を $X$ で置換える. さらに, 位置 $L[i,j]$ と $L[j+1,k]$ を統合して, 表 $PT$ 上での動作位置を $L[i,k]$ に置換える. 式で記述すると, 次のようになる.

$$
\begin{aligned}
& (\,\$ \ldots L[i,j]\,Y,\ a_{j+1}\,a_{j+2} \ldots a_k\,a_{k+1}\,a_{k+2} \ldots a_n\,\$,\ \lessdot\,) \\
\vdash\ & \ldots \\
\vdash\ & (\,\$ \ldots L[i,j]\,Y\,L[j+1,k]\,Z,\ a_{k+1}\,a_{k+2} \ldots a_n\,\$,\ \gtrdot\,) \\
\vdash\ & (\,\$ \ldots L[i,k]\,X,\ a_{k+1}\,a_{k+2} \ldots a_n\,\$,\ ?\,).
\end{aligned}
$$

但し, $X$ と $a_{k+1}$ の間の順位関係は議論しないため, 記号 ? を用いている.

一方, 順位関係 $Z \lessdot a_{k+1}$ が成立っている場合は, 先に $a_{k+1}$ を次の入力として受付ける.

$$
\begin{aligned}
& (\,\$ \ldots L[i,j]\,Y,\ a_{j+1}\,a_{j+2} \ldots a_k\,a_{k+1}\,a_{k+2} \ldots a_n\,\$,\ \lessdot\,) \\
\vdash\ & \ldots \\
\vdash\ & (\,\$ \ldots L[i,j]\,Y\,L[j+1,k]\,Z\,,\ a_{k+1}\,a_{k+2} \ldots a_n\,\$,\ \gtrdot\,) \\
\vdash\ & (\,\$ \ldots L[i,j]\,Y\,L[j+1,k]\,Z\,L[k+1,k+1]\,a_{k+1}\,,\ a_{k+2} \ldots a_n\,\$,\ ?\,).
\end{aligned}
$$

但し, $a_{k+1}$ と $a_{k+2}$ の間の順位関係は議論しないため, 記号 ? を用いている.

上で述べた構文解析の動作において, スタックの上に新しく積まれた位置 $L$ の軌跡は, 構文解析表 $PT$ 上で見た場合の単純順位文法に対する逐次型の構文解析処理の動作と考えることができる. すなわち, スタックを用いた逐次型の構文解析処理は, 表 $PT$ 上でポインタをたどる動作と等価である. 図 8 に, 構文解析表 $PT$ 上に描いた $L$ の軌跡の一例を示す.

### 7.2 Procedure Precedence_Parallel_Parser が生成する構文解析木

まず, 第 5 章で提案した Procedure Precedence_Parallel_Parser の各反復において, 構文解析表 $PT$ の要素 $PT[i,j,X]$ が値 pebbled をもつ時, $X \overset{*}{\underset{\mathrm{rm}}{\Longrightarrow}} a_i \ldots a_j$ であることを示そ

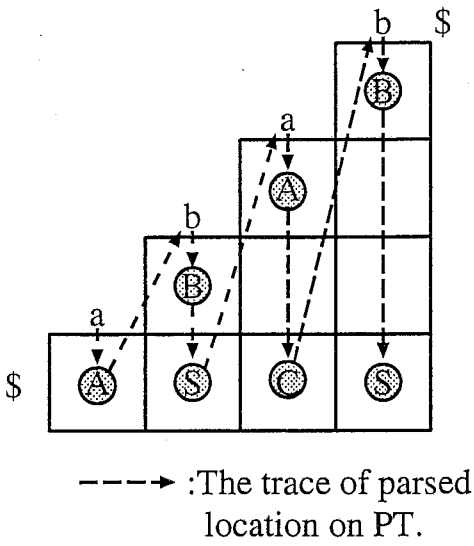


図 8: 構文解析表 $PT$ 上における位置 $L$ の軌跡

う. 実際もしこれが正しければ, 構文解析木の候補 $\mathcal{T}$ の節点 $(1, n, S)$ からはポインタが出発しないことに注意すると, Procedure Precedence_Parallel_Parser のステップ 9 で

$$PT[1, n, S] = \text{pebbled}$$

が成立つとき, 入力文字列 $a_1 \ldots a_n$ は与えられた文法で受理できることがわかる.

まず, Procedure Precedence_Parallel_Parser のステップ 2 では, 入力文字 $a_i$ に対して $A_i \to a_i$ なる生成規則が存在するとき, 構文解析表 $PT$ の対角要素 $PT[i, i, A_i]$ を pebbled している. ところで本稿では, 生成規則の記載方法を Chomsky 標準形に限定している. また, 単純順位文法では右辺が一致し左辺が異なる生成規則はとらないことから, 逐次型の構文解析処理においても, 各入力文字の終端記号に対してただ一つの非終端記号が生成される. よって, $PT[i, i, A_i]$ が値 pebbled をとる時, $A_i \overset{*}{\underset{\text{rm}}{\Longrightarrow}} a_i$ である.

次にステップ 3 では, 構文解析表 $PT$ の要素 $(i, j, Y)$ において条件 (1) ~ (4) が成立つ場合に, 要素 $(i, j, Y)$ から要素 $(i, Right[j, Y], X)$ にポインタが付加される (図 9(a)). 但し, $X$ は条件 (4) を満たす非終端記号である. このとき, 逐次型の構文解析処理の状況は, 第 7.1 節の議論より次のように記述される.

$$
\begin{aligned}
& (\,\$ \ldots L[i, j]\, Y,\ a_{j+1}\, a_{j+2} \ldots a_n \$,\ \lessdot\,) \\
\vdash\ & \ldots \\
\vdash\ & (\,\$ \ldots L[i, j]\, Y\ L[\,j+1, Right[j, Y]\,]\ Z,\ a_{Right[j,Y]+1}\, a_{Right[j,Y]+2} \ldots a_n \$,\ \gtrdot\,) \\
\vdash\ & (\,\$ \ldots L[i, Right[j, Y]]\, X,\ a_{Right[j,Y]+1}\, a_{Right[j,Y]+2} \ldots a_n \$,\ ?\,).
\end{aligned}
$$

但し, $X$ と $a_{Right[j,Y]+1}$ の間の順位関係は議論しないため, 記号 ? を用いている.

一方ステップ 4 では, 構文解析表 $PT$ の要素 $(i,j,Z)$ において条件 (5) ~ (8) が成立つ場合に, 要素 $(i,j,Z)$ から要素 $(Low[i,Z],j,X)$ にポインタが付加される (図 9(b)). 但し, $X$ は条件 (8) を満たす非終端記号である. このとき, 逐次型の構文解析処理の状況は, 第 7.1 節の議論より次のように記述される.

$$
\begin{aligned}
&(\,\$\ldots L[Low[i,Z],i-1]\,Y,\ a_i\,a_{i+1}\ldots a_j\,,\ a_{j+1}\,a_{j+2}\ldots a_n\,\$,\ \lessdot\,)\\
\vdash\ &\ldots\\
\vdash\ &(\,\$\ldots L[Low[i,Z],i-1]\ Y\ L[\,i,\,j\,]\ Z,\ \ a_{j+1}\,a_{j+2}\ldots a_n\,\$,\ \gtrdot\,)\\
\vdash\ &(\,\$\ldots L[Low[i,Z],j]\,X,\ a_{j+1}\,a_{j+2}\ldots a_n\,\$,\ ?\,).
\end{aligned}
$$

但し, $X$ と $a_{j+1}$ の間の順位関係は議論しないため, 記号 ? を用いている.

よって, Procedure Precedence_Parallel_Parser のステップ 3, 4 で新たに付加されたポインタをたどって節点 $(i,j,X)$ から $(k,\ell,W)$ へ行くことができるならば, 逐次型の構文解析においては次のような処理が可能である.

$$
\begin{aligned}
&(\,\$\ldots L[i,\,j]\,X\,,\ a_{j+1}\,a_{j+2}\ldots a_n\,\$,\ ?\,)\\
\vdash\ &\ldots\\
\vdash\ &(\,\$\ldots L[k,\ell]\,W,\ a_{\ell+1}\,a_{\ell+2}\ldots a_n\,\$,\ ?\,).
\end{aligned}
$$

但し, $k \le i,\ j \le \ell$ である. これは, もし $X \overset{*}{\underset{\mathrm{rm}}{\Longrightarrow}} a_i \ldots a_j$ ならば $W \overset{*}{\underset{\mathrm{rm}}{\Longrightarrow}} a_k \ldots a_\ell$, すなわち, 入力文字列のより長い部分列が解析できたことを意味している. さらにこのとき, Procedure Precedence_Parallel_Parser のステップ 7 より, $PT[i,j,X]$ の値が pebbled ならば, $PT[k,\ell,W]$ の値も pebbled となる.

以上により, 一般に $PT[i,j,X]$ の値が pebbled ならば, $X \overset{*}{\underset{\mathrm{rm}}{\Longrightarrow}} a_i \ldots a_j$ となることが示された.

なお, Procedure Precedence_Parallel_Parser のステップ 10 においてオイラーツアー法を適用すると, 節点 $(1,n,S)$ から構文解析木の節点のみを通過して, これらの節点に深さ優先順の番号をつけることができる. 従って, 順位のついた節点のみを選ぶことによって, 構文解析木を取出すことができる.

### 7.3 並列アルゴリズムの評価

第 5 章でも指摘したように, 本稿で提案した並列構文解析アルゴリズムの主要なステップは, 構文解析表 $PT$ の各要素に関して並列的に実行できる. 入力文字列の長さを $n$, 非終端

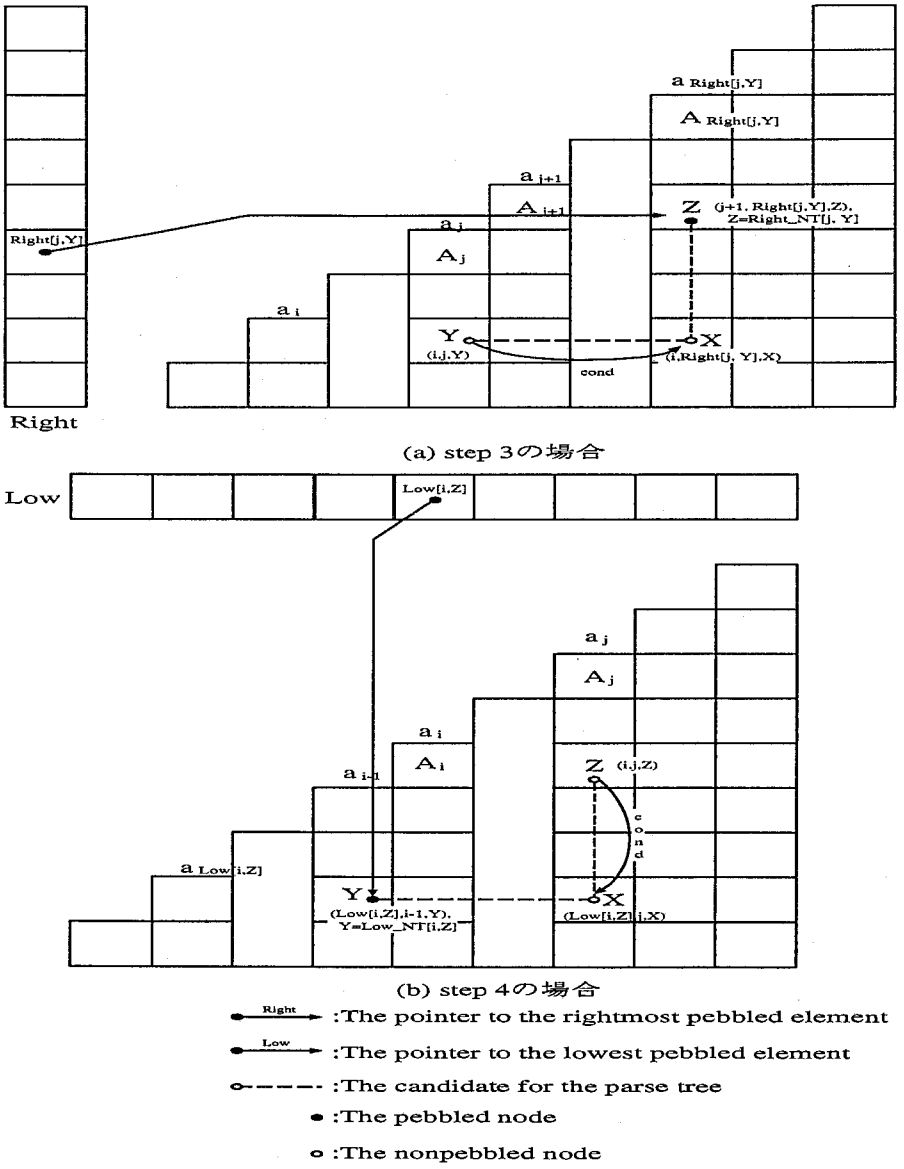


図 9: 構文解析表 $PT$ の要素間の依存関係

記号の数を $|N|$ とするとき, その要素数は $|N|n^2$ である. いま $|N|$ を定数と考えると, 必要なプロセッサの数は $O(n^2)$ となる.

一方, 葉の数が $n$ の二分木に対するペブルゲーム法の反復回数は, 最悪の場合で $\log n$ 回である [2]. また, ステップ 8 において, *Right* などの各表の要素の値を $n$ 個の添字 $i$ に対して並列的に求めている. 文献 [2] には, $m$ 個のデータの最大値または最小値を $m$ 個のプロセッサを用いて $O(\log m)$ 時間で求める並列アルゴリズムが示されている. 本稿で提案した並列構文解析のステップ 8 では, $|N|n$ 個のデータの最大値または最小値を $i$ について並列的に求めているので, 全体で $|N|n^2$ 個のプロセッサで $O(\log n)$ 時間を要する. よって, アルゴリズム全体では $O(n^2)$ 個のプロセッサで $O(\log^2 n)$ 時間を要することがわかる.

## 8 並列計算機 CM-5 への実装

提案したアルゴリズムの性能を評価するために, 並列計算機 CM-5 上への実装を行った. 実際に利用した CM-5 は 32 個の処理ノードから成り, 各処理ノードには標準的な SPARC プロセッサと合計 4 個のベクトルプロセッサが搭載されている. また, 各処理ノードには 8 Mbyte のローカル・メモリが 4 個装備されており, 大規模なデータの取扱いが可能であるとともに, データ・ネットワークを介して各処理ノード間で 1 対 1 のデータ通信を行うことができる. プログラムのコーディングにはデータ並列型のプログラミング言語である C* [6] を用いた. C* は CM-5 用に拡張された C 言語であり, 次のような特徴をもっている.

(1) プロセッサの構成を指定できる (2 次元のメッシュ等に固定されない).

(2) 処理途中で, プロセッサの構成を変更できる.

(3) ある条件を満たすプロセッサだけを選択して動作させることができる.

本稿で提案した並列構文解析アルゴリズムは, CREW P-RAM での実現を前提としているが, CM-5 は分散メモリ型の並列計算機であり, データ並列型の処理を行う場合には, 他のプロセッサに付属するローカル・メモリへアクセスすることはできない. 従って, CREW P-RAM の特徴である共有メモリにあるデータへのアクセスを, 各プロセッサがもつローカル・メモリへのアクセスに変更する必要がある. そこで, 共有メモリが保有すべきデータを, 各プロセッサのローカル・メモリに配置することにした.

今回実装したプログラムでは, プロセッサを $(n+2) \times (n+2) \times (|N|+1)$ の 3 次元格子状に配置した. また, 各プロセッサのローカルメモリには, 必要となる情報を全て保持させ

た．すなわち，各プロセッサは対応する構文解析表 $PT$ の位置，$PT$ の当該要素においてポインタ $cond$ が指す $PT$ の要素の位置，構文解析木において親に相当する節点の位置，兄弟に相当する節点の位置のほか，入力文字列，順位関数，生成規則，表 $Right$，$Right\_NT$，$Low$，$Low\_NT$ の全要素をもつ．図 10 に，プログラムのデータ構造と各プロセッサが保持するデータの項目を示す．

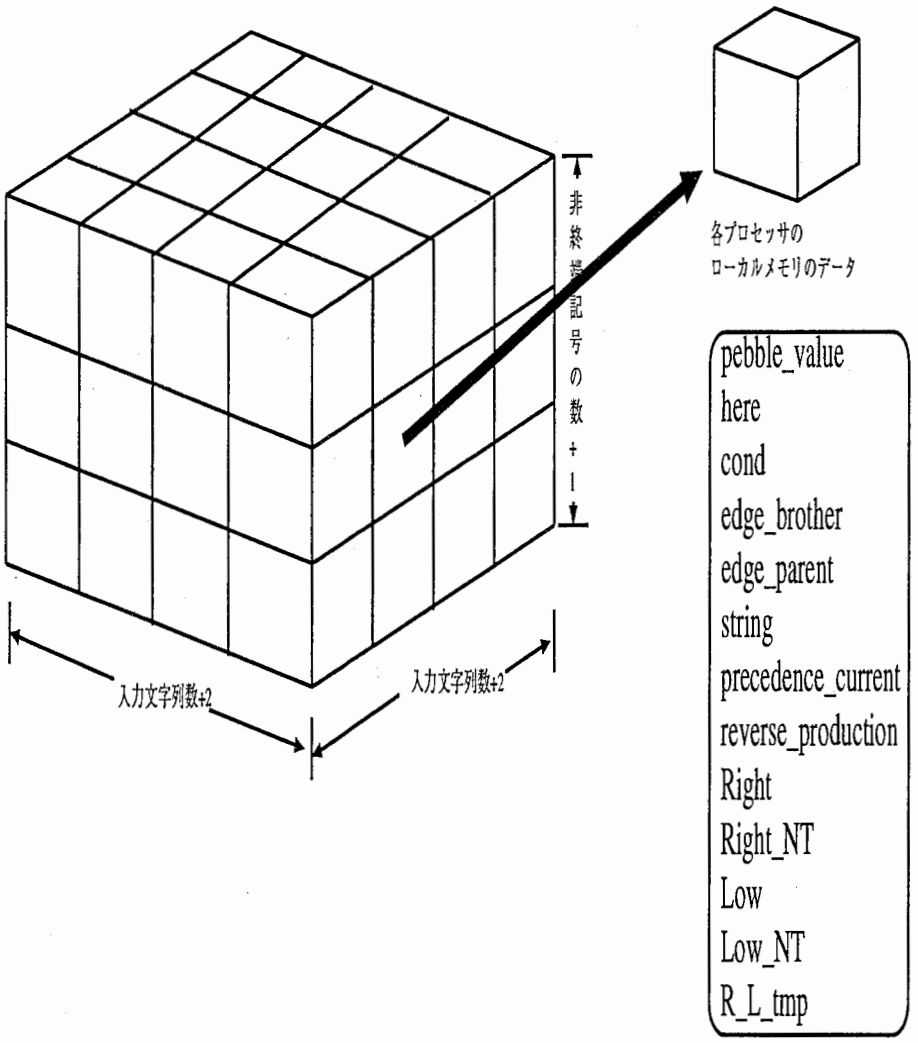


図 10: 実装したプログラムのデータ構造

## 9 実験結果

第 3 章で Chomsky 標準形に変換した単純順位文法の例として示した

$$G_2 = (\{A, B, C\}, \{a, b\}, \{S \to AB, S \to CB, C \to SA, A \to a, B \to b\}, S, \$)$$

を用いて, 提案した並列構文解析アルゴリズムの処理時間を測定した. 入力文字列としては, 文法 $G_2$ で受理し得るものを用い, 入力文字数 $n$ を 2, 10, 20, 30, 40, 50, 60, 70, 80, 90, 100 と変化させて実験を行った. CM-5 へ実装したプログラムによる処理時間を, 表 3 に示す. なお表 3 には, $n = 10$ 以上の場合について, 処理時間の増加量もあわせて示してある.

表 3 : CM-5 による並列構文解析アルゴリズムの処理時間とその増加量

| 入力文字数 | 処理時間 (秒) | 増加時間 (秒) |
|---|---|---|
| 2 | 0.045402 | — |
| 10 | 0.099378 | 0.099378 |
| 20 | 0.207033 | 0.107655 |
| 30 | 0.231287 | 0.024254 |
| 40 | 0.448540 | 0.217253 |
| 50 | 0.516167 | 0.067627 |
| 60 | 0.551283 | 0.035116 |
| 70 | 0.955156 | 0.403873 |
| 80 | 1.256573 | 0.301417 |
| 90 | 1.316108 | 0.059535 |
| 100 | 1.812640 | 0.496532 |

表 4 : Procedure Precedence_Parallel_Parser のステップごとの処理時間

| ステップ | 処理時間 (秒) | |
|---|---|---|
| | 入力文字列数 $n = 10$ | 入力文字列数 $n = 100$ |
| 2 | 0.000802 | 0.017495 |
| 3,4 | 0.001746 | 0.042702 |
| 5,6,7 | 0.011080 | 0.059403 |
| 8 | 0.070119 | 1.035073 |

一方, 入力文字列長 $n = 10$ と $n = 100$ の場合について, 並列構文解析の手続き Procedure Precedence_Parallel_Parser の各ステップごとの処理時間を計測した結果を, 表 4 に示す.

## 10 処理速度についての考察

理論的には, 処理時間は入力文字列長の対数に比例する割合で増加するものと考えられるが, 表 3 からは必ずしもそのような傾向は観測されない. また, 入力文字列長を増加させた場合の処理時間の延びにもばらつきが大きい. 一方, 表 4 からは, 最大値・最小値を計算してそれをすべてのプロセッサにブロードキャストしているステップ 8 の処理時間が, 全体の大半を占めていることがわかる. 特に, 入力文字列数が多くなるにつれてその傾向は顕著になっている. これらの処理は, CREW P-RAM を用いた場合は共有メモリによって効率よく処理できる部分であり, 分散メモリ型の並列計算機 CM-5 では, 多くの処理時間を消費したものと考えられる.

今回実装したプログラムでは, プロセッサの配置を立方体としている. しかし, 構文解析そのものとは関係のない処理をしている厳密な上三角部分を削除すれば, 仮想的なプロセッサ数と実際のプロセッサ数の比を約半分にできるため, 処理時間を大幅に削減できるものと考えられる. また, 各プロセッサのローカル・メモリに保持させている文法規則や順位表を圧縮して記憶させる等の方法によって, メモリ消費量を削減することも課題である.

## 11 おわりに

本稿では, 単純順位文法に対して $O(n^2)$ 個のプロセッサを用いて $O(\log^2 n)$ 時間で構文解析を実行する並列アルゴリズムを提案し, 実際の並列計算機 CM-5 に実装してその性能を評価した. 提案したアルゴリズムは CREW P-RAM の特徴である共有メモリに対するアクセスを利用することによって効率的に動作するものであるため, 分散メモリ型のシステムである CM-5 に実装するためには, 頻繁にアクセスする必要がある情報を並列的に処理を行う各プロセッサに配置しなければならなかった. そのため, プロセッサ間通信に処理時間の多くを費やし, 必ずしも理論通りの性能は観測されなかった. しかしながら, 世の中で実現されている大規模並列計算機は, 分散メモリ型のものが殆どである. 従って, 現実的にも有効な並列構文解析アルゴリズムを開発するためには, 各プロセッサのローカル・メモリに保持する必要があるデータについて, さらに吟味する必要があると考えられる.

なお, 本稿で対象とした単純順位文法は, 比較的制限事項が多い文法と言うことができる.

実際のプログラム言語や自然言語は, 単純順位文法よりも制限が少なく, 生成能力の高い文法のクラスに属するため, 逐次処理よりも並列処理の方が高速に解析できる可能性を含んでいる. その意味で, より生成能力の高い文法の並列処理のアルゴリズムの考案及び実装が, 残された研究課題であると考えられる.

## 参考文献


[1] Aho, A. V., Sethi, R. and Ullman, J. D.: *Compilers (Principles, Techniques and Tools)*. Addison-Wesley, 1986. 原田 賢一 訳: コンパイラ 1,2 (原理・技法・ツール). サイエンス社, 1990.

[2] Gibbons, A. and Rytter, W.: *Efficient Parallel Algorithms*. Cambridge University Press, 1988.

[3] 嵩 忠雄, 都倉 信樹, 谷口 健一: 形式言語理論 (情報とシステムシリーズ). 電子情報通信学会, 1988.

[4] ホップクロフト, J. and ウルマン, J. (野崎, 高橋, 町田, 山崎 訳): オートマトン 言語理論 計算論 1,2. サイエンス社, 1984.

[5] 井上 謙蔵: コンパイラ. 丸善, 1994.

[6] Thinking Machines Corporation: *C* Programming Guide*. Cambridge, Massachusetts, 1993.

[7] Tomita, M.: *Generalized LR Parsing*. Kluwer Academic Publishers, 1991.