構文解析の高速化に関する研究

1998年1月

博士（工学）

椎名広光

豊橋技術科学大学

構文解析の高速化に関する研究

1998 年 1 月

博士 (工学)

椎名 広光
豊橋技術科学大学

論文要旨

自然言語処理やプログラム言語の処理の大部分は, 形式言語理論における文脈自由言語及びその部分クラスの言語を対象としており, 既存の解析手法もそれに合わせた方法である. 言い替えれば, 文脈自由言語やその部分クラスの言語の処理方式に合わない部分を無視して処理しているのが現実である. しかしながら, 現実の言語現象には, 文脈自由言語の枠組では表現できないものが少なからず存在する. そこで, コンピュータの並列化や処理速度の高速化によって, これまでの逐次処理より高速な構文解析や, 文脈自由言語より広いクラスの言語に対する実用的な構文解析が実現できる可能性がでてきた. よって本研究では, それらを実現するアルゴリズムを考案することを目的とする.

第 2 章では, 単純順位文法に対する並列構文解析アルゴリズムを示す. 単純順位文法は文脈自由文法や LR文法の部分クラスであり, 単純順位文法に対するコンパイラの構文解析に部分的に使用されることが多い. 第 2 章で提案する並列アルゴリズムは, 文脈自由文法や LR 文法に対する構文解析の並列構文解析アルゴリズムより少ない $O(n^2)$ 個のプロセッサを用いて $O(\log^2 n)$ 時間で実行される.

第 3 章では, LR 文法に対する並列構文解析アルゴリズムを示す. LR 文法はほとんどのプログラム言語の文法をカバーする文法で, コンパイラなどの構文解析で良く使用される. 既に文脈自由文法に対しては, $O(n^6)$ 個のプロセッサを用いて $O(\log^2 n)$ 時間で処理する並列アルゴリズムが提案されているが, LR文法に対して $O(n^3)$ 個のプロセッサを用いて $O(\log n)$ 時間で処理する並列アルゴリズムを新たに提案する.

第 4 章では, unrestricted LR 文法 及び unrestricted LR 構文解析法を提案する. Harris によって提案された unrestricted $SLR(1)$ 構文解析法や unrestricted $LALR(1)$ 構文解析法は, LR構文解析法を文脈自由言語より大きいクラスに拡張する方法として知られている. しかし, unrestricted $SLR(1)$ 構文解析法 や unrestricted $LALR(1)$ 構文解析法は従来から知られている $SLR(1)$ 構文解析や $LALR$構文解析の拡張で, その性質として先読みの文字列数が終端記号の 1 個に限定されているため, 構文解析可能な言語のクラスが限定される. そこで, 第 4 章では, 先読み文字列を入力文字列から非終端記号と入力の最後を示す特殊記号\$に変更することによって, 先読み文字列の個数を限定しない unresticted $LR(k)$ 文法, 及び, unrestricted $LR(k)$ 構文解析法を提案する.

第 5 章では, 構文解析を行なう対象とする言語のクラスを, 句構造文法 の一つである unrestricted 文法 が生成する言語のクラスとし, その言

語を構文解析する方法としてGLR構文解析を拡張した$UGLR$構文解析 (unrestricted generalized LR 構文解析 の 略) を新たに提案する. また, この$UGLR$構文解析器が受理できる言語のクラスが文脈自由言語のクラスを含むことを, $UGLR$構文解析に与える文法に制限をつけると, $UGLR$構文解析の動作が Earley 法と一致することによって明らかにする.

そして, 最後の第 6 章において, 本研究で得られた結果をまとめ, 今後の研究課題について述べる.

Studies on Acceleration of Parsing Algorithms

abstract

Most of parsers for natural language processing and compilers parse context-free languages or languages in some subclass of context-free languages. On the other hand, the parallelization and acceleration of parsers may enable us to accelerate parsing and to extend the class of practically parsible languages. In this dissertation, we first develop parallel parsing algorithms for some subclasses of context-free languages and develop parsers for these extensions of languages.

In Chapter 2, we describe a parallel parsing algorithm for the class of simple precedence languages. This class of languages which is a subclass of context-free languages and LR languages but is wide enough for representing arithmetic expressions, and used often in compiler systems. Proposed parallel parsing algorithm works in $O(\log^2 n)$ time using $O(n^2)$ processors on CREW P-RAM.

In Chapter 3, we describe a parallel parsing algorithm for the class of LR languages which is a subclass of context-free languages used in many natural language parsers and compiler systems, and we show that this algorithm works in $O(\log n)$ time using $O(n^3)$ processors on CREW P-RAM.

In Chapter 4, we describe the unrestricted $LR(k)$ grammar and its parser. The unrestricted $SLR(1)$ language and the unrestricted $LALR(1)$ languages proposed by Harris are known to be extensions of context-free languages. But, such extensions of LR parsers are determinstc parsers with the lookahead strings consisting of terminal symbols and nonterminal symbols which allow one pass parsing from left to right of the input strings. However, in the unrestricted $SLR(1)$ grammar and the unrestricted $LALR(1)$ grammar, the length of the lookahead strings is restricted to be one. Due to this restriction, languages which can be parsed by these parsers are still limited. By changing the definition of the set of the lookahead strings to be consisting of nonterminal symbols or the end symbol instead of the input symbols, we propose the unrestricted $LR(k)$ grammar which is the extension of the $SLR(1)$ grammar and the

$LALR(1)$ grammar.

Chapter 5 proposes the $UGLR$ parser as an extension of the GLR parser. The $UGLR$ parser is strong enough for parsing deterministically any phrase structure language if it is a recursive set and can parse any context free language as fast as a conventional GLR parser. Natural language processing often requires a parser for languages belonging to classes larger than that of context free languages, and the proposed parser is useful for this purpose.

Moreover, in Chapter 6, we conclude with a discussion of the results and suggest future works.

A. 発表論文リスト

1. Hiromitsu Shiina and Shigeru Masuyama: Proposal of the unrestricted LR(k) grammar and its parser, Mathematica Japonica, Vol. 46, No.1, pp. 129-142, 1997. (第 4 章に対応)

2. 椎名広光, 増山繁: 単純順位文法に基づく並列構文解析アルゴリズム, 電子情報通信学会論文誌 D-I, (採録決定). (第 2 章に対応)

3. 椎名広光, 増山繁: LR 文法に対する並列構文解析アルゴリズム, 電子情報通信学会論文誌 D-I, (投稿中). (第 3 章に対応)

B. 国際会議論文

1. Hiromitsu Shiina and Shigeru Masuyama: Proposal of the UGLR Parser for Phrase Structure Grammars. Proc. of Forth Natural Language Processing Pacific-Rim Symposium, pp. 529-532, 1997. (第 5 章に対応)

C. 学会研究発表

1. 椎名広光, 増山繁: Unrestricted LR 文法 及び unrestricted LR 構文解析法の提案, 京都大学 数理解析研究所 数理解析研究所講究録 950, pp. 221-227, (1996).

2. 椎名広光, 増山繁: LR 属性を拡張した R-L 属性の提案, 電子情報通信学会技術研究報告 (コンピュテーション研究会), COMP95-75, 95 巻, 95 号, pp. 21-30, (1996).

3. 椎名広光, 増山繁: LR 構文解析の並列アルゴリズムについて, 京都大学 数理科学講究録刊行会 数理科学講究録 871, pp. 145-153, (1994).

4. 椎名広光, 増山繁: 属性文法に基づく意味解析の並列アルゴリズムについて, 情報処理学会第 45 回 (平成 4 年 後期) 全国大会講演論文集, 1 巻, pp. 73-74, (1992).

5. 椎名広光, 増山繁: 属性文法に基づく意味解析の並列アルゴリズムの計算量について, 平成 4 年度電気関係学会 東海支部 連合大会講演論文集, p. 298, (1992).

6. 椎名広光, 増山繁: 属性文法に基づく意味解析の並列アルゴリズムについて, 92 年度 夏の LA シンポジウム, 情報基礎理論ワークショップ, pp. 5-8, (1992).

7. 椎名広光, 増山繁: L 属性文法に基づく意味解析の並列アルゴリズムの開発, 情報処理学会第 44 回 (平成 4 年 前期), 5 巻, pp. 129-130, (1992).

目 次

第 1 章

序論

1.1 研究の背景と動機

電子技術の発展により, 高機能化及び高速化された単一プロセッサが開発され, 一方、計算機システム技術の発展により, 複数のプロセッサを持つ並列コンピュータやネットワークによる分散処理が開発された. これらの開発は, 単独又は複数のコンピュータの処理速度の向上をもたらし, これまでは処理の限界と考えられてきた問題をより速く解くことも可能とし, 従来は実際上処理時間が大きいため処理不可能と考えられてきた問題にも, コンピュータ処理の糸口を与えてきている.

コンピュータの処理速度の向上に対して, 言葉の理解や言語処理のために扱われる言語は自然言語（natural language）と 人工言語（artifical language）の 2 つに大別される.

自然言語処理では, その応用として機械翻訳, 文書処理, 談話処理 及び音声認識などがあげられる. 特に文書処理や談話処理では大量のデータ処理が必要であったり, 音声認識などではリアルタイム処理を要求されるため, 実際上の処理時間が大きな問題となり, アルゴリズムが複雑なものは利用されなかったり考案されてこなかったのが現実である. 但し, 言語理論では複雑なアルゴリズムしか持たない幾つかの理論が提案されている. 例えば, GB 理論 (Government-Binding Theory, 統率・束縛理論)[9], GPSG (Generalized Phrase Structure Grammar, 一般化句構造文法)[9], LFG(Lexical-Functional Grammar, 語彙機能文法)[9] などである. しかし, これらの手法は生成文法と呼ばれる方法で, 言語現象の理解を目的にした理論であり, その理論が一般の自然言語に対して 100 %適合するものではなく, その自然言語近似度を高くすればするほど複雑性を増す. これらの

言語理論においては, アルゴリズムの実用性はほとんど考慮されず, コンピュータ処理への応用は LFG の一部以外は見られない.

一方, コンピュータ処理における人工言語は, C, PASCAL, BASIC などのプログラム言語である. プログラム言語で記述されたプログラムをコンピュータが直接処理できる機械語への変換には, インタプリタやコンパイラ [10] を用いる. インタプリタやコンパイラの内部処理では, 記述されているプログラムに対して 字句解析, 構文解析, 意味解析 及び コード生成の処理を行なう.

それらの処理は, 以下の通りである.

- 字句解析 : コンパイラ内部での最小単位となる 数, 変数名, 命令に切り分ける処理を行なう.
- 構文解析 : 記述されているプログラムがそれぞれのプログラム言語の文法規則から導出可能であることをチェックし, プログラムの 1 文ごとの導出過程の再現とその構造を明らかにする構文解析木を作成する.
- 意味解析 : プログラムの 1 文ごとの論理的な解釈の正当性を判定する.
- コード生成 : 記述されたプログラムに対応した機械語を生成する.

コンパイラの内部で行なわれる処理のうち自然言語処理では構文解析と意味解析が共通に利用され, 字句解析のかわりに形態素解析が利用される. 但し, 自然言語処理とコンパイラの内部で用いられる構文解析は共通している部分が多いが, 目的が相違している. コンパイラの場合では, 構文解析木が複数作成されるような曖昧性を排除した一つの木だけ作成するようにすれば良いのに対して, 自然言語処理では, 入力文字列の曖昧性処理も考慮しなければならないので, 構文解析木の候補者を複数個出力する処理も必要となる.

これまで自然言語処理やプログラム言語の処理のほとんどが, 形式言語理論における文脈自由言語 [4, 7] 及びその部分クラスの言語を対象としており, 解析手法もそれに合わせた方法である. 言い替えれば, 文脈自由言語やその部分クラスの言語の処理方式に合わない部分を無視して処理しているのが現実である. そこで, コンピュータの並列化や処理速度の高速化によって, これまでの逐次処理より高速な構文解析や文脈自由言語より広いクラスの言語の構文解析も可能となってきており, それにあわせたアルゴリズムも必要となってきている.

逐次処理のコンピュータの高速化に対して, 複数個のプロセッサを同時に利用する並列コンピュータの開発によってコンピュータの処理速度の高速化を図っている. その開発によって, 並列コンピュータ上での効率の良いアルゴリズムの考案が必要となってきた. しかし, 並列コンピュータといっても, プロセッサの結合方式にはメッシュ結合, 木結合, ハイパーキューブ結合などの形状があり, 結合方式によってそれぞれに合わせた最適な並列アルゴリズムが存在する. そこで, アルゴリズムを理論的に解析して, 並列処理の問題の本質を見究めるためには, プロセッサの結合形状やプロセッサ間の通信やメモリのアクセスなどを理想化した P-RAM (Parallel Random Access Machine)[19, 22] と呼ばれる仮想計算機モデルを利用する. なぜなら, 実際の並列計算機におけるデータのプロセス間の通信の問題は, プロセッサの結合方式の相違などによる並列計算機の種類だけ存在し, インプリメント上の問題と考えられるからである. そのため実際の並列計算機に依存しない並列アルゴリズムに共通する問題点や本質を明らかにするためには, P-RAM モデルのようなプロセッサの結合形状やプロセッサ間の通信やメモリのアクセスなどを理想化したモデル上で並列アルゴリズムを考案する必要がある.

ここで, P-RAM モデルにおいて並列アルゴリズムが効率の良い並列アルゴリズムであるとは, 有限個の入力に対して, 使用するプロセッサ数が多項式個でかつ計算時間が対数多項式個であることを意味する. そして, ある問題 Q が効率の良い並列アルゴリズムを持つ問題のクラス NC(Nick' クラス, 最初に提唱した Nick Pippenger にちなんだクラス) に属することは, 以下の定義で示される [19].

「問題 Q の入力データ長 n であるような任意の問題例に対して, PRAM 上で入力サイズ n の多項式個 ($O(n^k)$ 個,kは定数) のプロセッサを用いて, 対数多項式時間 ($O(\log^l n)$, lは定数) で解を求めることができるならば, Q はクラス NCに属するという. 」

既に W.Rytter によって $O(n^6)$ 個のプロセッサを用いて $O(\log^2 n)$ 時間で処理する文脈自由言語に対する構文解析の並列アルゴリズムが提案されている [19]. よって文脈自由言語の構文解析問題は NCであるので, 文脈自由言語より狭い言語のクラスの構文解析には, 効率の良い並列アルゴリズムが存在する. 例えば, 文脈自由文法の部分クラスには, 人工言語の代表である単純順位文法 [10, 20], LL 文法 [10, 20], LR 文法 [4, 7, 10, 20] などがある. つまり, 単純順位文法, LL 文法, LR 文法などの構文解析の場合, 言語のクラスが狭くなったことによって, 一般の文脈自由言語の構

文解析の並列アルゴリズムより, 使用するプロセッサ数や計算時間を削減できる可能性がある.

一方, 逐次処理では, 入力に対して多項式時間で処理可能であるが効率の良い並列アルゴリズムを持たないであろうと考えられている問題のクラスをP-完全 [19] な問題のクラスとしている. グラフの有向辺に容量を付加したネットワークに対して, 始点と終点間の最大流量を求める最大流問題がその例である. 文脈依存言語に対する構文解析の問題はP-領域完全であるがP-領域完全問題 [7] は並列の場合のP-完全な問題と等価であるので, P-完全といっても良い. そこで, 文脈自由言語より広いクラスに対する構文解析問題の場合は, 並列性よりも, 逐次処理における効率性を考える必要がある. つまり, より処理時間が少なくなるようにするか, または, 同じ処理時間でも構文解析可能な言語のクラスをより広くする必要がある.

1.2 本論文の内容

本論文では, 構文解析の処理の高速化と処理可能な言語のクラスを拡大する試みを述べている. 特に, 文脈自由言語より小さい言語のクラスに対する構文解析アルゴリズムについては, これまで知られていた方法より, より少ないプロセッサを用いる効率の良い並列アルゴリズムを提案している. そして, 文脈自由言語より大きい言語のクラス, または, 文脈自由言語より部分的に大きい言語のクラスを定義し, その効率的な逐次処理の構文解析アルゴリズムを提案している. 以下に, 各章で述べる問題と結果を示す.

第2章では, 「単純順位文法に対する並列構文解析アルゴリズム」を示す. 単純順位文法は数式を表現する程度の小さいクラスの文法であるが, それは, コンパイラの構文解析に部分的に使用されている手法である. また, 単純順位文法は文脈自由文法やLR文法の部分クラスであるので, 単純順位文法に対する並列アルゴリズムは, 文脈自由文法や LR 文法に対する構文解析の並列アルゴリズムで使用するプロセッサ数よりも少ない必要があり, $O(n^2)$ 個のプロセッサを用いて $O(\log^2 n)$ 時間で処理する並列アルゴリズムを提案する.

第3章では, 「LR 文法に対する並列構文解析アルゴリズム」を示す. LR 文法はほとんどのプログラム言語の文法をカバーする文法で, コンパイラなどの構文解析で良く使用される. 既に文脈自由文法に対しては, $O(n^6)$ 個のプロセッサを用いて $O(\log^2 n)$ 時間で処理する並列アルゴリズ

ムが提案されているが, LR文法に対しては $O(n^3)$ 個のプロセッサを用いて $O(\log n)$ 時間で処理する並列アルゴリズムを新たに提案する.

第 4 章では, 「unrestricted LR 文法 及び unrestricted LR 構文解析法の提案」を示す. Harris によって提案された unrestricted $SLR(1)$ 構文解析法 や unrestricted $LALR(1)$ 構文解析法 は LR構文解析法を文脈自由言語より大きいクラスに拡張する方法として知られている. しかし, unrestricted $SLR(1)$ 構文解析法 や unrestricted $LALR(1)$ 構文解析法は従来から知られている $SLR(1)$ 構文解析や $LALR$構文解析の拡張で, その性質として先読みの文字列数が終端記号の 1 個に限定されているため, 構文解析可能な言語のクラスが限定される. そこで, 本論文では先読み文字列を入力文字列から非終端記号と入力の最後を示す特殊記号\$に変更することによって, 先読み文字列の個数を限定しない unresticted $LR(k)$ 文法, 及び, unrestricted $LR(k)$ 構文解析法を提案する.

第 5 章の「GLR構文解析を拡張した$UGLR$構文解析の提案」では, 構文解析を行なう対象とする言語のクラスを句構造文法 の一つである unrestricted 文法 が生成する言語のクラスとし, その言語を構文解析する方法として GLR構文解析を拡張した $UGLR$構文解析 (unrestricted generalized LR 構文解析 の 略) を新たに提案する. また, この $UGLR$構文解析器が受理できる言語のクラスが文脈自由言語のクラスを含むことを, $UGLR$構文解析に与える文法に制限をつけると, $UGLR$構文解析の動作が Earley 法と一致することによって明らかにする.

1.3 文法 及び 言語の定義

本論文で利用する形式言語理論の記号を定義する。ここで定義する以外の記号や用語は, 各章での出現時に述べる. 形式言語理論の詳細については, [4, 7] などを参照されたい.

形式言語論において文法 G は, $G = (P, S, T, N)$ の形式で表される. そのうち, Nと Tは非終端記号及び終端記号の有限集合を表し, 終端記号はプログラムなどの入力となり得る記号で, 非終端記号は終端記号でない文法記号である. 構文解析木においては, 終端記号は木の葉に当たり, 非終端記号は木の節点に当たる. Pは生成規則の有限集合を表し, Sは開始記号を表す.

開始記号 Sから生成規則を何度か適用することによって終端記号だけの終端記号列を作ることが可能である. この終端記号列の有限集合を文法 G から生成される言語と呼び $L(G)$ と表す. また, $L(G)$ は, 形式的に以

下のように示される.

$$L(G) = \{w | w \in T^*, S \overset{*}{\Longrightarrow} w\}$$

$\overset{*}{\Longrightarrow}$は 0 回以上の生成規則の適用, T^*は終端記号の有限集合を示す.

生成規則の形式によって, 生成される言語のクラスがある. そこで文法ごとの形式について, 以下に述べる.

- 正規文法： 生成規則 $A \to aB, a \in T, A, B \in N$ の形式のみからなる文法
- 文脈自由文法： 生成規則 $\alpha \to \beta, \alpha \in N, |\alpha| = 1, |\beta| \leq 1$ の形式のみからなる文法, $|\alpha|$ と $|\beta|$ は, α, βの文字列長を示す.
- 文脈依存文法： 生成規則 $\alpha \to \beta, |\alpha| < |\beta|$ の形式のみからなる文法.
- 句構造文法：生成規則に制限のない文法.

以上の文法から生成される言語のクラスには, 包含関係があり,

正規文法で生成される言語のクラス
$\subseteq$ 文脈自由文法で生成される言語のクラス
$\subseteq$ 文脈依存文法で生成される言語のクラス
$\subseteq$ 句構造文法で生成される言語のクラス

である.

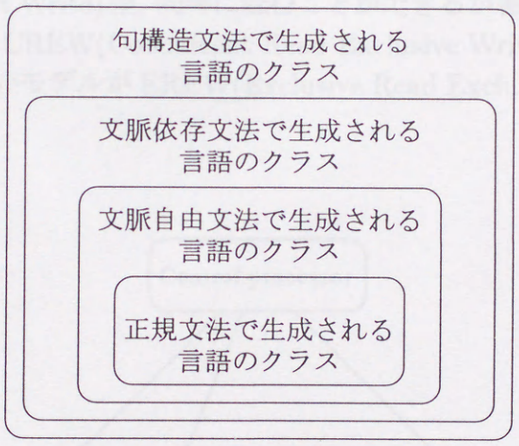


図 1.1: 言語の包含関係.
Fig. 1.1: Classfication of Languages

1.4 P-RAM モデルの定義

ここでは, 並列アルゴリズムが動作する並列計算機モデル P-RAM [19, 22] について述べる.

P-RAM モデルは, 共有メモリ (Common global random-access memory) と任意の有限個の RAM(Random Access Machine) からなり, 特に RAM を制御するプロセッサ (Control processor) を置く時, SIMD(Single Intruction Multiple Data stream) 型 P-RAM と呼ばれる.

また, 複数個の RAM と共有メモリ間の読み込みや書き込みのアクセスが, それぞれ同時にできるどうかによっても P-RAM モデルが分類され, 同一番地のメモリに同時に読み書きできるモデルが CRCW (Concurrent Read Concurrent Write) 型, 同時に読むことができるが書き込むことができないモデルが CREW(Concurrent Read Exclusive Write) 型, 同時には読み書きできないモデルが EREW(Exclusive Read Exclusive Write) 型がある.

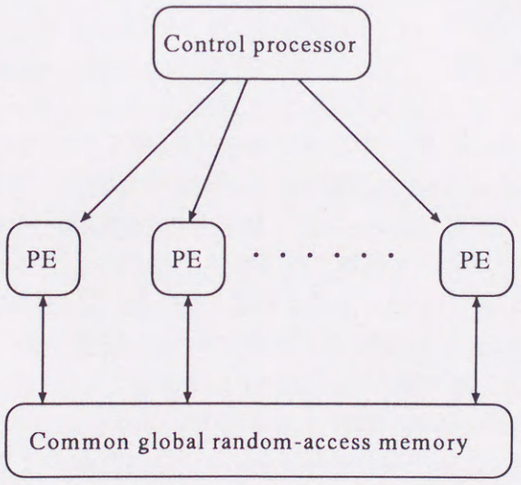


図 1.2: CREW P-RAM の構造.
Fig. 1.2: The structure of CREW P-RAM.

なお, 本論文では, CRCW, CREW, EREW P-RAM のうち, 実現され

ている並列コンピュータに近いモデルである CREW P-RAM を用いる.

第 2 章

単純順位文法に対する並列構文解析

2.1 まえがき

並列アルゴリズムの理論的な解析には, P-RAM(Parallel Random Access Machine)[19] が良く利用される. P-RAM は, 1 個の制御プロセッサ (Control processor) と複数個の並列プロセッサ, 及び, 共有メモリ (Common global random-access memory) から構成される SIMD 型並列計算機の理論的なモデルで, P-RAM の中でも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持つ並列計算機モデルを, CREW P-RAM (Concurrent Read Exclusive Write P-RAM) と呼んでいる (例えば, 文献 [19] 参照).

既に CREW P-RAM 上における構文解析の並列アルゴリズムがいくつか知られている. 特に文献 [19] では一般の文脈自由言語 [10, 7, 20, 27] 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 [13, 14], 及び, ブラケット言語 (Bracket Language)[19] とインプットドリブン言語 (Input-driven Language)[19] に対して $O(n/\log n)$ 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が紹介されている.

しかし, 一般の文脈自由言語に対する構文解析の並列アルゴリズムのように用いられるプロセッサ数が $O(n^6)$ 個と多い場合や, ブラケット言語や, インプットドリブン言語のように言語のクラスが小さい場合などは, コンパイラにおける構文解析に部分的にしか利用できない. そこで本論文では, コンパイラなどで利用される単純順位文法 [10, 20, 27](本論文の単

純順位文法の定義は文献 [20] に準拠) で生成される言語に対して CREW P-RAM 上で $O(n^2)$ 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する.

なお, アルゴリズムの説明などの形式的な部分は文献 [19] に準拠し, 単純順位文法 $G = (N, T, P, S)$, N: 非終端記号の集合, T: 終端記号の集合, P: 生成規則の集合, S: 開始記号 と入力文字列 $a_1 a_2 \ldots a_n$, $a_1, a_2, \ldots, a_n \in T$ が与えられるものとする. また, 構文解析実行の便宜上, 入力文字列の開始と終了を示す特殊記号 $\$(\notin N \cup T)$ を利用する.

2.2 準備

本論文で取り扱う単純順位文法, 及び, 構文解析の並列アルゴリズムで用いるペブルゲーム法について述べる.

2.2.1 単純順位文法の定義

単純順位文法は, 構文解析を逐次に進める際に, 適用すべき生成規則を文法記号 ($N \cup T$の要素) 間の順位関係 (precedence relation) のみで一意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, $X \lessdot Y$, $X \gtrdot Y$, 及び, $X \doteq Y$ の 3 種類がある. $X \lessdot Y$ の時, 「X は Y より順位が低い」と呼ばれ, 図 2.1(a) のように構文解析木上で X のすぐ右側の兄弟を根 $Z(\neq Y)$ とする時, その Zの部分木中の最左の節点または葉に Yがあることを意味している. 逆に $X \gtrdot Y$ の時, 「X は Y より順位が高い」と呼ばれ, 図 2.1(b),(c) のように Yのある部分木の根のすぐ左の兄弟を根 $Z(\neq Y)$ とする時, その Zの部分木中の最右の節点または葉に Xがあることを意味する. 一方, $X \doteq Y$ の時, 「X と Y は順位が等しい」と呼ばれ, 図 2.1(d) のように X, Y を右辺にもち, XYの順で並ぶ生成規則が存在することを意味している. なお, これら以外の文法記号間に順位関係は存在しない.

順位関係をより詳しく説明すると, 最右導出 $S \overset{*}{\underset{\mathrm{rm}}{\Longrightarrow}} \mu A v \underset{\mathrm{rm}}{\Longrightarrow} \mu\alpha v$ ($\overset{*}{\underset{\mathrm{rm}}{\Longrightarrow}}$は生成規則を 0 回以上用いた最右導出, $\underset{\mathrm{rm}}{\Longrightarrow}$は生成規則を 1 回用いた最右導出) において, $\mu = X_q X_{q-1} \cdots X_{k+1}$, または $\mu = \varepsilon$(空語), かつ, $\alpha = X_k X_{k-1} \cdots X_1$ とする時, 各隣接記号の間に次の関係があるものとする ($A \in N$, $v \in T^*$, $X_i \in N \cup T$).

(1) X_{i+1}と $X_i (i > k)$ の間: $X_{i+1} \doteq X_i$
または $X_{i+1} \lessdot X_i$,

(2) X_{k+1}と X_k の間: $X_{k+1} \lessdot X_k$,

(3) X_iと $X_{i-1} (1 < i \leq k)$ の間: $X_i \doteq X_{i-1}$,

(4) X_1と $head_1(v)$ の間: $X_1 \gtrdot head_1(v)$.

但し, $head_1(v)$ はv中の先頭の終端記号である.

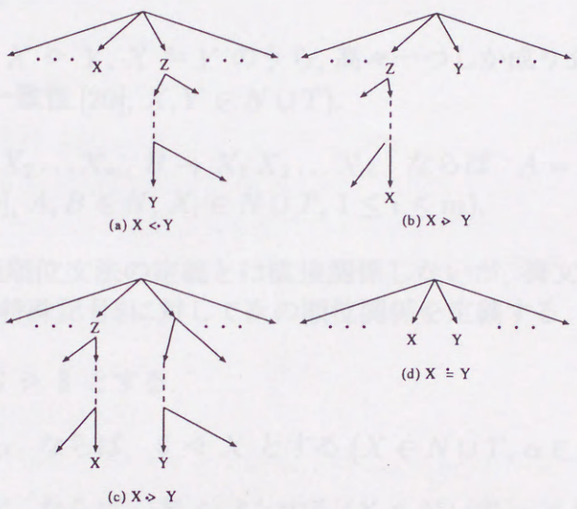


図 2.1: 構文解析木と順位関係の関係.

Fig. 2.1: The relation between a parse tree and the precedence relation.

以上の順位関係 $\lessdot$, $\gtrdot$, $\doteq$ を用いて, 単純順位文法を次のように定義する.

定義 [27] 文脈自由文法 $G = (N, T, P, S)$ が次の条件を満たす時, これを単純順位文法という.

(1) $A \to \varepsilon \notin P \ \ (A \in N)$.

(2) $A \overset{+}{\Longrightarrow} A$ なる導出は起こらない ($A \in N$, $\overset{+}{\Longrightarrow}$ は 1 回以上の導出, $A \in N$).

(3) $X \lessdot Y, X \gtrdot Y, X \doteq Y$ のうち, 高々一つしか成り立たない (**順位関係の一致性** [20], $X, Y \in N \cup T$).

(4) $A \to X_1 X_2 \ldots X_m, B \to X_1 X_2 \ldots X_m$ ならば $A = B$ (**逆方向決定性** [20], $A, B \in N, X_i \in N \cup T, 1 \leq i \leq m$).

なお, 単純順位文法の定義とは直接関係しないが, 構文解析実行の便宜上追加する特殊記号\$に対して次の順位関係を定義する.

(1) $\$ \lessdot S, S \gtrdot \$$ とする.

(2) $S \overset{+}{\Longrightarrow} X\alpha$ ならば, $\$ \lessdot X$ とする ($X \in N \cup T, \alpha \in (N \cup T)^*$)

(3) $S \overset{+}{\Longrightarrow} \alpha X$ ならば, $X \gtrdot \$$とする ($X \in N \cup T, \alpha \in (N \cup T)^*$).

また, 逐次処理の構文解析の処理の概略は, **第 6.1 節**で説明する.

2.2.2 ペブルゲーム法

本論文で提案する構文解析の並列アルゴリズムはペブルゲーム法 [19] を拡張したものを用いる. そこで, 本節では拡張する前のペブルゲーム法の概要を紹介する. 但し, ペブルゲーム法の詳細は, 文献 [19] 等を参照されたい.

ペブルゲーム法は二分木 $T = (V, E)$ に対して処理を行ない, 各節点が持つラベルに対して, 葉から根への順にラベル "pebbled" を付すアルゴリズムである. ペブルゲーム法のステップは大きく以下の 2 つに分けられる. また, 図 2.2 に ペブルゲーム法の Step 2 の実行例を示す. なお, 予め与えられた二分木の葉の数は n 個とする.

Step 1: 二分木のすべての葉に ラベル "pebbled" を付す.

Step 2: 以下に示す **手続き** を $\lfloor \log n \rfloor$ 回 繰り返す.

- **手続き activate**
  ポインタ $cond(v_1) = v_1$を満たす (文献 [19] の non-active な条件) 節点 v_1 に対して, 一方の子供の節点 v_2 にラベル "pebbled" が付されているならば, v_1 のポインタ $cond(v_1)$ はラベル "pebbled" が付されていない子供の節点 v_3 を指すものとする. また, 節点 v_2, v_3 のいずれにもラベル "pebbled" が付されていなければ, ポインタ $cond(v_1)$ は v_2 と v_3 のいずれか一方を指すものとする.
- **手続き square1**
  ある節点 v のポインタ $cond(v)$ が指す節点から更にポインタ $cond(cond(v))$ によって指される節点が存在するならば, $cond(v) := cond(cond(v))$ として, ポインタの指す先を変更する.
- **手続き square2**
  手続き square1 を繰り返す.
- **手続き pebble**
  ある節点 v のポインタが指す節点 $cond(v)$ にラベル "pebbled" が付されていれば, 節点 v にもラベル "pebbled" を付す. □

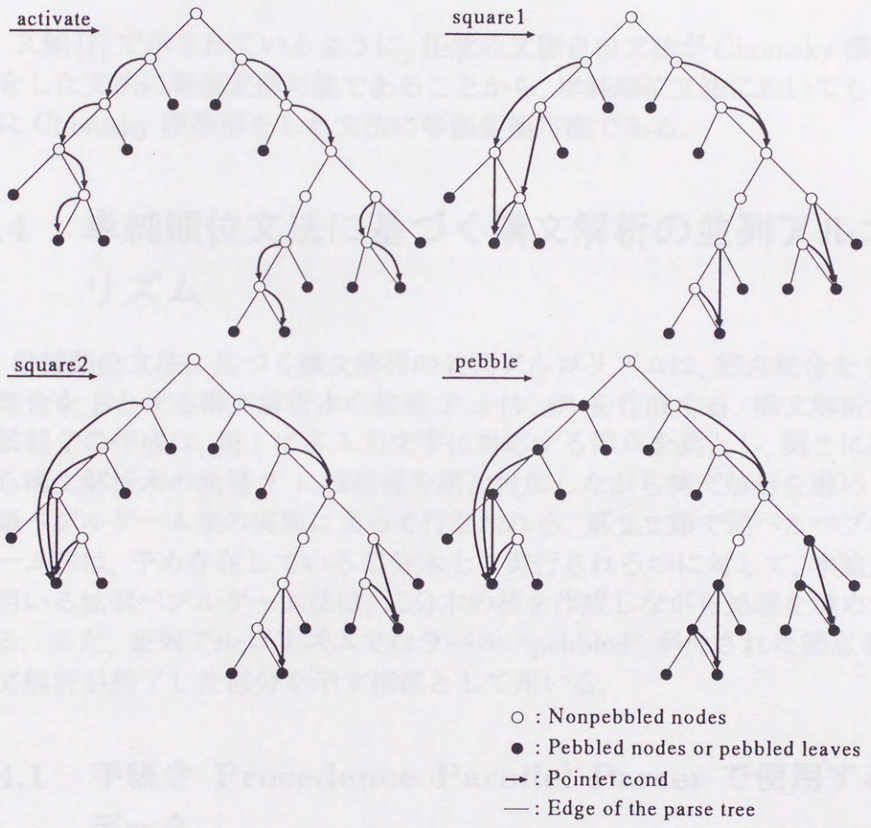


図 2.2: ペブルゲーム法の実行.
Fig. 2.2: The pebble game.

2.3 文法に対する制約

本論文で対象とする単純順位文法の生成規則には, 次の制約があるものと仮定する.

[**制約**] 与えられる文法の生成規則の形式は
Chomsky 標準形で表現されている.

文献 [7] で示されているように, 任意の文脈自由文法が Chomsky 標準形をした文法に等価変換可能であることから, 単純順位文法においても同様に Chomsky 標準形をした文法に等価変換可能である.

2.4 単純順位文法に基づく構文解析の並列アルゴリズム

単純順位文法に基づく構文解析の並列アルゴリズムは, 節点集合を V, 枝集合を Eとする構文解析木の候補 $T=(V,E)$ を作成する. 構文解析木の候補 Tの作成は, 第 1 に各入力文字に対応する節点を葉とし, 第 2 に葉から構文解析木の候補 T に枝候補を順次付加しながら構文解析を進める拡張ペブルゲーム法の実施によって行なわれる. **第 2.2 節**で述べたペブルゲーム法は, 予め存在している二分木上で実行されるのに対して, 本論文で用いる拡張ペブルゲーム法は, 二分木の枝を作成しながら処理を進めている. また, 並列アルゴリズムではラベル "pebbled" が付された節点を, 構文解析が終了した部分を示す指標として用いる.

2.4.1 手続き Precedence Parallel Parser で使用するデータ

(1) 構文解析表 $PT:\{1,2,\ldots,n\}\times\{1,2,\ldots,n\}\times N$ の要素からなる 3 次元配列である. 各要素 $PT[i,j,X]$ は, 次の値を取るように処理する.

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

(2) 構文解析木の候補 $T=(V,E)$: Tの各節点は, PTの各要素に対応する. 即ち, 節点集合 V は $\{1,2,\ldots,n\}\times\{1,2,\ldots,n\}\times N$ の部分集合, 枝集合 E は $V\times V$ の部分集合として表現される.

(3) ポインタ $cond$: $\{1,2,\ldots,n\}\times\{1,2,\ldots,n\}\times N$ の要素からなる 3 次元の配列である. $X \overset{*}{\underset{\mathrm{rm}}{\Longrightarrow}} a_i\ldots a_j$が成り立つために$Y \overset{*}{\underset{\mathrm{rm}}{\Longrightarrow}} a_k\ldots a_l$ が必要な時, 要素 $cond(i,j,X)$ は値として (k,l,Y) を取り, 「ポインタ $cond(i,j,X)$ が節点 (k,l,Y) を指す」と表す. また, 構文解析木が確定していない部分木の根を指し示す役割を持つ.

(4) 表 $Right$: $\{1,2,\ldots,n\}\times N$ の要素からなる 2 次元の配列で, ある節点に対して右側の兄弟の節点候補を表にしたものである.

(5) 表 $Right\_NT$: $\{1,2,\ldots,n\}\times N$の要素からなる 2 次元の配列で, 要素 $Right[i,Y]$ の値を求める時に探索した非終端記号を格納する表である.

(6) 表 $Left$: $\{1,2,\ldots,n\}\times N$の要素からなる 2 次元の配列で, ある節点に対して左側の兄弟の節点候補を表にしたものである.

(7) 表 $Left\_NT$: $\{1,2,\ldots,n\}\times N$の要素からなる 2 次元の配列で, $Left[j,Z]$ の値を求める時に探索した非終端記号を格納する表である.

2.4.2 手続き Precedence Parallel Parser

Step 1:(初期化)

- 構文解析木の候補 Tの節点集合 Vと枝集合 Eに対して $V := \{1,\ldots,n\}\times\{1,\ldots,n\}\times N$, $E:=\emptyset$ とする.
- 構文解析表 PT の要素全てにプロセッサを 1 個ずつ割り当てる. 各プロセッサ $P_{\{i,j,X\}}$, $1\le i\le j\le n-1$, $X\in N$ は

$$PT[i,j,X] := \text{unpebbled};$$

を実行する.

Step 2: (構文解析木の葉の初期化)

各入力文字 a_i, $1\le i\le n$ に対してプロセッサを 1 個ずつ割り当てる. 各プロセッサ $P_{\{i\}}$ は

- 各入力文字 a_iに対して, 生成規則 $A \to a_i$ によって構文解析木である二分木の葉の部分を作る.

if $A \to a_i \in P$ **then**
$PT[i, i, A] :=$ pebbled;

- 構文解析表 PTの各行の最大の位置, 及び, それに対応する非終端記号を初期化する.

if $X \to AB \in P$ **then**
begin
$Right[i, A] := i + 1$;
$Right\_NT[i, A] := B$;
end

- 構文解析表 PTの各列の最下の位置, 及び, それに対応する非終端記号を初期化する.

if $Y \to CA \in P$ **then**
begin
$Left[i, A] := i - 1$;
$Left\_NT[i, A] := C$;
end

Step 3: (activate 1)

この **Step** は右側の子供の部分木候補を求める処理で, 構文解析表 PTの要素の位置関係が図 2.3 で示される通りであり, かつ, 条件 $Y \lessdot a_{j+1}$, $Right\_NT[j, Y] \gtrdot a_{Right[j,Y]+1}$, $Y \doteq Right\_NT[j, Y]$, $X \to Y$ $Right\_NT[j, Y] \in P$, $cond(i, Right[j, Y], X) = (i, Right[j, Y], X)$ を満たす時, ポインタ $cond$ と枝を追加する. (なお, 構文解析表 PT は本来 $\{1, 2, \ldots, n\} \times \{1, 2, \ldots, n\} \times N$ の 3 次元配列であり, その各要素は, 各非終端記号に対して値 pebbled, または 値 unpebbled を取るが, ここでは 構文解析表 PT を便宜上 $\{1, 2, \ldots, n\} \times \{1, 2, \ldots, n\}$ の 2 次元の三角行列で表現し, その各要素は非終端記号のいずれかの値で表す.)

- 構文解析表 PT の各要素にプロセッサを 1 個ずつ割り当てる. 各プロセッサ $P_{\{i,j,Y\}}$, $1 \le i \le j \le n-1$, $Y \in N$ は,

if $Y \lessdot a_{j+1}$ **and**
$Right\_NT[j,Y] \gtrdot a_{Right[j,Y]+1}$ **and**
$Y \doteq Right\_NT[j,Y]$ **and**
$X \to Y \;\; Right\_NT[j,Y] \in P$ **and**
$cond(i, Right[j,Y], X) = (i, Right[j,Y], X)$ **then**
begin
$cond(i, Right[j,Y], X) := (i,j,Y)$;
$E := E \cup \{((i,j,Y),(i, Right[j,Y], X))\}$
$\cup\{((j+1, Right[j,Y], Right\_NT[j,Y]),$
$(i, Right[j,Y], X))\}$;
end

を実行する.

なお, 条件 $cond(i, Right[j,Y], X) = (i, Right[j,Y], X)$ はペブルゲーム法の non-active な条件と同じである.

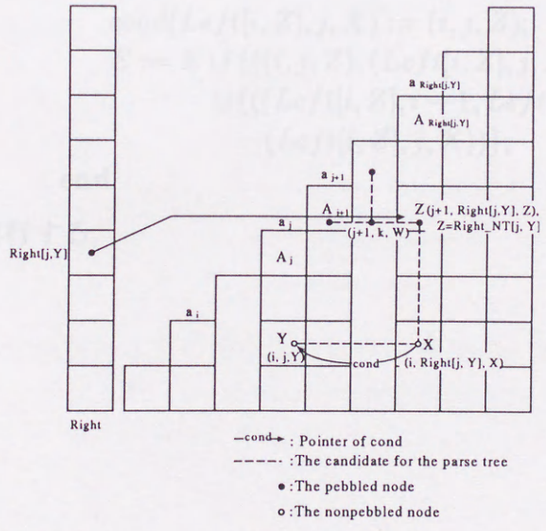


図 2.3: 構文解析表 PT の要素間の依存関係 (**Step 3 の場合**).
Fig. 2.3: The relation among elements in PT (**Case of Step 3**).

Step 4: (activate 2)

この **Step** は左側の子供の部分木候補を求める処理で, 構文解析表 PTの各要素の位置関係が図 2.4 で示される通りであり, かつ, 条件 $Z \gtrdot a_{j+1}$, $Left\_NT[i,Z] \lessdot a_i$, $Left\_NT[i,Z] \doteq Z$, $X \rightarrow Left\_NT[i,Z]\ Z \in P$,
$cond(Left[i,Z],j,X) = (Left[i,Z],j,X)$ を満たす時, ポインタ $cond$ と枝を追加する. 以下にその処理を示す.

- 構文解析表 PT の各要素にプロセッサを 1 個ずつ割り当てる. 各プロセッサ $P_{\{i,j,Z\}}$, $2 \leq i \leq j \leq n$, $Z \in N$ は,

  if $Z \gtrdot a_{j+1}$, **and**
  $Left\_NT[i,Z] \lessdot a_i$ **and**
  $Left\_NT[i,Z] \doteq Z$ **and**
  $X \rightarrow Left\_NT[i,Z]\ Z \in P$ **and**
  $cond(Left[i,Z],j,X) = (Left[i,Z],j,X)$ **then**
  begin
  $cond(Left[i,Z],j,X) := (i,j,Z)$;
  $E := E \cup \{((i,j,Z),(Left[i,Z],j,X))\}$
  $\cup\{((Left[i,Z],i-1,Left\_NT[i,Z]),$
  $(Left[i,Z],j,X))\}$;
  end

  を実行する.

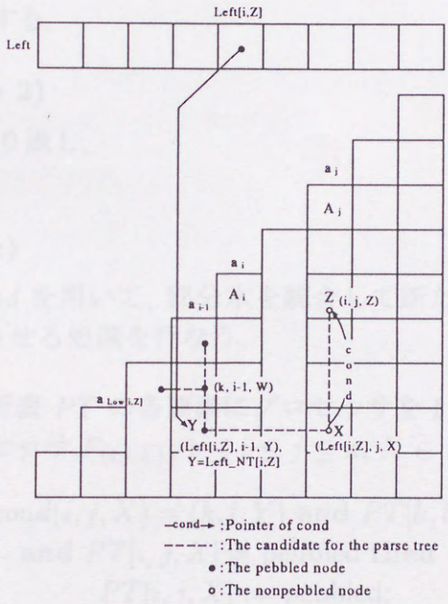


図 2.4: 構文解析表 PT の要素間の依存関係 (**Step 4 の場合**).

Fig. 2.4: The relation among elements in PT (**Case of Step 4**).

Step 5: (square 1)

確定している構文解析木の部分木を拡大するために, 統合できていない部分木の根を示すポインタ $cond$ を付け直す.

- 構文解析表 PT の各要素にプロセッサを 1 個ずつ割り当てる. 各プロセッサ $P_{\{i,j,X\}}$, $1 \leq i \leq j \leq n$, $X \in N$ は

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

を実行する.

Step 6: (square 2)

Step 5 の繰り返し.

Step 7: (pebble)

ポインタ $cond$ を用いて, 部分木を統合して新たな構文解析木の部分木を確定させる処理を行なう.

- 構文解析表 PT の各要素にプロセッサを 1 個ずつ割り当てる. 各プロセッサ $P_{\{i,j,X\}}$, $1 \leq i \leq j \leq n$, $X \in N$ は,

if $cond(i,j,X) = (k,l,Y)$ **and** $PT[k,l,Y] =$ pebbled
and $PT[i,j,X] \neq$ pebbled **then**
$PT[i,j,X] :=$ pebbled;

を実行する.

Step 8: (Right and Left の計算)

各節点の左右の節点候補を最大値と最小値を計算して求める. 処理は, $1 \leq i \leq n$ と非終端記号 $X \in N$ の各組合せごとにつぎの max, arg max, min, arg min の計算を実行する. また最大値, 最小値の計算には, 文献 [19] の並列アルゴリズムを用いる.

$$Right[i,Y] = \max_j\{j \mid PT[i,j,Z] = \text{pebbled}, \; X \to YZ \in P\};$$

$$
\begin{aligned}
&Right\_NT[i,Y] = \\
&\quad \arg\max_{Z \in N}\{j \mid PT[i,j,Z] = \text{pebbled}, \\
&\quad X \to YZ \in P\}; \\
&Left[j,Z] = \\
&\quad \min_i\{i \mid PT[i,j,Y] = \text{pebbled}, \\
&\quad X \to YZ \in P\}; \\
&Left\_NT[j,Z] = \\
&\quad \arg\min_{Y \in N}\{i \mid PT[i,j,Y] = \text{pebbled}, \\
&\quad X \to YZ \in P\};
\end{aligned}
$$

なお, $\arg\max$ は, j が最大値の時の $X \in N$ の値を返し, $\arg\min$ は, j が最小値の時の $X \in N$ の値を返す.

Step 9: (ループ終了判定)

Step 3 から **Step 8** を $\lfloor \log n \rfloor$ 回繰り返し, もし $PT[1,n,S] =$ pebbled ならば, 与えられた文法で入力文字列が生成できるので **Step 10** に進む. そうでなければ, 与えられた文法で入力文字列が生成できないので **error** を出力して異常終了する.

Step 10: (構文解析木の再構成)

- 構文解析表 PT の各要素にプロセッサを 1 個ずつ割り当てる. 各プロセッサ $P_{\{i,j,X\}}$, $1 \le i \le j \le n$, $X \in N$ は,

  if $PT[i,j,X] \neq$ pebbled **then**
  節点 (i,j,X) を節点集合 Vから取り除く;

  を実行する.

- 入力文字列の各文字 a_i, $1 \le i \le n$, に対してプロセッサを 1 個ずつ割り当てる. 各プロセッサ $P_{\{i\}}$ は

  if $A \to a_i \in P$ **then**
  begin
  $V := V \cup \{(i, i-1, a_i)\}$;
  $E := E \cup \{((i, i-1, a_i), (i, i, A))\}$;
  end

  を実行する.

2.5 構文解析の並列アルゴリズムの動作

単純順位文法 $G = (N, T, P, S)$, $N = \{S, A, B\}$, $T = \{a, b\}$, $P = \{ S \to SS, S \to AB, A \to a, B \to b \}$ と入力文字列の開始と終了を示す特殊記号\$を与えると, その単純順位文法 G の各文法記号と特殊記号\$の順位関係を表にした順位表 (表 1) が作成できる.

上の文法例に対して, 入力列を *ababab* とした時, 構文解析表 PTは **Step 1**, **Step 2** の実行後に図 2.5, **Step 3** ~ **Step 8** 実行後に図 2.6となる. 次に, **Step 9** ではループの繰り返し回数が 1 回 ($< \lfloor \log 6 \rfloor$) であるため, **Step 3** ~ **Step 8** を再び実行し 図 2.7に示す構文解析表 PT が作られる. ループ後の **Step 9** では $PT[1, 6, S]$ が 値 pebbled を取り, **Step 10** に実行が移る. **Step 10** では 構文解析表 PTで 値 pebbled となる節点と枝の両端の節点が値 pebbled となる枝だけ残すと求める構文解析木を得ることができる (図 2.8).

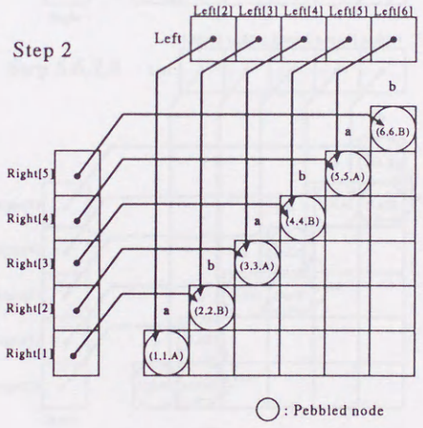


図 2.5: 並列構文解析 **Step 2** の実行.

Fig. 2.5: **Step 2** of the parallel parser.

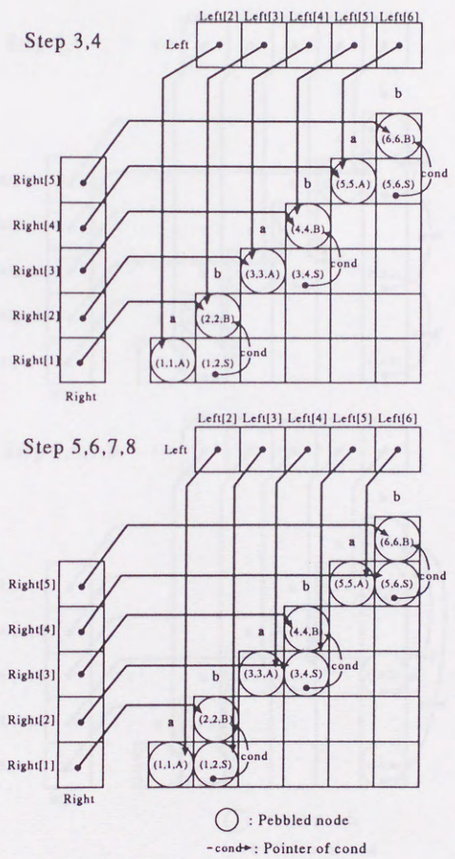


図 2.6: 並列構文解析 **Step 3 ～ 8** の実行 (その 1).
Fig. 2.6: **Steps 3** to **8** of the parallel parser (No. 1).

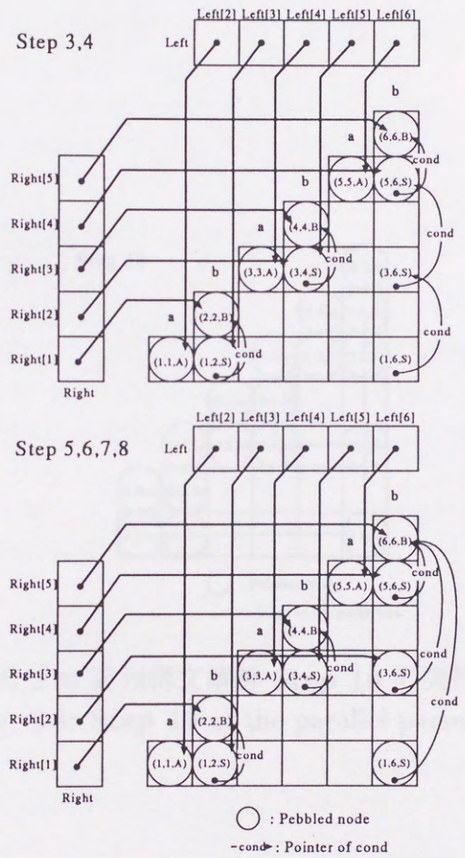


図 2.7: 並列構文解析 **Step 3 ～ 8** の実行 (その 2).
Fig. 2.7: **Steps 3** to **8** of the parallel parser (No. 2).

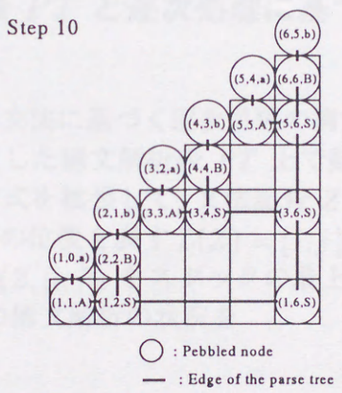


図 2.8: 並列構文解析 **Step 10** の実行.
Fig. 2.8: **Step 10** of the parallel parser.

2.6 構文解析の並列アルゴリズムの正当性と計算量

本章では, **第 4 章**で提案した単純順位文法に基づく構文解析の並列アルゴリズムの正当性と計算量を示す. まず **第 6.1 節**では, 構文解析の並列アルゴリズム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の構文解析の関係について述べる. 次に**第 6.2 節**において, **手続き Precedence Parallel Parser** を用いることにより, 逐次処理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 6.3 節**において, 提案した構文解析の並列アルゴリズムの計算量について述べる.

2.6.1 構文解析表 PT と逐次処理に基づく単純順位の構文解析

本節では, 単純順位文法に基づく逐次処理の構文解析が, 構文解析の並列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. そのために逐次の処理方式を拡張して, 文法記号 Z の他に構文解析表 PT 上の Zに対応する要素の位置を表す $L(Z) = [i, j]$ もスタックに積むことにする. すなわち, 組 $(Z, [i, j])$ をスタックの最上段に積むように拡張する. この時, 逐次処理の構文解析の状況を

(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, スタックに対する 1 回のプッシュまたはポップの操作による変化を $\vdash$で表すと, 構文解析の状況は次のように変化する. ここで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, 構文解析表 PT 上の 要素の位置 $[i, j]$ によって表すこととする.

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

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 $Y \in N$ と入力文字 a_i との順位関係を比較すると, 必ず $Y \lessdot a_i$ が成り立っている. そこで, 入力文字 a_i を 構文解析表 PT上の a_iに対応する要素の位置 $[i, i]$ 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. この時, 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 $X \to a_i$ に従って X に置き換える. 構文解

析表 PT 上の各要素の位置 $L(Y)$ を含めて状況の変化を形式的に記述すると, 次のようになる.

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

但し, X と a_{i+1} の間の順位関係は議論しないため, 記号 ? とする.

(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 $(Y,[i,j])$ が新たに積まれた場合を考える. この時, 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 $Y \lessdot a_{j+1}$ が成り立っている. 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組のうち上位 2 つの組中の非終端記号が 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, 構文解析表 PT 上の要素の位置 $[j+1,k]$ が共に積まれる. この時, 非終端記号 Z と次の入力文字 a_{k+1} との間には, 順位関係 $Z \lessdot a_{k+1}$ または $Z \gtrdot a_{k+1}$ のいずれかが成り立つ. $Z \doteq a_{k+1}$は, 本論文の制約である Chomsky 標準形をした単純順位文法により成立しない.

(2.a) 順位関係 $Z \gtrdot a_{k+1}$ が成り立っている場合は, スタックの最上段の Z とその下の Y との間に順位関係 $\doteq$ が成り立つ. 従って, 生成規則 $X \to YZ$ を適用して, YZ を X で置換える. 更に, 構文解析表 PT 上の要素の位置 $[i,j]$ と 構文解析表 PT 上の要素の位置 $[j+1,k]$ から, 構文解析表 PT 上での動作位置を位置 $[i,k]$ に置換える. 状況変化を形式的に記述すると, 次のようになる.

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

但し, X と a_{k+1} の間の順位関係は議論しないため, 記号 ? とする.

(2.b) 順位関係 $Z \lessdot a_{k+1}$ が成り立っている場合は, まず a_{k+1} を次の入力として受付ける.

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

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 ? とする.

上で述べた逐次処理の構文解析において, スタックの最上段における構文解析表 PT上の要素の位置を $L_1 = [i, j]$ とし, 次に新しく積む構文解析表 PT上の要素の位置を $L_2 = [k, l]$ とする時, 積む動作を 「PT上の位置 L_1 から位置 L_2への移動」と呼ぶ. この移動には, 下記の (1), (2.a), (2.b) の 3 つの場合がある.

(1) の場合: 生成規則 $X \to a_i$によって a_iを Xに変換する動作と構文解析表 PT上の a_iから構文解析表 PT上の位置 $[i, i]$ への移動が対応する. なお, a_iは構文解析表 PTの要素ではないが, 便宜上 構文解析表 PT上にあるとして取り扱う.

(2.a) の場合: 生成規則 $X \to YZ$によって YZを Xに変換する動作と構文解析表 PT上の位置 $[j+1, k]$ から構文解析表 PT上の位置 $[i, k]$ への移動とが対応する.

(2.b) の場合: 構文解析表 PT上の位置 $[j+1, k]$ から構文解析表 PT上の a_{k+1}への移動とが対応する. なお, (1) と同様に a_{k+1}は構文解析表 PTの要素ではないが, 便宜上構文解析表 PT上にあるとして取り扱う.

つまり, 構文解析表 PT上での一連の移動は, 構文解析表 PT上の要素の位置 $[1, 1]$ から開始され, 節点 $(1, n, S)$ に対応する 構文解析表 PT上の要素の位置 $[1, n]$ で終了する. それらの一連の移動は単純順位文法に基づく逐次処理の構文解析の動作と等価である. また, 構文解析表 PT上での動作の軌跡は, 構文解析木の葉を持つ枝を左から右へ順にたどり, 最も右の葉を持つ枝に達したら, そこから根に向かってたどっていくことに対応する. 以上の 構文解析表 PT 上の要素の位置の一連の移動の軌跡を図 2.9 に示す.

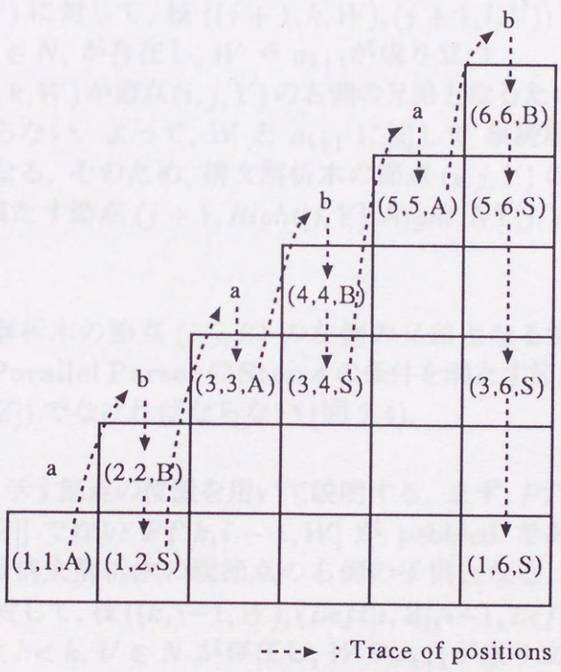


図 2.9: 構文解析表 PT 上での移動の軌跡.
Fig. 2.9: The trace of positions on PT.

2.6.2 手続き Precedence Parallel Parser における性質

[定理 1] 構文解析木の節点 (i,j,Y) の右側の兄弟となる節点は, **手続き Precedence Parallel Parser** の **Step 3** の条件を満たす節点 $(j+1, Right[j,Y], Right\_NT[j,Y])$ でなければならない (図 2.3).

(証明) 図 2.3に示す節点の位置を用いて説明する. まず, $PT[j+1, Right[j,Y], Right\_NT[j,Y]]$ でない $PT[j+1,k,W]$ が pebbled である場合, 節点 $(j+1,k,W)$ は構文解析木の親節点の左側の子供となる. つまり, 節点 $(j+1,k,W)$ に対して, 枝 $((j+1,k,W),(j+1,l,U)) \in E$, $k < l < Right[j,Y]$, $U \in N$, が存在し, $W \lessdot a_{k+1}$が成り立つ.

一方, $(j+1,k,W)$ が節点 (i,j,Y) の右側の兄弟となるためには, $W \gtrdot a_{k+1}$ でなければならない. よって, W と a_{k+1} に関して 単純順位文法の定義を満たさなくなる. そのため, 構文解析木の節点 (i,j,Y) の兄弟となる節点は, 条件を満たす節点 $(j+1, Right[j,Y], Right\_NT[j,Y])$ でなければならない. □

[定理 2] 構文解析木の節点 (i,j,Z) の左側の兄弟となる節点は, **手続き Precedence Parallel Parser** の **Step 4** の条件を満たす節点 $(Left[i,Z], i-1, Left\_NT[i,Z])$ でなければならない (図 2.4).

(証明) 図 2.4に示す節点の位置を用いて説明する. まず, $PT[Left[i,Z], i-1, Left\_NT[i,Z]]$ でない $PT[k,i-1,W]$ が pebbled である場合, 節点 $(k,i-1,W)$ は構文解析木の親節点の右側の子供となる. つまり, 節点 $(k,i-1,W)$ に対して, 枝 $((k,i-1,W),(Left[i,Z],i-1,Left\_NT[i,Z])) \in E$, $Left[i,Z] < l < k$, $U \in N$ が存在し, $W \gtrdot a_{k+1}$が成り立つ.

一方, $(k,i-1,W), W \in N$が構文解析木の左側の節点になるためには, $W \lessdot a_i$ でなければならない. よって, Wと a_iに関して単純順位文法の定義を満たさなくなる. そのため, 構文解析木の節点 $(k,i-1,W)$ に対応する右側の節点は, 条件を満たす節点 $(Left[i,Z], i-1, Left\_NT[i,Z])$ でなければならない. □

[定理 3] $cond(i,j,X) = (k,l,Y)$, $Y \overset{*}{\underset{\mathrm{rm}}{\Longrightarrow}} a_k \ldots a_l$ ならば, $X \overset{*}{\underset{\mathrm{rm}}{\Longrightarrow}} a_i \ldots a_j$である.

(証明)
(場合 1) $cond(i,j,X) = (k,l,Y)$ である時に $(k-i)+(j-l) = 1$ が成り

立つ場合.

構文解析木は二分木を合成して作られるので, 節点 (k,l,Y) の兄弟を節点 (m,n,Z) とすると, $m=n$ が成り立つ. なぜなら, 例えば節点 (m,n,Z) を親節点 (i,j,X) の右側の子供とすると $k=i$, $j=n$, $m=l+1$ が成り立ち, (場合 1) の条件は $j-l=1$ と変形されるので, $m=n$ が成り立つ. また, 同様に節点 (m,n,Z) を親節点 (i,j,X) の左側の子供としても $m=n$ が成り立ち, $Z \overset{*}{\underset{\mathrm{rm}}{\Longrightarrow}} a_m$を満たす. よって **Step 3,4** の処理から $X \overset{*}{\underset{\mathrm{rm}}{\Longrightarrow}} a_i \ldots a_j$が成り立つ.

(場合 2) $cond(i,j,X)=(k,l,Y)$ である時に $(k-i)+(j-l)>1$ が成り立つ場合.

Step 2 の処理から節点 (k,l,Y) から枝集合 Eの枝をたどって (i,j,X) に到達でき, ポインタ $cond$ は次の関係を成り立たせるように必ず 1 度は設定されている.

$$\begin{aligned} &cond(i,j,X)=(i_1,j_1,X_1),\\ &cond(i_1,j_1,X_1)=(i_2,j_2,X_2),\\ &\qquad\ldots,\\ &cond(i_{m-1},j_{m-1},X_{m-1})=(k,l,Y), \end{aligned}$$

但し, $(i_1-i)+(j-j_1)=1$, $(i_2-i_1)+(j_1-j_2)=1$, ..., $(k-i_{m-1})+(j_{m-1}-l)=1$ が成り立ち, $X,Y,X_1,X_2,\ldots,X_{m-1}\in N$である.

(場合 1) の $(k-1)+(j-l)=1$ の場合から各ポインタ $cond$ については [定理 3] が成り立ち, $Y \overset{*}{\underset{\mathrm{rm}}{\Longrightarrow}} a_k \ldots a_l$ ならば, $X \overset{*}{\underset{\mathrm{rm}}{\Longrightarrow}} a_{i_{m-1}} \ldots a_{j_{m-1}}$ であり, $X \overset{*}{\underset{\mathrm{rm}}{\Longrightarrow}} a_{i_{m-1}} \ldots a_{j_{m-1}}$ ならば, $X \overset{*}{\underset{\mathrm{rm}}{\Longrightarrow}} a_{i_{m-2}} \ldots a_{j_{m-2}}$ であり, ..., $X \overset{*}{\underset{\mathrm{rm}}{\Longrightarrow}} a_{i_1} \ldots a_{j_1}$ ならば, $X \overset{*}{\underset{\mathrm{rm}}{\Longrightarrow}} a_i \ldots a_j$ である.

よって, **Step 3,4,5,6** でポインタ $cond$ の指す先を変えても, [定理 3] が成り立つ. □

[定理 4] 構文解析の並列アルゴリズム $PT[i,j,X]$ = pebbled ならば, $X \overset{*}{\underset{\mathrm{rm}}{\Longrightarrow}} a_i \ldots a_j$である.

(証明) $a_i \ldots a_j$の導出の段階に関する帰納法によって, 証明する.

(i. 初期段階) **手続き Precedence Parallel Parser** の **Step 2** では i 番目の入力文字 a_iに対して $A \to a_i \in P$ ならば, $PT[i,i,A]$ に値 pebbled を設定している. 一方, **第 2 章**の単純順位文法の**定義** (4) の**逆方向決定性**と Chomsky 標準形の制約から入力文字 a_iに対応する生成規則は $A \to a_i$のみである. よって, $PT[i,j,X]$ = pebbled ならば, $X \overset{*}{\underset{\mathrm{rm}}{\Longrightarrow}} a_i \ldots a_j$である.

(ii.a. 帰納段階) $PT[k,l,Y]$ = pebbled ならば, $Y \overset{*}{\underset{\mathrm{rm}}{\Longrightarrow}} a_k \ldots a_l$と仮定する. この時, **Step 7** から [定理 3] の条件が成り立つ.
(ii.b. 帰納段階) (ii.a 帰納段階) の仮定と [定理 3] より, $cond(i,j,X) = (k,l,Y)$, $Y \overset{*}{\underset{\mathrm{rm}}{\Longrightarrow}} a_k \ldots a_l$であるので, $X \overset{*}{\underset{\mathrm{rm}}{\Longrightarrow}} a_i \ldots a_j$である.
一方, **Step 7** では $PT[k,l,Y]$ = pebbled ならば, $PT[i,j,X]$:= pebbled としている. よって, [定理 4] が成り立つ. □

なお, **Step 10** において構文解析表 PTで値 pebbled を取る節点と両端の節点が値 pebbled を取る枝のみを残している. 言い替えれば, 構文解析木の節点候補と枝候補を残している. なぜなら, 単純順位文法の構文解析は, 最右導出の構文解析木を 1 つだけ求めているので, 各節点は左の子供の節点と右の子供の節点の組を必ず 1 組だけ持つためである. よって, **Step 10** で求めた構文解析木の候補者 Tが求める構文解析木である.

2.6.3 構文解析の並列アルゴリズムの計算量

本論文で提案した構文解析の並列アルゴリズムでは, **手続き Precedence Parallel Parser** の **Step 2, 8, 9** を除く **Step 1,3,4,5,6,7,10** プロセッサを構文解析表 PTの各要素に割り当てている. 入力文字列の長さを n, 非終端記号の数を $|N|$ とする時, 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 $O(n^2)(= O(|N|n^2))$ で, 処理時間は定数時間である. 以下に **Step 2,8,9** について述べる.

- **Step 2** では, 入力文字列ごとにプロセッサを割り当てているので, $O(n)$ 個のプロセッサ数が必要で, 処理時間は定数時間である.
- **Step 8** では, 各最大値と最小値の計算は, 構文解析表 PTの第 1 要素または第 2 要素が等しい $|N|n$ 個の要素に対して実行している. CREW P-RAM 上で最大値や最小値の計算は入力個数 kに対して $O(\log k)$ 時間と k個のプロセッサを必要とする [19] ことから, 要素数 $|N|n$ に対して, $O(\log n)(= O(\log(|N|n)))$ 時間と $O(n)(= O(|N|n))$ 個のプロセッサ数を必要とする. また, 表 $Right$, $Left$ の要素ごとに最大値または最小値を求めていることから, $O(\log n)$ 時間と $O(n^2)$ 個のプロセッサ数が必要となる.
- **Step 9** では, **Step 3** ~ **Step 8** のループの終了判定処理を行うのでプロセッサを 1 つ必要とし, 時間は定数時間である.

また, **Step 3** ～ **Step 8** は $\lfloor \log n \rfloor$ 回ループし, **Step 8** で $O(\log n)$ 時間かかることから, **Step 3** ～ **Step 9** では $O(\log^2 n)$ 時間必要となる.

以上のことから, 本論文で示した構文解析の並列アルゴリズムは, $O(\log^2 n)$ 時間と $O(n^2)$ 個のプロセッサ数で実現される.

2.7 むすび

本論文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た. 本論文のアルゴリズムの特徴は, 構文解析木のある節点に対して, もう一方の兄弟に当たる節点候補者を見つけるのに, 最大値と最小値を求める並列アルゴリズムを用いれば良い点である.

また, 本論文の構文解析の並列アルゴリズムは二次元に配置したデータに対して 1 個ずつプロセッサを割り当てるため, プロセッサ間の連結がメッシュ構造を持つ並列コンピュータへの適用が容易であり, かつ高速に実行可能である. 今後は実際の並列コンピュータへのインプリメントを行ない, 本論文の構文解析の並列アルゴリズムの実用上における有効性を検証する予定である.

第 3 章

LR 文法に対する並列構文解析

3.1 はじめに

並列計算機の一理論モデルである CREW P-RAM [19] (Concurrent Read Exclusive Write P-RAM) 上における一般の文脈自由文法 [7, 19] とその幾つかの部分クラスに対する構文解析の並列アルゴリズムは既にいくつか知られている. そのうち第 2 章では, 単純順位文法に対して $O(\log^2 n)$ 時間と $O(n^2)$ 個のプロセッサで並列構文解析が実行可能 [28] であることを明らかにした.

また, 決定性の文脈自由言語を認識する並列アルゴリズムは $O(\log^2 n)$ 時間と $O(n^{2+\epsilon})$ 個, $0 < \varepsilon < 1$, のプロセッサが必要 [26] なことが分かっているが, 文献 [19] で述べているように, 認識問題で得られた情報から構文解析木を得るには少なくとも $O(n^3)$ 個のプロセッサが必要となり, 文献 [26] のアルゴリズムを利用する限り $O(\log^2 n)$ 時間と $O(n^3)$ 個のプロセッサが必要となる. そこで本章では, CREW P-RAM 上で, LR文法に対して $O(\log n)$ 時間と $O(n^3)$ 個のプロセッサで構文解析を行う並列アルゴリズムを提案する.

なお, アルゴリズムの説明などの形式的な部分は文献 [19] に準拠し, 文脈自由文法 $G = (N, T, P, S)$, N: 非終端記号の集合, T: 終端記号の集合, P: 生成規則の集合, S: 開始記号 と入力文字列 $a_1 a_2 \ldots a_n a_{n+1}$, $a_1, a_2, \ldots, a_n \in T$ が与えられるものとする. また, 構文解析実行の便宜上, 入力文字列の終りを示す特殊記号\$ ($\notin N \cup T$) を用意し, $a_{n+1} =$ \$として扱う.

3.2 LR 文法

LR文法を [27] に準拠して定義する.

- 文脈自由文法 $G=(N,T,P,S)$ の 最右導出
  $S \overset{*}{\underset{rm}{\Longrightarrow}} \mu A\nu \underset{rm}{\Longrightarrow} \mu\alpha\nu$, $A \in N$, $\alpha,\mu,\nu \in T^*$と $S \overset{*}{\underset{rm}{\Longrightarrow}} \xi B\nu_1 \underset{rm}{\Longrightarrow} \xi\beta\nu_1 = \mu\alpha\nu_2$, $B \in N$, α, β, ξ, ν_1, $\nu_2 \in T^*$に対して,

$$head_k(\nu) = head_k(\nu_2)$$

  ならば, $\xi B\nu_1 = \mu A\nu_2$ となる G を $LR(k)$ 文法と定義する. 但し, $\overset{*}{\underset{rm}{\Longrightarrow}}$ は 0 回以上の最右導出を示し, $\underset{rm}{\Longrightarrow}$ は 1 回の最右導出を示す. また, $head_k(\nu)$ はνの先頭から k個の文字列を表す.

例えば, $G_{ex}=(N,T,P,S)$, $N=\{S,A,B,C,D\}$, $T=\{a,b,c\}$, $P=\{S \to DC,\ S \to AC,\ D \to AA,\ A \to BA,\ A \to a,\ B \to b,\ C \to c\}$ は $LR(0)$ 文法の例である.

なお, 本章では $LR(k)$ 文法 (kは 0 以上の整数) を LR文法と総称する.

3.2.1 前提

本章が対象としている LR文法には, 次の制限を設けている.

(1) LR文法から生成される LR言語は, ε(空語) を含まない.

(2) LR文法の生成規則は Chomsky 標準形 [7] で与えられている. Chomsky 標準形は, 非終端記号 A,B,C と 終端記号 a に対して生成規則 $A \to BC$, $A \to a$ の形式しか持たない.

文献 [7] で示されているように, 任意の文脈自由文法が Chomsky 標準形の文法に等価変換可能であることから, ε導出を含まない任意の LR文法は Chomsky 標準形の文法に等価変換可能である.

3.3 並列アルゴリズムの概要

文献 [28] で示した単純順位文法に対する並列構文解析アルゴリズムでは, 前処理が必要なく文法記号間の優先順位を構文解析実行中に計算する必要がある. それに対して, 本章で示す並列アルゴリズムは, 文法から逐

次処理のLR構文解析でも作成される*LR*状態遷移図(*LR* Automaton, 以降 **LRA** と略記) と, 新たに作成する**終端記号入力状態遷移表** (Terminal symbol State Transition Table, 以降 **TSTT** と略記) と**非終端記号入力状態遷移表** (Nonterminal symbol State Transition Table, 以降 **NSTT** と略記) の3つの表を構文解析前に求め, その情報をLR構文解析用に拡張した**ペブルゲーム法** [19] に適用することで実現している.

ここで表 **TSTT** と **NSTT** は, それぞれ終端記号, 非終端記号によって **LRA** のある状態から次の新たな入力を受け付けるまでの状態への遷移をまとめた表で, 遷移には空スタック [25] を利用する. 例えば, *LR*文法例 G_{ex}の **LRA**(図 3.1, 定義は**第 3.1 節**) の状態 s_3から A によって状態 r_4 に移る. 状態 r_4では, 還元する文法規則 $D \rightarrow AA$ の右側の AA を用いて **LRA** を逆に遷移して s_0に移り, 空のスタックに Dを積む. 状態 s_0では入力としてスタックの最上段にある Dによって状態 s_1に移り, スタックから最上段にある Dを降ろす. スタックは Dを降ろすと空になるので, ここで遷移を終了する. 以上の一連の遷移に対して, 「A による状態 s_3 から s_1への遷移がある」とする.

一方, 並列アルゴリズムは, **Step 1** として入力文字列の各入力文字ごとに可能な状態遷移を求めることができ, それらをつなぎあわせて**非決定性状態遷移リスト** (Nondeterministic State Transition List, 以降 **NDSL** と略記) を作成する. 次に **Step 2** では **NDSL** の要素の組合せによって*LR*構文解析表 (*LR* Parse Table, 以降 **LRPT** と略記) を作り, **Step 3** では, *LRPT*の要素の間に**第 3.7 節**に示す性質があれば, その要素間に有向辺を付加し, その有向グラフに*LR*構文解析用に拡張したペブルゲーム法を用いて, 構文解析木から終端記号の直接導出部を除いた二分木部分 (以降, 構文解析木の主要部) を取りだす. そして最後に **Step 4** で構文解析木を作成する.

以下の節では, 表の定義を**第 3.4 節**で, また, 並列アルゴリズムの **Step 1,2,3,4** をそれぞれ**第 3.5,3.6,3.8,3.9 節**で述べる.

3.4 表の定義

並列アルゴリズムの**前処理**に当たる3つの表 **LRA**, **TSTT**, **NSTT** の定義について述べる.

3.4.1 LR状態遷移図 (LRA)

LR構文解析の動作を示す **LRA** は, $\mathbf{LRA} = (Q, \Sigma, s_0, s_{acc}, \lambda, \$)$,

(1) 状態集合 $Q = Q_{pop} \cup Q_{push}$,

(2) ポップ動作を行う状態の集合 $Q_{pop} = \{r_1, r_2, \ldots, r_{|Q_{pop}|}\}$,

(3) プッシュ動作を行う状態の集合 $Q_{push} = \{s_0, s_1, \ldots, s_{|Q_{push}-2|}, s_{acc}\}$,

(4) 入力記号の集合 Σ. 但し, $\Sigma = N \cup T$,

(5) 開始状態 $s_0 \in Q_{push}$,

(6) 受理状態 $s_{acc} \in Q_{push}$,

(7) 状態遷移関数 δ: $Q \times (N \cup T) \times (T^* \cup \{\$\}^*) \longrightarrow Q$,

(8) 入力文字列の終了を示す特殊記号 $\$ \notin N \cup T$ と定義する.

以下に先読み文字列数 kの場合の **LRA** の作成方法を以下に示す.

[作成 1] ダミー記号 S'を用意する. 次に $S' \to S$を Pに追加して $LR(k)$ 項 $[S' \to \cdot S, \$]$ を状態 $s_0 (\in Q_{push})$ の要素とする.

[作成 2] $LR(k)$ 項 $[\lambda \to \mu_1 \cdot \alpha\mu_2, \beta]$ が状態 $s_i \in Q_{push}$に存在するなら, 生成規則$\alpha\zeta \to \eta \in P$ の形式を持つ生成規則に対応するすべての $LR(k)$ 項 $[\alpha\zeta \to \cdot\eta]$ を状態 s_iに追加する ($\alpha \in (T \cup N)$, $\lambda, \beta \in (T \cup N)^+$, $\mu_1, \mu_2, \zeta, \eta \in (T \cup N)^*$).

[作成 3] $LR(k)$ 項 $[\lambda \to \mu_1 \cdot \alpha\mu_2, \beta]$ が状態 $s_i \in Q$ に存在するなら, 新たに状態 s_jを用意し, 状態 s_jが $LR(k)$ 項 $[\lambda \to \mu_1\alpha \cdot \mu_2, \beta]$ から成るとする.

[作成 4] 状態 $s_i \in Q$ に $LR(k)$ 項 $[\lambda \to \mu_1 \cdot \alpha\mu_2, \beta]$, $\alpha \in (T \cup N)$ があり, 状態 $s_j \in Q$ に $LR(k)$ 項 $[\lambda \to \mu_1\alpha \cdot \mu_2, \beta]$ があるなら, αによる状態 s_iから状態 s_jへのシフト動作の遷移を追加する.

[作成 5] 状態 $r_i \in Q_{pop}$に $LR(k)$ 項 $[\lambda \to \mu\cdot, \beta]$ があるなら, 状態 r_iにおいて, レデュース動作の遷移を追加する.

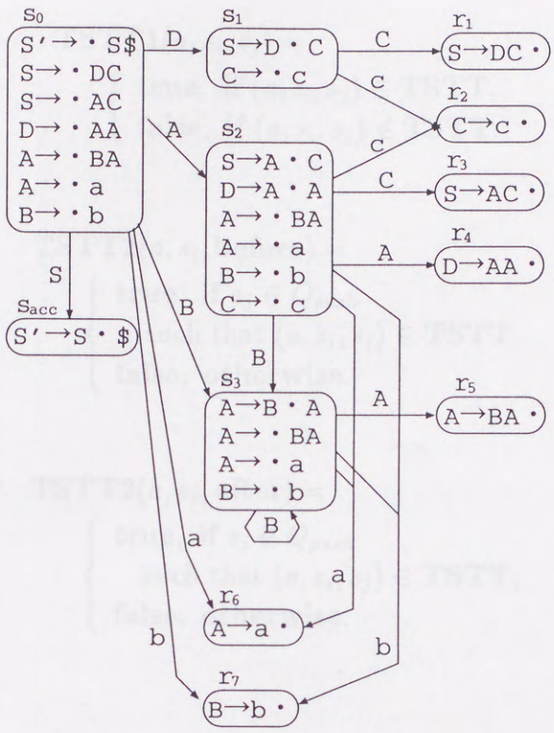


図 3.1: 文法 G_{ex}に対する **LRA**
Fig. 3.1: **LRA** corresponding to G_{ex}.

3.4.2 終端記号入力状態遷移表 (TSTT)

$s_i \overset{LR_G}{\underset{\alpha}{\Longrightarrow}} s_j$は, **LRA** 上で状態 $s_i(\in Q_{push})$ からα $(\in T \cup N)$ を入力として空スタックの LR構文解析と同じ動作を実行する時, LR構文解析が次の入力を受け付ける状態 $s_j(\in Q_{push})$ に遷移することを示す.

これに対して, **TSTT** を以下のように定義する.

$$\mathbf{TSTT} = \{(a, s_i, s_j) |\ s_i \overset{LR_G}{\underset{a}{\Longrightarrow}} s_j,\ s_i, s_j \in Q_{push}, a \in T\}.$$

以上の定義に対して, 表 3.1に **TSTT** の例を示す.

また, **TSTT** は以下に示す 2 つの関数 **TSTT1**,**TSTT2** によって間接的に参照される.

$$\mathbf{TSTT1}(a, s_i, s_j) =$$
$$\begin{cases} \textbf{true, if } (a, s_i, s_j) \in \mathbf{TSTT}, \\ \textbf{false, if } (a, s_i, s_j) \notin \mathbf{TSTT}. \end{cases}$$

$$\mathbf{TSTT2}(a, s_i, \textbf{before}) =$$
$$\begin{cases} \textbf{true, if } s_j \in Q_{push} \\ \quad \text{such that } (a, s_i, s_j) \in \mathbf{TSTT}, \\ \textbf{false, otherwise.} \end{cases}$$

$$\mathbf{TSTT2}(a, s_j, \textbf{after}) =$$
$$\begin{cases} \textbf{true, if } s_i \in Q_{push} \\ \quad \text{such that } (a, s_i, s_j) \in \mathbf{TSTT}, \\ \textbf{false, otherwise.} \end{cases}$$

表 3.1: **TSTT** の例
Table 3.1: An example of **TSTT**.

(Terminal symbol, First state, Final state)
(a, s_0, s_3),(a, s_2, s_1),(a, s_3, s_1), (a, s_3, s_2)
(b, s_0, s_3),(b, s_3, s_3)
(c, s_1, s_{acc}),(c, s_2, s_{acc})

3.4.3 非終端記号入力状態遷移表 (NSTT)

NSTT は, **LRA** の状態 s_iから非終端記号 A によって遷移を開始させ, 次の入力を受け付ける状態 s_j に遷移する場合に, $(A, s_i, s_j, action) \in$ **NSTT**, $action \in$ {**shift, reduce**} とする表で, 以下に定義する. なお, *action* が **shift** である要素は, $(s_i, A, \varepsilon) \to s_j \in \lambda$, $s_i, s_j \in Q_{push}$ であり, 一方, **reduce** 要素は $(s_i, A, \varepsilon) \to s_j \notin \lambda$, $s_i, s_j \in Q_{push}$であることを示す.

$$\mathbf{NSTT} = \left\{ \begin{array}{l|l} (A, s_i, s_j, & s_i \overset{LR_G}{\underset{A}{\Longrightarrow}} s_j, \\ \textbf{shift}) & s_i, s_j \in Q_{push}, A \in N, \\ & \lambda(s_i, A, \beta) = s_j, \\ & \beta \in T^* \cup \{\$\}^* \end{array} \right\} \cup$$

$$\left\{ \begin{array}{l|l} (A, s_i, s_j, & s_i \overset{LR_G}{\underset{A}{\Longrightarrow}} s_j, \\ \textbf{reduce}) & s_i, s_j \in Q_{push}, A \in N, \\ & \lambda(s_i, A, \beta) \neq s_j, \\ & \lambda(s_i, A, \beta) = r_{i+1}, \\ & r_{i+1} \in Q_{pop}, \beta \in T^* \cup \{\$\}^* \end{array} \right\}.$$

また, LR文法 G_{ex}に対する **NSTT** の例を表 3.2に示す.

表 3.2: **NSTT** の例
Table 3.2: An example of **NSTT**.

(Nonterminal symbol, First state, Final state, Action)
$(S, s_0, s_{acc}, \textbf{shift})$, $(A, s_0, s_2, \textbf{shift})$, $(B, s_0, s_3, \textbf{shift})$, $(D, s_0, s_1, \textbf{shift})$
$(C, s_1, s_{acc}, \textbf{reduce})$
$(A, s_2, s_1, \textbf{reduce})$, $(B, s_2, s_3, \textbf{shift})$, $(C, s_2, s_{acc}, \textbf{reduce})$
$(A, s_3, s_2, \textbf{reduce})$, $(B, s_3, s_3, \textbf{shift})$

3.5 非決定性状態遷移リスト (NDSL) の作成 (Step 1)

NDSL は入力文字列の各文字ごとに **TSTT** から計算できる **LRA** 上における状態遷移前の状態と遷移後の状態の候補をつなぎあわせて作る. **NDSL** は有向グラフ $\mathbf{NDSL} = (V_1, E_1)$ として表す. 以下に, その作成の

ステップを示す.

Step 1.1 (NDSL の 節点集合 V_1の構成):

(S.1) $V_1 := \phi$(空集合);

(S.2) **for all** $1 < i \leq n$, $s_i \in Q_{push}$ **in parallel do**
begin

(S.3) **if TSTT2**$(a_i, s_i,$ **after) and**
TSTT2$(a_{i+1}, s_i,$ **before) then**

(S.4) $V_1 := V_1 \cup \{(i, s_i)\}$;
end

(S.5) $V_1 := V_1 \cup \{(1, s_0)\}$; $V_1 := V_1 \cup \{(n+1, s_{acc})\}$;

Step 1.2 (NDSL の辺集合 E_1の構成):

(S.1) $E_1 := \phi$(空集合);

(S.2) **for all** $1 \leq i \leq n$, $s_i, s_{i+1} \in Q_{push}$
in parallel do
begin

(S.3) **if** $(i, s_i), (i+1, s_{i+1}) \in V_1$
and TSTT1(a_i, s_i, s_{i+1}) **then**

(S.4) $E_1 := E_1 \cup \{((i, s_i), (i+1, s_{i+1}))\}$;
end. □

なお, **Step 1** の処理過程と **NDSL** の例を図 3.2に示す.

3.6 LR構文解析表 (LRPT) の作成 (Step 2)

NDSL の 2 つの節点間の有向辺による到達可能性を答える関数 Rc を以下のように定義する.

$$Rc((i, s_i), (j, s_j)) = \begin{cases} \textbf{true}, & \textbf{if NDSL 上において節点 } (i, s_i) \text{ から} \\ & \text{節点 } (j, s_j) \text{ へ有向辺を辿って到達可能.} \\ \textbf{false}, & \textbf{otherwise.} \end{cases}$$

節点間の到達可能性の判定は **NDSL** に対して 1 度実行しておけばよく, その到達可能性の判定にはブール行列積問題の並列アルゴリズム [15, 16] を適用すればよい.

関数 Rc を利用して LR構文解析表を以下のように定義する. なお, **LRPT** は誤解の生ずるおそれのない限り, それに対応する有向グラフの節点集合と同一視する.

$$\mathbf{LRPT} = \left\{ (i, j, s_i, s_j, A, action) \;\middle|\; \begin{array}{l} Rc((i,s_i),(j,s_j)) = \mathbf{true}, \\ (A, s_i, s_j, action) \in \mathbf{NSTT}, \\ 1 \le i < j \le n+1, \\ s_i, s_j \in Q_{push}, A \in N, \\ action \in \{\mathbf{shift}, \mathbf{reduce}\} \end{array} \right\}.$$

以上の定義に基づき, 以下のステップを実行する.

(S.1) **NDSL** 中の各要素対に対して, 最短路を求める並列アルゴリズム [19] を適用する.

(S.2) **for all** $1 \le i < j \le n+1$, $s_i, s_j \in Q_{push}$, $A \in N$ **in parallel do**
begin

(S.3) **if** $Rc((i,s_i),(j,s_j))$= **true** **and** $(A, s_i, s_j, \mathbf{shift}) \in$ **NSTT then**

(S.4) **LRPT** := **LRPT** $\cup \{(i, j, s_i, s_j, A, \mathbf{shift})\}$;

(S.5) **if** $Rc((i,s_i),(j,s_j))$= **true** **and** $(A, s_i, s_j, \mathbf{reduce}) \in$ **NSTT then**

(S.6) **LRPT** := **LRPT** $\cup \{(i, j, s_i, s_j, A, \mathbf{reduce})\}$;

end. □

例として, 図 3.2に示す **NDSL** に対して作成される **LRPT** を 図 3.3 に示す. 例えば (3,4,s_3,s_3,B, **shift**) $\in$ **LRPT** は, 図 3.3の 3 行 4 列の **shift** の要素である.

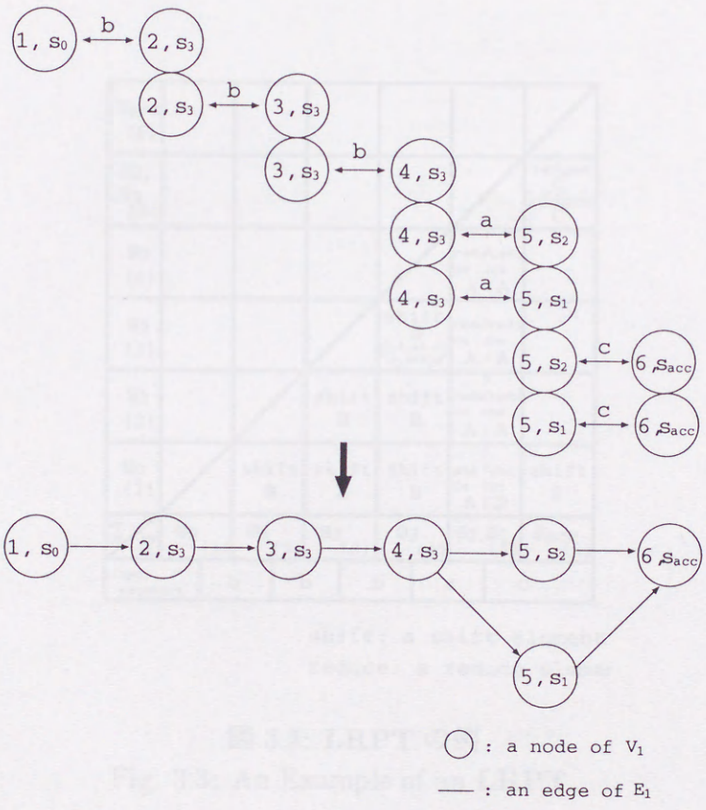


図 3.2: **Step 1** の処理過程と **NDSL** の例

Fig. 3.2: A process of **Step 1** and an example of an **NDSL**.

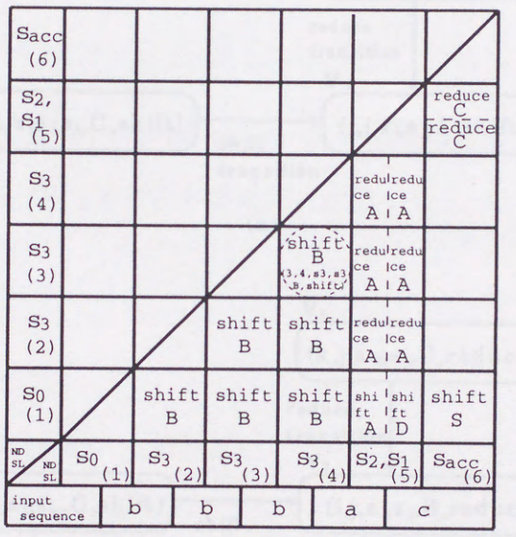
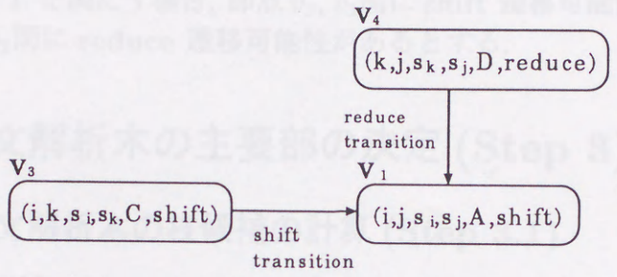


図 3.3: **LRPT** の例

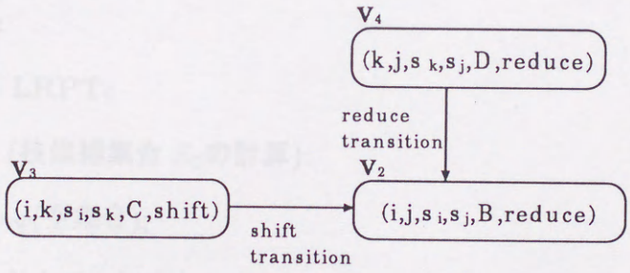
Fig. 3.3: An Example of an **LRPT**.

3.7 LRPT の要素間の性質

LRPT の要素間には, 次の 2 つの性質が成り立つ. ここでは説明のため, **LRPT** の 5 つの要素を節点 v_1=(i,j,s_i,s_j,A,**shift**), v_2=(i,j,s_i,s_j,B,**reduce**), v_3=(i,l,s_i,s_l,C,**shift**), v_4=(k,j,s_k,s_j,D,**reduce**) とする (図 3.4参照).



(a)



(b)

図 3.4: **LRPT** 上での要素の配置

Fig. 3.4: Arrangement of elements in an **LRPT**.

[**性質 1**] 図 3.4(a) のように配置される節点 v_1, v_3, v_4に対して 条件 $A \to CD \in P, A, C, D \in N, (A, s_i, s_j, \textbf{shift}), (C, s_i, s_l, \textbf{shift}), (D, s_k, s_j, \textbf{reduce}) \in \textbf{NSTT}$ を満たす場合, 節点 v_3, v_1間に **shift 遷移可能性**, 及び, 節点 v_4, v_1間に **reduce 遷移可能性**があるとする.

[**性質 2**] 図 3.4(b) のように配置される節点 v_2, v_3, v_4に対して 条件 $B \to CD \in P, B, C, D \in N, (B, s_i, s_j, \textbf{reduce}), (C, s_i, s_k, \textbf{shift}), (D, s_k, s_j, \textbf{reduce}) \in \textbf{NSTT}$ を満たす場合, 節点 v_3, v_2間に **shift 遷移可能性**, 及び, 節点 v_4, v_2間に **reduce 遷移可能性**があるとする.

3.8 構文解析木の主要部の決定 (Step 3)

3.8.1 構文解析木の枝候補の計算 (Step 3.1)

第 3.7 節で述べた **LRPT** の要素間に成り立つ 2 つの性質に対応して枝の候補を示す有向グラフ G_2を作成する. なお, 本節の節点は図 3.4の位置と同じである.

Step 3.1.1:

(S.1) $V_2 :=$ **LRPT**;

Step 3.1.2 (枝候補集合 E_2の計算):

(S.1) $E_2 := \phi$(空集合);

(S.2) **for all** $1 \le i, k \le j \le n+1$, $s_i, s_j, s_k \in Q_{push}$
$A, B, C, D, E \in N$ **in parallel do**
begin
v_1:=$(i, j, s_i, s_j, A, \textbf{shift})$;
v_2:=$(i, j, s_i, s_j, B, \textbf{reduce})$;
v_3:=$(i, k, s_i, s_k, C, \textbf{shift})$;
v_4:=$(k, j, s_k, s_k, D, \textbf{reduce})$;

(S.3) **if** v_1, v_3, v_4 が [**性質 1**] を満たす **then**
$E_2 := E_2 \cup \{(v_3, v_1)\} \cup \{(v_4, v_1)\}$;
else begin

(S.4) **if** v_2, v_3, v_4 が [**性質 2**] を満たす **then**
$E_2 := E_2 \cup \{(v_3, v_2)\} \cup \{(v_4, v_2)\}$;

end
end. □

有向グラフ G_2 は**第 3.7 節**の性質によって, 親と 2 つの子供の節点を持つ部分二分木を重ねて構成される. また, 図 3.3に示した **LRPT** に対して, **Step 3.1** を実行した結果を図 3.5に示す.

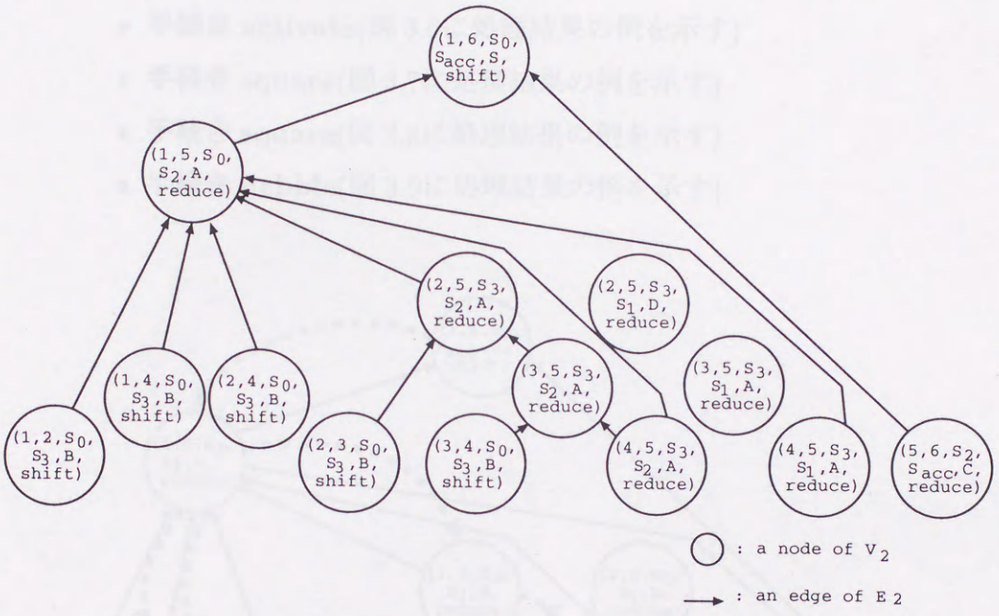


図 3.5: **並列構文解析の Step 3.1 の実行**
Fig. 3.5: **Step 3.1** of the parallel parser.

3.8.2 到達不可能な節点の削除 (Step 3.2)

部分二分木を重ねて構成された有向グラフ G_2 に対して, 木を構成しながらラベルをつけるペブルゲーム法に拡張した並列アルゴリズムを適用する. また, 並列アルゴリズムでは, ペブルゲーム法のポインタ cond に対応する有向辺集合 E_3 を作る.

Step 3.2.1: $V_3 := V_2$;

Step 3.2.2: 初期化として葉に当たる節点にラベル
"pebbled" を付し, 各節点から出発する有向辺を自分自身にする.

- $v = (i, i+1, s_i, s_{i+1}, A, action) \in V_3$, $1 \leq i \leq n$, $A \in N$, $action \in$ {**shift,reduce**} を満たす節点 vに対してラベル "pebbled" を付す.
- $v \in V_3$に対して, $E_3 := E_3 \cup \{(v, v)\}$;

Step 3.2.3: 以下に示す **手続き** を $\lfloor \log n \rfloor$ 回 繰り返す.

- **手続き activate**(図 3.6に処理結果の例を示す)
- **手続き square**(図 3.7に処理結果の例を示す)
- **手続き square**(図 3.8に処理結果の例を示す)
- **手続き pebble**(図 3.9に処理結果の例を示す)

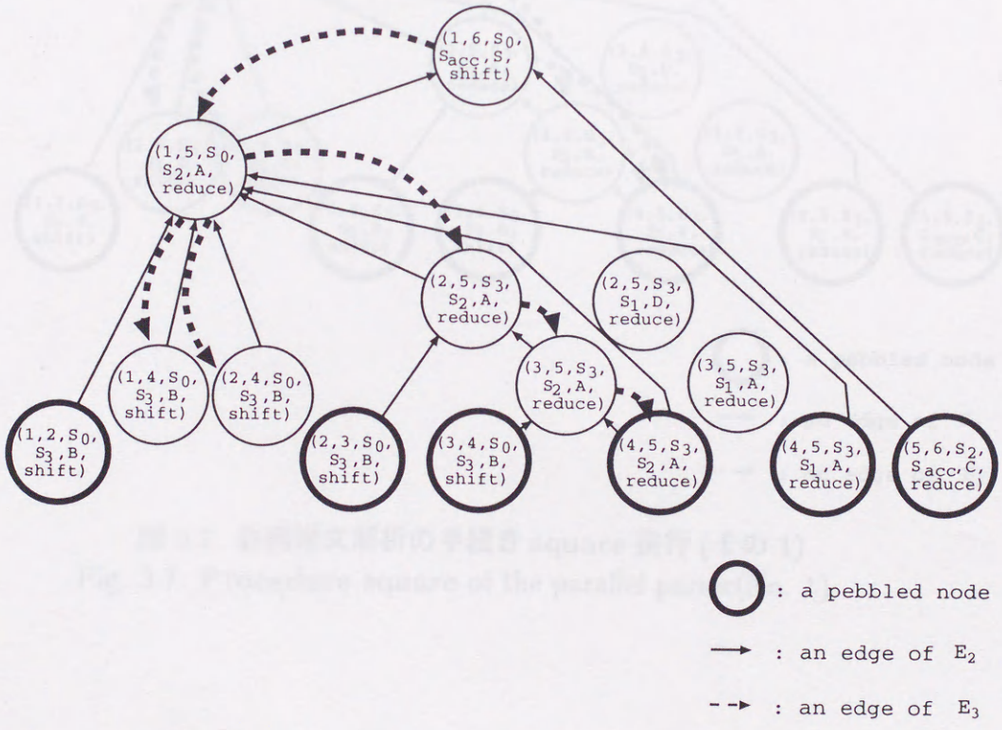


図 3.6: **並列構文解析の手続き activate 実行**
Fig. 3.6: **Procedure activate** of the parallel parser.

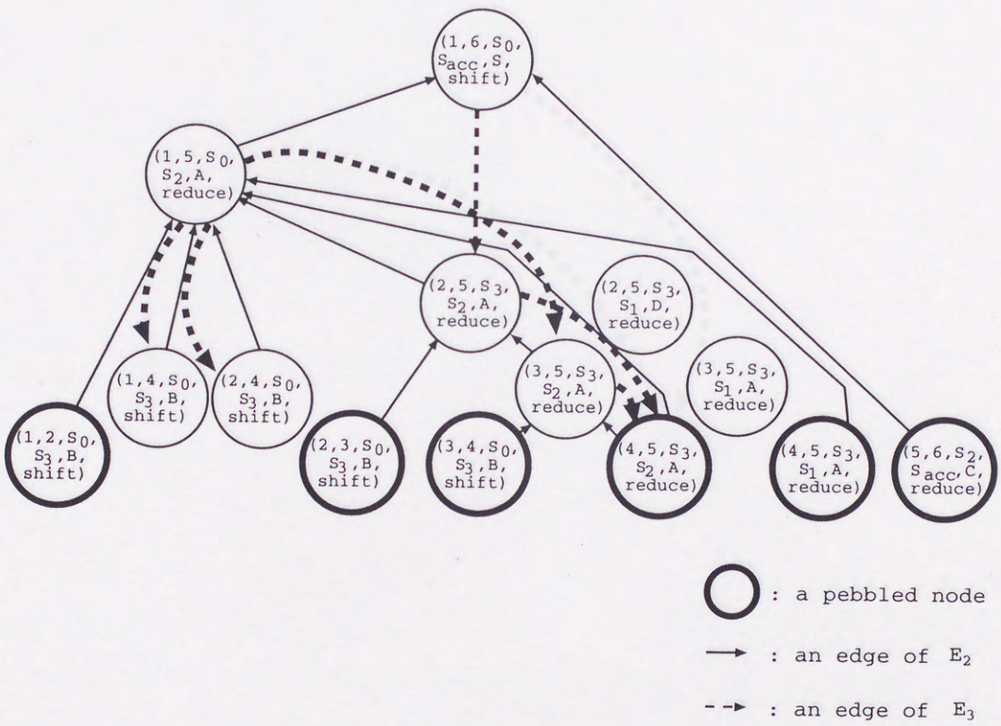


図 3.7: 並列構文解析の手続き square 実行 (その 1)

Fig. 3.7: **Procedure square** of the parallel parser(No. 1).

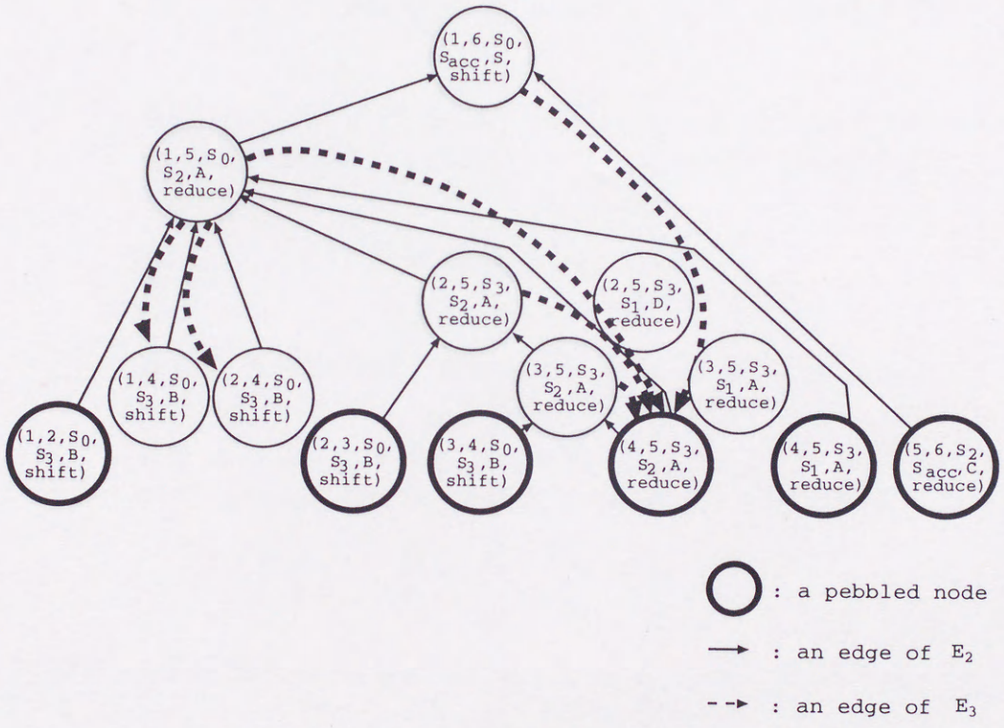


図 3.8: 並列構文解析の手続き square 実行 (その 2)

Fig. 3.8: **Procedure square** of the parallel parser(No. 2).

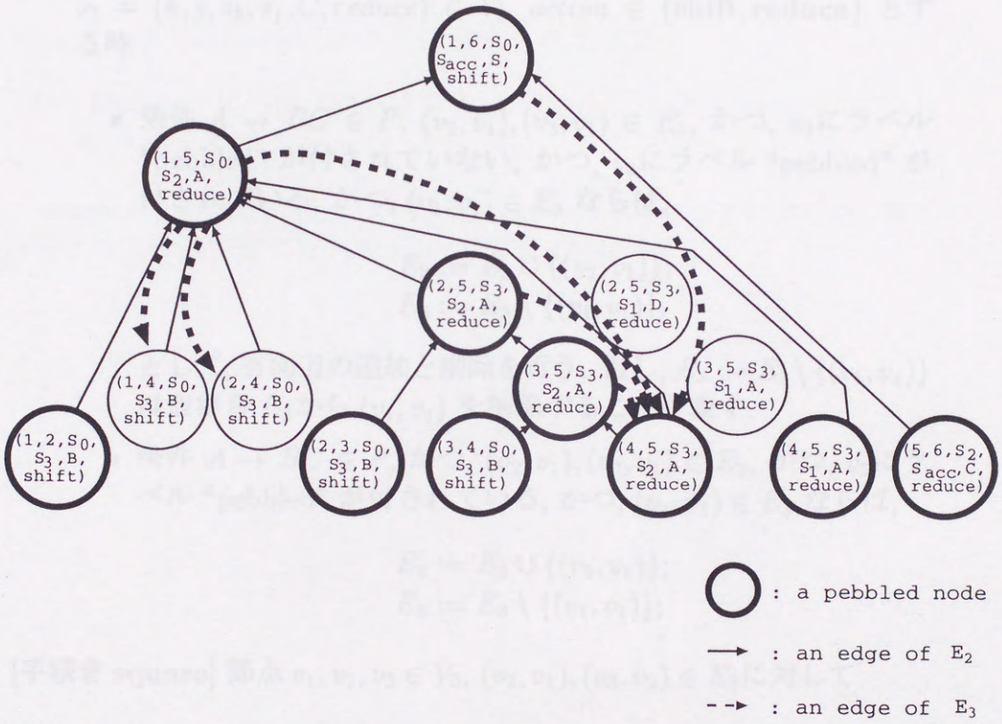


図 3.9: 並列構文解析の手続き pebble 実行

Fig. 3.9: **Procedure pebble** of the parallel parser.

Step 3.2.4: 節点 (1,$n+1$,s_0,s_{acc},S,**shift**) $\notin V_3$, または, その節点にラベル "pebbled" が付されてなければ, 与えられた文法で入力文字列が生成できないので, **error** を出力し停止する. ラベル "pebbled" が付されていれば生成でき, 次のステップに進む. □

以下に各**手続き activate, square, pebble** について述べる.

[**手続き activate**] G_2の節点 $v_1 = (i, j, s_i, s_j, A, action)$, $v_2 = (i, k, s_i, s_k, B, shift)$, $v_3 = (k, j, s_k, s_j, C, reduce) \in V_2$, $action \in \{\textbf{shift}, \textbf{reduce}\}$ とする時

- 条件 $A \to BC \in P$, $(v_2, v_1), (v_3, v_1) \in E_2$, かつ, v_2にラベル "pebbled" が付されていない, かつ, v_3にラベル "pebbled" が付されている, かつ, $(v_1, v_1) \in E_3$ ならば,

$$E_3 := E_3 \cup \{(v_2, v_1)\};$$
$$E_3 := E_3 \setminus \{(v_1, v_1)\};$$

  として, 有向辺の追加と削除を行う. 但し, $E_3 := E_3 \setminus \{(v_1, v_1)\}$ は辺集合 E_3から (v_1, v_1) を削除することを表す.

- 条件 $A \to BC \in P$, かつ, $(v_2, v_1), (v_3, v_1) \in E_2$, かつ, v_2にラベル "pebbled" が付されている, かつ, $(v_1, v_1) \in E_3$ ならば,

$$E_3 := E_3 \cup \{(v_3, v_1)\};$$
$$E_3 := E_3 \setminus \{(v_1, v_1)\};$$

[**手続き square**] 節点 $v_1, v_2, v_3 \in V_3$, $(v_2, v_1), (v_3, v_2) \in E_3$に対して

$$E_3 := E_3 \cup \{(v_3, v_1)\};$$
$$E_3 := E_3 \setminus \{(v_2, v_1)\};$$

[**手続き pebble**] 節点 $v_1, v_2 \in V_3$, v_2にラベル "pebbled" が付されている, かつ, $(v_2, v_1) \in E_3$であるなら, 節点 v_1にラベル "pebbled" を付す.

3.8.3 構文解析木の主要部の作成 (Step 3.3)

オイラーツアー法 [19] を適用して, 構文解析木の主要部を取り出す. 以下に, そのステップを示す.

Step 3.3.1 (pebbled されていない節点を削除):

(S.1) 節点 $v \in V_3$のラベル "pebbled" が付されている節点を V_4に移す.

(S.2) $v_1, v_2 \in V_4$, $(v_2, v_1) \in E_2$を満たすなら, $E_4 := E_4 \cup \{(v_2, v_1)\}$ とする.

Step 3.3.2 (構文解析木の二分木部分の選択):

(S.1) $(1, n+1, s_0, s_{acc}, S, \mathbf{shift}) \in V_4$ を根とし, **オイラーツアー法**の並列アルゴリズム [19] を用いて先駆け順 (preorder) に番号を振る.

(S.2) (S.1) で番号が振られた節点集合を V_5とする.

(S.3) 節点集合 V_5の節点間を結ぶ E_4の辺集合を E_5とし, 有向グラフ G_5を作成する. □

Step 3.2 の最後に得られた図 3.9に対して, **Step 3.3** の実行後, 構文解析木の主要部を表す有向グラフ G_5を図 3.10に示す.

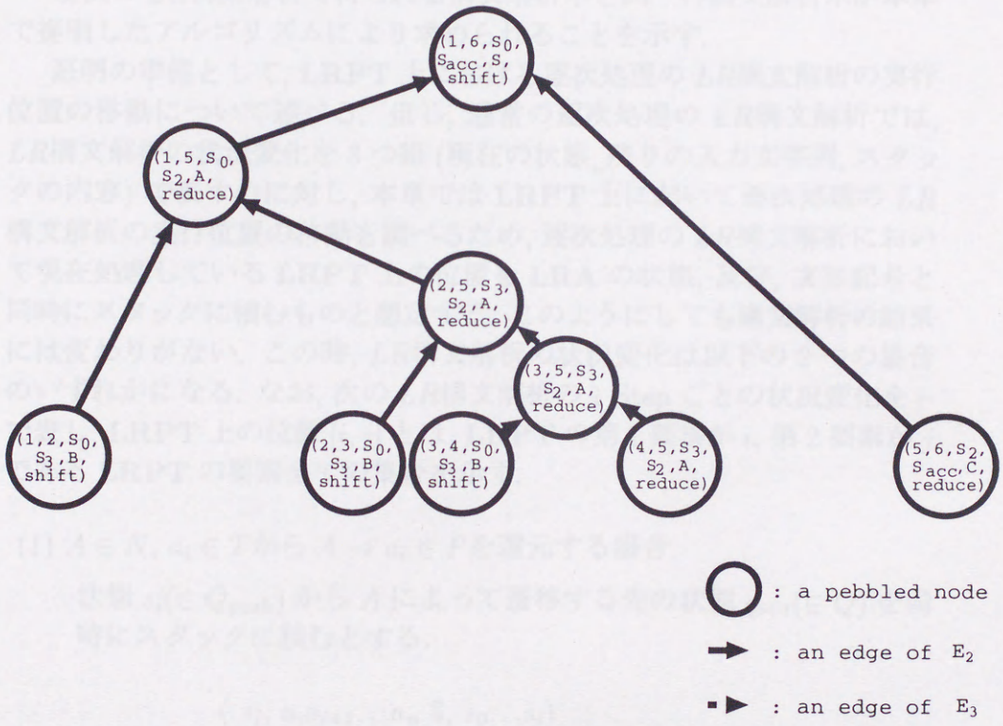


図 3.10: **並列構文解析の Step 3.3 の実行**

Fig. 3.10: **Step 3.3** of the parallel parser.

3.9 構文解析木の生成 (Step 4)

Step 3 で求めた構文解析木の主要部 G_5に各終端記号から非終端記号への辺をそれぞれ追加して, 構文解析木を生成する.

3.10 アルゴリズムの正当性

逐次の LR構文解析で得られる構文解析木と同一の構文解析木が本章で提唱したアルゴリズムにより求められることを示す.

証明の準備として, **LRPT** 上における逐次処理の LR構文解析の実行位置の移動について述べる. 但し, 通常の逐次処理の LR構文解析では, LR構文解析の状況変化を 3 つ組 (現在の状態, 残りの入力文字列, スタックの内容) で表すのに対し, 本章では **LRPT** 上において逐次処理の LR構文解析の実行位置の移動を調べるため, 逐次処理の LR構文解析において現在処理している **LRPT** 上の位置を **LRA** の状態, 及び, 文法記号と同時にスタックに積むものと想定する. このようにしても構文解析の結果には変わりがない. この時, LR構文解析の状況変化は以下の 2 つの場合のいずれかになる. なお, 次の LR構文解析の 1 Step ごとの状況変化を$\vdash$で表し, **LRPT** 上の位置 $[i,j]$ とは, **LRPT** の第 1 要素が i, 第 2 要素が j である **LRPT** の要素全ての集合を示す.

(1) $A \in N$, $a_i \in T$から $A \to a_i \in P$を還元する場合.

状態 $s_i(\in Q_{push})$ から A によって遷移する先の状態 $q_{i+1}(\in Q)$ を同時にスタックに積むとする.

$$
\begin{aligned}
&(\ s_i,\ a_i a_{i+1} \ldots a_n\ \$,\ s_0 \ldots s_i) \\
\vdash\ &(r_i,\ a_{i+1}\ a_{i+2}\ \ldots\ a_n\ \$,\ s_0\ \ldots\ s_i\ a_i\ r_i) \\
\vdash\ &(q_{i+1},\ a_{i+1}\ a_{i+2}\ \ldots a_n\ \$,\ s_0\ \ldots\ s_i A[i,i+1] q_{i+1}).
\end{aligned}
$$

(2) $A, B, C \in N$ から $A \to BC \in P$ を還元する場合.

状態 $s_i(\in Q_{push})$ から A によって遷移する先の状態 $q_{i+1}(\in Q)$ を同時にスタックに積むとする.

$$
\begin{aligned}
&(r_i,\ a_{i+1} a_{i+2} \ldots a_n\ \$,\ s_0 \ldots s_i B[i,k] s_k\ C[k,j] r_j) \\
\vdash\ &(q_{i+1}, a_{i+1} a_{i+2} \ldots a_n\ \$,\ s_0 \ldots s_i A[i,j]\ q_j).
\end{aligned}
$$

3.10.1 第 3.7 節の性質の正当性

(定理 1) **第 3.7 節の [性質 1,2] の reduce 遷移可能性**は, 逐次の LR構文解析処理の生成規則還元動作と同等である.

(証明) 生成規則 $A \to BC$, $A, B, C \in N$ に対して LR構文解析で還元動作をする場合は, 次のように逐次処理の LR構文解析が進む.

$$(r_i, a_{i+1}a_{i+2}\ldots a_n, s_0\ldots B[i,k]s_kC[k,j]r_j)$$
$$\vdash (r_{i+1}, a_{i+1}a_{i+2}\ldots a_n, s_0\ldots A[i,j]s_j).$$

スタックに Bと同時に積まれている **LRPT** 上の位置 $[i,k]$ と Cと同時に積まれている **LRPT** 上の位置 $[k,j]$ に対応し, A と同時に **LRPT** 上の位置 $[i,j]$ を積んでいる. これは **LRPT** 上の位置 $[k,j]$ から位置 $[i,j]$ への移動である. また, **LRPT** 上の位置 $[k,j]$ から位置 $[i,j]$ への移動には, 位置 $[i,k]$ を通過することが必要であるため, a_iの入力前に **LRPT** 上の位置 $[i,k]$ を積む動作が行なわれ, $(i,k,s_i,s_k,B,\textbf{shift}) \in$ **LRPT** となる.

よって, 位置 $[k,j]$ から $[i,j]$ への移動は, 逐次の LR 構文解析処理の還元動作と見なすことができ, **[性質 1] の reduce 遷移可能性**と同じと見なすことができる.

また, 同様にして **[性質 2]** についても成り立つ. □

以上の **reduce 遷移可能性**の正当性から, **LRPT** の要素を節点としたグラフに有向辺を付加することが可能となる.

3.10.2 節点にラベル "pebbled" が付されていることと生成可能性との同等性

(定理 2) **Step 3.2(第 3.8.2 節)** において, 節点 $(1, n+1, s_0, s_{acc}, \textbf{shift})$ にラベル "pebbled" が付されていることと入力文字列が LR文法によって生成されることとは同等である.

(証明) **Step 3.2(第 3.8.2 節)** において, 節点 $v = (i, j, s_i, s_j, A, \text{action}), A \in N$, action $\in$ {**shift,reduce**} にラベル "pebbled" が付されていることと, LR構文解析が状態 s_iから始まり, 入力文字列 $a_i\ldots a_{j-1}$を処理した後, 状態が s_jに達し文法記号 A を生成することとが同等であることを以下の帰納法で明らかにする.

(1) 空スタックの LR構文解析が入力文字 a_iによって状態 s_iから s_jへ遷移し生成規則 $B \to a_i \in P$ から Bを生成することと, 葉に当たる節

点 $(i, i+1, s_i, s_{i+1}, B, \text{action})$ にラベル "pebbled" が付されていることとは同等である.

(2) 構文解析木の部分二分木 $BT = (\{v_1, v_2, v_3\}, \{(v_2, v_1), (v_3, v_1)\}$ があるとする. 子供の節点に当たる節点 v_2, v_3にラベル "pebbled" が付されている場合, その BTの親の節点 v_1にラベル "pebbled" を付す.

ここで, 節点をそれぞれ$v_1 = (i, j, s_i, s_j, A, \textbf{shift})$, $v_2 = (i, k, s_i, s_k, B, \textbf{shift})$, $v_3 = (k, j, s_l, s_j, C, \textbf{reduce})$ とする. この場合, v_2にラベル "pebbled" が付されていることから, LR構文解析が状態 s_iから始まり, 入力文字列 $a_i \ldots a_{k-1}$を処理した後, 状態が s_kに達し文法記号 Bを生成する. また, v_3にラベル "pebbled" が付されていることから, LR構文解析が状態 s_kから始まり, 入力文字列 $a_k \ldots a_{j-1}$を処理した後, 状態が s_jに達し文法記号 Cを生成する. もし, 生成規則 $A \to BC \in P$ が存在するならば, LR構文解析が状態 s_iから始まり, 入力文字列 $a_i \ldots a_{j-1}$を処理した後, 状態 s_jに状態が達し文法記号 A を生成することは明らかである.

以上の帰納法から, 節点 $(1, n+1, s_0, s_{acc}, S, \textbf{shift})$ にラベル "pebbled" が付されていることと逐次処理の LR構文解析が入力文字列 $a_1 \ldots a_n$を受理可能であることとは, 同等である □

3.10.3 正当な構文解析木が得られること

(定理 3) 本節で示しているアルゴリズムによって構成された構文解析木は正しい構文解析木である.

(証明) **Step 3.2(第 3.8.2 節)** で構文解析木の根に当たる節点 $(1, n+1, s_0, s_{acc}, S, \textbf{shift})$ にラベル "pebbled" が付されていない場合は, (定理 2) から入力文字列が LR文法では生成できないので, 構文解析木を作成せずに **error** を出力している.

一方, 入力列が LR文法から生成できる場合は, ラベル "pebbled" が付されている節点の子供が, 1 組のみであることから示される.

ここで, ラベル "pebbled" が付されている節点の子供が, 1 組のみであることを, 図 3.11 を用いて説明する (なお, 図 3.11の要素の位置は LRPT に対応させている). まず, 節点 $v_1 = (i, j, s_i, s_j, A, \textbf{shift})$, $v_2 = (i, k, s_i, s_k, B, \textbf{shift})$, $v_3 = (k, j, s_k, s_j, C, \textbf{reduce})$, $v_4 = (i, l, s_i, s_l, D, \textbf{shift})$, $v_5 = (l, j, s_l, s_j, E, \textbf{reduce})$, $v_6 = (k, n, s_k, s_n, F, \textbf{shift})$, $v_7 = (m, l, s_m, s_l, G, \textbf{reduce})$,

$v_8 = (k, k+1, s_j, s_{j+1}, a_j, \textbf{shift})$, $v_9 = (k, k+1, s_j, s_{j+1}, a_j, \textbf{reduce})$ であるとする.

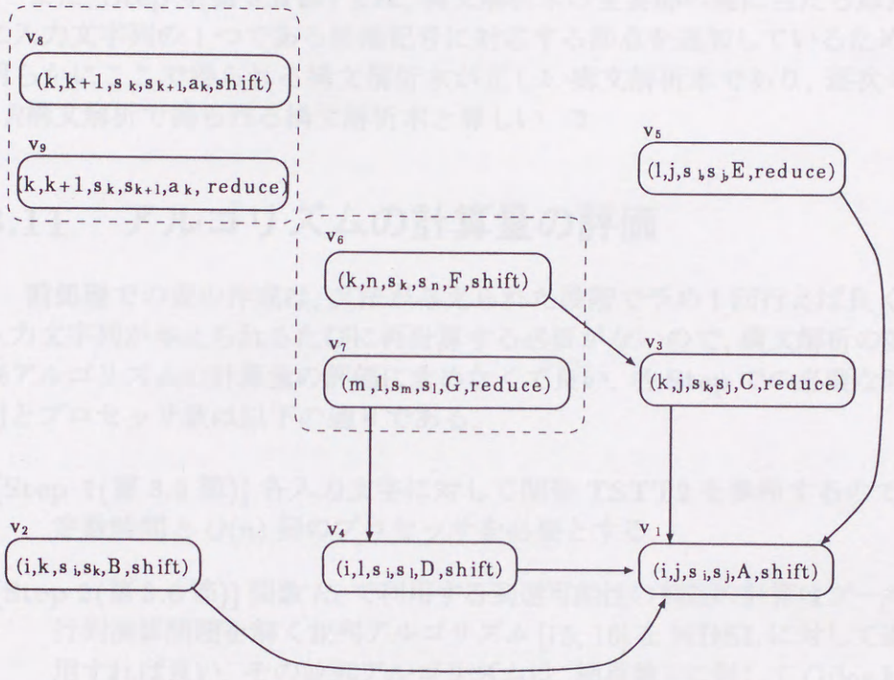


図 3.11: 構文解析の節点の子供の候補

Fig. 3.11: The candidate of sons of a node in parse tree .

- $k = m$, $l = n$ の場合.

  この場合は表 **NSTT** の作成法から同一の状態から同一の状態への遷移があるときは, **shift** 遷移のみを取ることから, 節点 v_6 と v_7 の両方にラベル "pebbled" はつけられず, 節点 v_6 にラベル "pebbled" がつけられる. よって, 節点 v_1 の子供は v_2 と v_3 の組である.

- $k < m$, $l < n$ の場合.

  この場合, 節点 v_2 と v_6 にラベル "pebbled" をつけるためには, 節点 v_8 と v_9 が存在しなければならない. しかし前の場合と同様に, 表 **NSTT** の作成法から, 同一の状態から同一の状態への遷移があるときは, **shift** 遷移のみを取ることから, 節点 v_8 と v_9 の両方にラベル

"pebbled" はつけられず, 節点 v_8にラベル "pebbled" がつけられる. よって, 節点 v_1の子供は v_2 と v_3の組である.

また, **Step 4(第 3.9 節)** では, 構文解析木の主要部の葉に当たる節点に入力文字列の 1 つである終端記号に対応する節点を追加しているため, 明らかにここで得られる構文解析木が正しい構文解析木であり, 逐次の LR構文解析で得られる構文解析木と等しい. □

3.11 アルゴリズムの計算量の評価

前処理での表の作成は, 文法が与えられた段階で予め 1 回行えば良く, 入力文字列が与えられるたびに再計算する必要がないので, 構文解析の並列アルゴリズムの計算量の評価に含めなくて良い. 各 **Step** での必要な時間とプロセッサ数は以下の通りである.

[Step 1(第 3.5 節)] 各入力文字に対して関数 **TSTT2** を参照するので, 定数時間と $O(n)$ 個のプロセッサを必要とする.

[Step 2(第 3.6 節)] 関数 Rc で利用する到達可能性の判定の計算はブール行列演算問題を解く並列アルゴリズム [15, 16] を **NDSL** に対して適用すれば良い. その並列アルゴリズムは, 節点数 kに対して $O(\log k)$ 時間と $O(k^{2.376})$ 個のプロセッサを必要とする [?]. **NDSL** の節点の数は $O(n)$ 個であるから, 到達可能性の判定の計算には $O(\log n)$ 時間と $O(n^{2.376})$ 個のプロセッサを必要とする.

また, 関数 Rc 以降の **LRPT** の作成アルゴリズムは, **NDSL** の状態を組合わせて作成する手続きであり, 定数時間と $O(n^2)$ 個のプロセッサを必要とする.

[Step 3.1(第 3.8.1 節)] (S.2) では, 変数として i, j, kを用いている. 一方, $|Q_{push}|$ は定数と見なせるので, CRCW P-RAM[19] (Concurrent Read Concurrent Write P-RAM) 上で定数時間と $O(n^3)$ 個のプロセッサが必要となる. そこで CRCW P-RAM の動作を CREW P-PRAM でシミュレートする [19] と $O(\log n)$ 時間と $O(n^3)$ 個のプロセッサを必要とする.

[Step 3.2(第 3.8.2 節)] **手続き square** では, 節点 $v_1, v_2, v_3 \in V_3$, (v_1, v_2), $(v_2, v_3) \in E_3$となる時, $E_3 := E_3 \cup \{(v_1, v_3)\}$ としている.

節点集合 E_3から $v_1 = (i, k, s_i, s_k, A, action)$, $v_2 = (i, j, s_i, s_j, B, \textbf{shift})$, $v_3 = (j, k, s_j, s_k, C, \textbf{reduce}) \in V_2$, $action \in$ {**shift**,**reduce**}を選択していることから, i, j, kを選択するのと同じであるので, 使用するプロセッサ数は $O(n^3)$ 個である.

また, **手続き activate**, **square**, **square**, **pebble** は $\lfloor \log n \rfloor$ 回繰り返すので, $O(\log n)$ 時間必要となる.

[Step 3.3(第 3.8.3 節)] **Step 3.3** の**オイラーツアー法** [19] の実行では, 節点数 $O(n^2)$ に対して $O(\log n)$ 時間と $O(n^2)$ 個のプロセッサを必要とする [19].

[Step 4(第 3.9 節)] 構文解析木の葉に当たる節点に対して 1 本の有向辺と 1 個の節点を付加しているので, 定数時間と $O(n)$ 個のプロセッサを必要とする.

以上の各 **Step** の時間計算量とプロセッサ数を総合すると, 本章で提案した LR構文解析の並列アルゴリズムは, $O(\log n)$ 時間と $O(n^3)$ 個 ($= O(n + n^{2.376} + n^2 + n^3)$) のプロセッサを必要とする.

一方, 文献 [26] の決定性の文脈自由言語を認識する並列アルゴリズムは, $O(\log^2 n)$ 時間と $O(n^{2+\varepsilon})$ 個, $0 < \varepsilon < 1$, のプロセッサが必要である. しかし, その並列アルゴリズムで得られた情報から構文解析木を得るには, 文献 [19] の定理から少なくとも $O(n^3)$ 個のプロセッサが必要となり, 文献 [26] のアルゴリズムを利用する限り $O(\log^2 n)$ 時間と $O(n^3)$ 個のプロセッサが必要となる.

3.12 むすび

本章では, CREW P-RAM 上において, $O(\log n)$ 時間と $O(n^3)$ 個のプロセッサを用いて LR構文解析を行う並列アルゴリズムを提案した. この並列アルゴリズムは, **LRA** の各状態からの遷移の可能性を求め, それをペブルゲーム法に適用することで実現しており, その利点は第 1 に文脈自由文法の部分クラスの中で文の生成能力の点でコンパイラなどの作成に十分広い文法のクラスである LR文法に対して並列に構文解析ができる点, 第 2 にこれまで LR文法であっても並列に構文解析する時は一般の文脈自由文法として扱われるため理論的に効率の良い構文解析の並列アルゴリズム [19] は, $O(\log^2 n)$ 時間と $O(n^6)$ 個のプロセッサを要するのに対

して, 本並列アルゴリズムはより少ない $O(n^3)$ 個のプロセッサ数で済む点である.

第 4 章

unrestricted LR 文法及び unrestricted LR 構文解析法の提案

4.1 はじめに

第 4 章

unrestricted LR 文法 及び unrestricted LR 構文解析法の提案

4.1 はじめに

これまで文脈自由言語及びその部分クラスの言語に対する構文解析法は, 十分に研究されてきた. 一方, 文脈自由言語より広いクラスの言語も自然言語処理などに必要となってきている. それにもかかわらず, 文法の生成規則に制限を与えない unrestricted 文法から生成される言語に対しては, Loeckx, Walters, Vold'man 及び Harris によって提案された構文解析手法が知られているのみである [6, 17].

特に, Harris によって提案された unrestricted $SLR(1)$ 構文解析法 [17] や unrestricted $LALR(1)$ 構文解析法 [17] は LR構文解析法 [23] を文脈自由言語より大きいクラスに拡張する方法として知られている. しかし, unrestricted $SLR(1)$ 構文解析法 や unrestricted $LALR(1)$ 構文解析法は従来から知られている $SLR(1)$ 構文解析や $LALR$構文解析の拡張で, その性質として先読みの文字列数が終端記号の 1 個に限定されているため, 構文解析のできる言語のクラスが限定される.

一方, 先読み文字を使用しない方法の例として, Earley 法 [23] を拡張した G.Sh. Vol'dman による方法 [6] がある. しかし, この方法では対象としている言語は文脈依存言語に限定される.

そこで, 本論文では先読み文字列を入力文字列から非終端記号と入力の最後を示す特殊記号$に変更することによって, 先読み文字列の個数を

限定しない unresticted $LR(k)$ 文法, 及び, unrestricted $LR(k)$ 構文解析法を提案する.

例えば, 文法 $G_1 = (P, S, T, N)$, $P = \{ S \to ACD, S \to ACE, C \to cC, A \to a, B \to a, C \to c, D \to d, E \to e\}$, $N = \{S, A, B, C, D, E\}$, $T = \{a, c, d, e\}$ に対して, LR 構文解析を実行すると, 図 4.1に示すような入力 a に対する木を一つに特定するのには c のあとの d または e の出現が必要となる. しかし文法 G_1の場合, c の入力の可能性が無限個あるので G_1は LR 文法ではない.

それに対して, unrestricted LR(k) 文法では, $A \to a$ または $B \to a$ を導出するのに A または Bの右側の兄弟である 2 個の CD と CEを先読みに変更した.

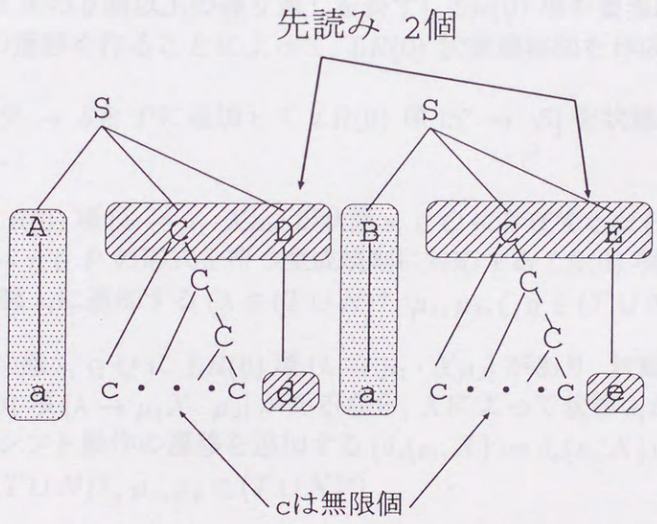


図 4.1: 構文解析木の例.

Fig. 4.1: An example of parsing tree.

4.2 準備

本節では準備として記号列集合 $head_k$について述べる.

- $head_k(\lambda)$ はλの先頭から長さ kの記号列である.

4.3 $LR(0)$ 状態遷移図

unrestricted LR 文法で利用する $LR(0)$ 状態遷移図 $M = (Q, \Sigma, \Gamma, \delta, q_0, S')$ に対して, Q: $LR(0)$ 状態遷移図の状態の有限集合, Σ: 入力アルファベット, Γ: スタックアルファベット, δ:遷移関数 (δ は以下に示すシフト動作の遷移関数δ_sとレデュース動作の遷移関数δ_rの和 ($\delta = \delta_s \cup \delta_r$) からなり, $\delta_s(q, X)$=$\{(S, q') | q, q' \in Q, X \in T \cup N \cup \varepsilon\}$, $\delta_r(q) = \{(R, \lambda, \mu) | q \in Q, \lambda \to \mu \in P\}$ である), q_0: $q_0 \in Q$ で, $LR(0)$ 状態遷移図の初期状態, S': 開始記号を表す.

また, 文法 G=(P, S, T, N) が与えられた時, 生成規則$\lambda \to \mu_1\mu_2$に対してメタ記号 $\cdot$ をつけた形式 $[\lambda \to \mu_1 \cdot \mu_2]$ を $LR(0)$ 項と呼び ($\lambda \in (T \cup N)^+$, $\mu_1, \mu_2 \in (T \cup N)^*$, $(T \cup N)^+$はTまたはNの 1 回以上の繰り返し, $(T \cup N)^*$はTまたはNの 0 回以上の繰り返しを表す), $LR(0)$ 項を要素に持つ状態と状態間の遷移を作ることによって, $LR(0)$ 状態遷移図を作成する.

[作成 1] $S' \to S$を Pに追加して $LR(0)$ 項 $[S' \to \cdot S]$ を状態 $s_0 \in Q$ とする.

[作成 2] $LR(0)$ 項 $[\lambda \to \mu_1 \cdot X\mu_2]$ が状態 $s_i \in Q$ に存在するなら, 生成規則 $X\zeta \to \eta \in P$ の形式を持つ生成規則に対応する $LR(0)$ 項 $[X\zeta \to \cdot\eta]$ を状態 s_iに追加する ($\lambda \in (T \cup N)^+$, $\mu_1, \mu_2, \zeta, \eta \in (T \cup N)^*$).

[作成 3] 状態 $s_i \in Q$ に $LR(0)$ 項 $[\lambda \to \mu_1 \cdot X\mu_2]$ があり, 状態 $s_j \in Q$ に $LR(0)$ 項 $[\lambda \to \mu_1 X \cdot \mu_2]$ があるなら, Xによって状態 s_iから状態 s_jへのシフト動作の遷移を追加する ($\delta_s(s_i, X) := \delta_s(s_i, X) \cup \{(S, s_j)\}$, $\lambda \in (T \cup N)^+$, $\mu_1, \mu_2 \in (T \cup N)^*$).

[作成 4] 状態 $s_i \in Q$ に $LR(0)$ 項 $[\lambda \to \mu\cdot]$ があるなら, 状態 s_iにおいて, レデュース動作の遷移を追加する ($\delta_r(s_i) := \delta_r(s_i) \cup \{(R, \lambda, \mu)\}$, $\lambda \in (T \cup N)^+$, $\mu \in (T \cup N)^+$, μはεではないことに注意).

[作成 5] 生成規則の右側にεを含む生成規則$\lambda \to \varepsilon$, ($\lambda \in (T \cup N)^+$) があり, 状態 $s_i \in Q$ に $LR(0)$ 項 $[\lambda \to \cdot\varepsilon]$ があるなら, $LR(0)$ 項 $[\lambda \to \varepsilon\cdot]$ を持つ状態 s_jを Q に追加し ($Q := Q \cup \{s_j\}$), 状態 s_iから s_jへεによるシフト動作の遷移を追加し ($\delta_s(s_i, \varepsilon) := \delta_s(s_i, \varepsilon) \cup \{(S, s_j)\}$), 状態 s_iにおいてレデュース動作の遷移を追加する ($\delta_r(s_j) := \delta_r(s_j) \cup \{(R, \lambda, \varepsilon)\}$).

ここで, 文法 $G_2 = (\ \{1 : S \to EBE, 2 : EA \to EC, 3 : EB \to EC, 4 : CA \to AAC, 5 : CE \to AAE, 6 : E \to \varepsilon, 7 : A \to a, 8 : B \to a\}$,

$\{S,A,B,C,E\}$, $\{a\}$, S) に対する $LR(0)$ 状態遷移図を図 4.2に示す. 文法 G_1は文脈依存文法ではない例である.

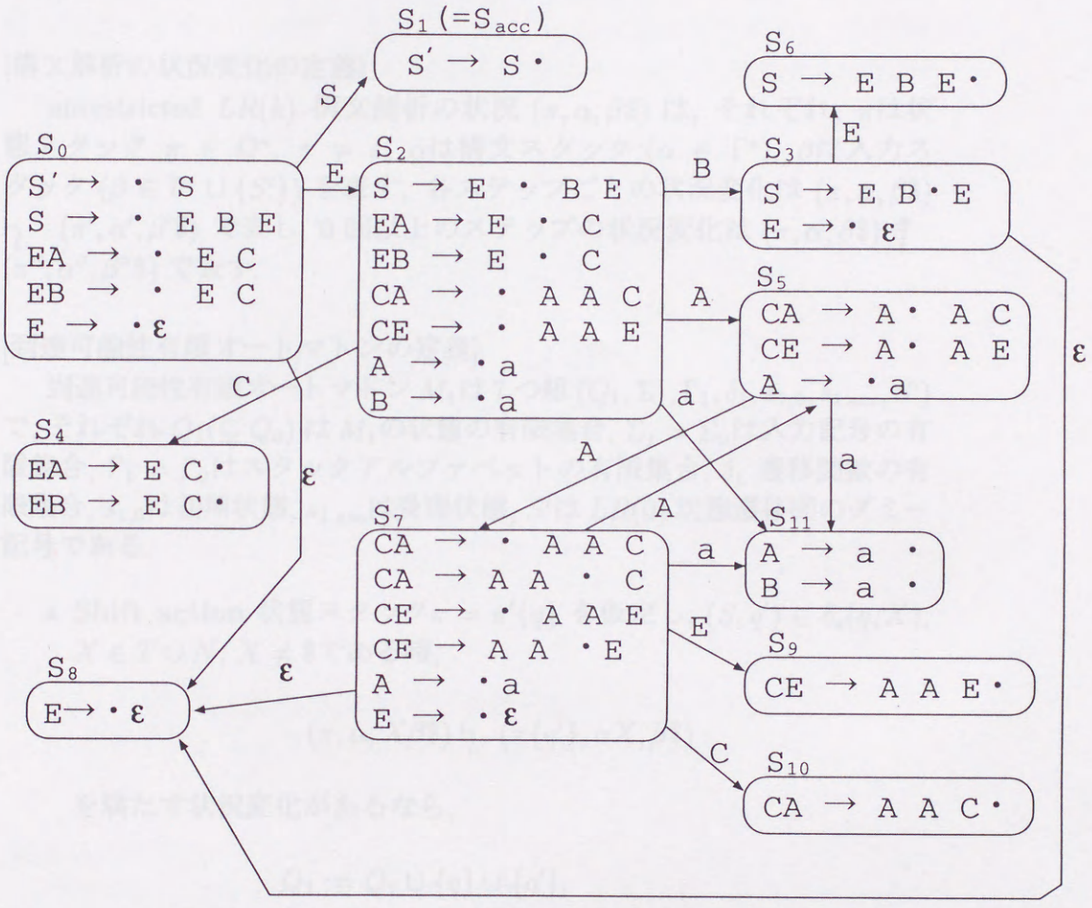


図 4.2: 文法 G_2に対する $LR(0)$ 状態遷移図.

Fig. 4.2: $LR(0)$ automaton for G_2.

4.4 unrestricted LR(k) 文法の定義

本節では, unrestricted LR(k) 文法及び構文解析で用いられる定義を示す.

[構文解析の状況変化の定義]

unrestricted $LR(k)$ 構文解析の状況 $(\pi, \alpha, \beta\$)$ は, それぞれ, πは状態スタック $\pi \in Q^*$, $\pi \neq \varepsilon$, αは構文スタック ($\alpha \in \Gamma^*$), βは入力スタック ($\beta \in \Gamma^* \cup \{S'\}$) を表す. 各ステップごとの状況変化は $(\pi, \alpha, \beta\$) \vdash_l (\pi', \alpha', \beta'\$)$ で表し, 0 回以上のステップの状況変化は $(\pi, \alpha, \beta\$) \vdash_l^* (\pi'', \alpha'', \beta''\$)$ で表す.

[到達可能性有限オートマトンの定義]

到達可能性有限オートマトン M_1は 7 つ組 $(Q_1, \Sigma_1, \Gamma_1, \delta_1, s_{1,0}, s_{1.acc}, S')$ で, それぞれ $Q_1(\subseteq Q_0)$ は M_1の状態の有限集合, $\Sigma_1 = \Sigma_0$は入力記号の有限集合, $\Gamma_1 = \Gamma_0$はスタックアルファベットの有限集合, δ_1 遷移関数の有限集合, $s_{1,0}$は初期状態, $s_{1.acc}$は受理状態, S'は $LR(0)$ 状態遷移図のダミー記号である.

- **Shift action** 状態スタック$\pi = \pi'\{q\}$ を仮定し, $(S, q') \in \delta_s(q, X)$, $X \in T \cup N$, $X \neq \$$である時,

$$(\pi, \alpha, X\beta\$) \vdash_l (\pi\{q'\}, \alpha X, \beta\$)$$

を満たす状況変化があるなら,

$$Q_1 := Q_1 \cup \{q\} \cup \{q'\},$$
$$\delta_1(q, X) := \delta_1(q, X) \cup \{q'\} \text{ とする.}$$

- **Reduce action** $(R, \lambda, \mu X) \in \delta_r(q)$, $X \in T \cup N$, $\alpha = \alpha_0\Upsilon$, $\Upsilon = \Upsilon_1, \ldots, \Upsilon_{|\mu|}$である時,

$$(\pi, \alpha X, \beta\$) \vdash_l (\pi_0, \alpha_0, \lambda\beta\$) \vdash_l^* (\pi_1\{q'\}, \alpha', \beta\$)$$

を満たす状況変化があるなら,

$$Q_1 := Q_1 \cup \{q\} \cup \{q'\},$$
$$\delta'(q, X) := \delta_1(q, X) \cup \{q'\} \text{ とする.}$$

文法例 G_2 = ($\{1 : S \to EBE, 2 : EA \to EC, 3 : EB \to EC, 4 : CA \to AAC, 5 : CE \to AAE, 6 : E \to \varepsilon, 7 : A \to a, 8 : B \to a,\}$, $\{S,A,B,C,E\}$, $\{a\}$, S) に対して, $LR(0)$ 状態遷移図 (図 4.2) の状態 s_0から s_2に非終端記号 Eのシフト動作によって遷移している場合, 到達可能性有限オートマトンでは Eによる s_0から s_2への遷移を作成する. 非終端記号によるシフト遷移を全ての状態の組合わせに対して求めたのが到達可能性有限オートマトンである.

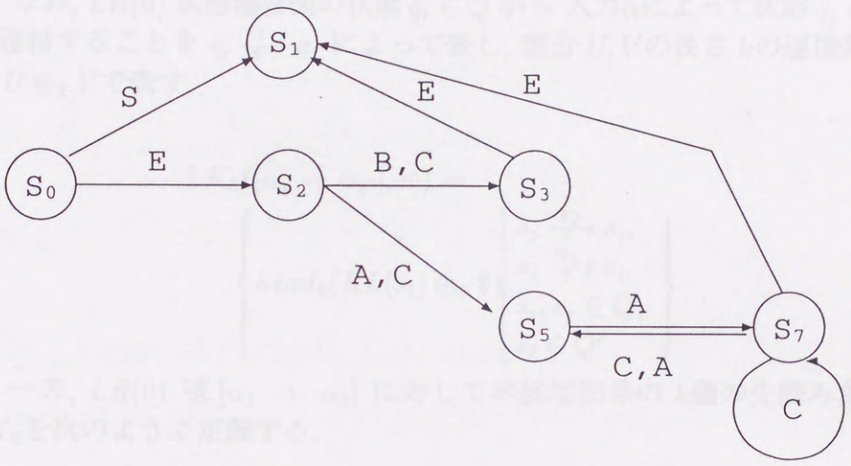


図 4.3: 文法 G_2に対する到達可能性オートマトン.
Fig. 4.3: The reachablity automaton for G_2.

[到達可能性言語]

到達可能性有限オートマトンの各状態から受理状態へ遷移させる語の集合である言語 $RL(q_i)$, $q_i \in Q'$, を

$$RL(q_i) = \left\{ w \left| \begin{array}{l} q_i^2 \in \delta_1(q_i, w_1), q_i^3 \in \delta_1(q_i^2, w_2), \ldots, \\ s_{acc} \in \delta_1(q_i^{|w|-1}, w_{|w|}),\ q_i^j \in Q', \\ 2 \le j \le |w| \end{array} \right. \right\}$$

と定義する.

また, 図 4.3の到達可能性有限オートマトンは, $LR(0)$ 状態遷移図の状態から受理状態へ到達するための残りの非終端記号列を表す. 例えば, 図 4.3の状態 s_2からは, 言語 $RL(s_2) = \{C|CA\}^*E\$$ が残りの非終端記号列であることが分かる.

[先読み文字列 LK_kの定義]

上で定義された $RL(q_i)$ を利用して状態 s_i の $LR(0)$ 項 $[\alpha_1 \to \alpha_2 \cdot]$ に対して非終端記号の k個の先読み集合 LK_kを定義する.

なお, $LR(0)$ 状態遷移図の状態 $q_i \in Q$ から 入力αによって状態 $q_j \in Q$ に遷移することを $q_i \xrightarrow[*]{\alpha} q_j$ によって表し, 集合 U, Vの長さ kの連接集合を $U \oplus_k V$で表す.

$$LK_k([\alpha_1 \to \alpha_2 \cdot], s_i) = \left\{ head_k(RL(s_l) \oplus_k \$) \;\middle|\; \begin{array}{l} s_j \xrightarrow[*]{\alpha_2} s_i, \\ s_j \xrightarrow[*]{\alpha_1} s_l, \\ s_i, s_j \in Q, \\ s_l \in Q' \end{array} \right\}$$

一方, $LR(0)$ 項 $[\alpha_1 \to \cdot\alpha_2]$ に対して非終端記号の k個の先読み集合 LK_kを次のように定義する.

$$LK_k([\alpha_1 \to \cdot\alpha_2], s_i) = \left\{ head_k(RL(s_l) \oplus_k \$) \;\middle|\; \begin{array}{l} s_i \xrightarrow[*]{\alpha_1} s_l, \\ s_i \in Q, s_l \in Q' \end{array} \right\}$$

[受理可能集合の定義]

構文解析が状態 s_i内の LR 項 $[\alpha_1 \to \alpha_2 \cdot]$ を還元する時, 次の入力を受け付ける状態の集合 AS_{core}(受理可能性集合) を以下の通り定義し,

$$AS_{core}([\alpha_1 \to \alpha_2 \cdot], s_i) = \left\{ s_l \;\middle|\; \begin{array}{l} s_j \xrightarrow[*]{\alpha_2} s_i,\ s_j \xrightarrow[*]{\alpha_1} s_l, \\ s_i, s_j \in Q,\ s_l \in Q' \end{array} \right\}$$

$LR(0)$ 項に依存しない各状態ごとの AS(受理可能性集合) を以下の通り定義する.

$$AS(s_i) = \bigcup_{p_j \in P} AS_{core}([p_j \cdot], s_i).$$

文法 G_2に対する $LR(0)$ 状態遷移図 (図 4.2) から,

$$AS_{core}([EA \to EC\cdot], s_4) = \{s_5\}$$
$$AS_{core}([EB \to EC\cdot], s_4) = \{s_3\}$$

及び

$$AS_{core}([A \to a\cdot], s_{11}) = \{s_5, s_7\}$$
$$AS_{core}([B \to a\cdot], s_{11}) = \{s_3\}$$

である.

4.5 unrestricted LR 文法

本稿で提案する unresticted LR 文法に対する**定義**について述べる.

[**定義**]
(1) 生成規則は

(a) $A \to a$, $A \in N, a \in T$,
(b) $A \to \varepsilon$, $A \in N, \varepsilon$(空語),
(c) $\lambda \to \mu$, $\lambda, \mu \in N^+$

の 3 つの形式のみからなり,
(2) 文法 G の $LR(0)$ 状態遷移図の状態 s_i の $LR(0)$ 項 $[p_i \cdot]$, $p_i \in P$ に対して,

$$\bigcap_{p_j \in P} AS_{core}([p_j \cdot], s_i) = \phi$$

を満たし,
(3) $\alpha_l \to \beta \in P$, $\alpha_i, \beta \in N^+$, $i = 1, \ldots, |\alpha_i|$ に対して, $LR(0)$ 項 $[\alpha_l \to \cdot\beta]$ が同一の状態 s_jに存在して,

$$\bigcap_{\alpha_l \to \beta \in P} LK_k([\alpha_l \to \cdot\beta], s_j) = \phi$$

を満たすとき, 文法 G を unrestricted $LR(k)$ 文法と定義する.

文法 G_2の場合, LR 項 $[EA \to \cdot EC]$ と $[EB \to \cdot EC]$, 及び, LR 項 $[A \to \cdot a]$ と $[B \to \cdot a]$ に関して先読み文字列 LK_1を求めると, それぞれ

$$LK_1([EA \to \cdot EC], s_0) = \{A\}$$
$$LK_1([EB \to \cdot EC], s_0) = \{E\}$$

及び

$$LK_1([A \to \cdot a], s_2) = \{A\}$$
$$LK_1([B \to \cdot a], s_2) = \{B\}$$

表 4.1: 文法 G_2に対する AS(受理可能性集合)

Table 4.1: AS_{core} **for** G_2

$AS_{core}([S' \to S\cdot], s_1) = \{s_1\}$	$AS_{core}([EA \to EC\cdot], s_4) = \{s_5\}$
$AS_{core}([EB \to EC\cdot], s_4) = \{s_3\}$	$AS_{core}([S \to EBE\cdot], s_6) = \{s_1\}$
$AS_{core}([E \to \varepsilon\cdot], s_8) = \{s_1, s_2\}$	$AS_{core}([CE \to AAE\cdot], s_9) = \{s_1\}$
$AS_{core}([CA \to AAC\cdot], s_{10}) = \{s_7\}$	$AS_{core}([A \to a\cdot], s_{11}) = \{s_5, s_7\}$
$AS_{core}([B \to a\cdot], s_{11}) = \{s_3\}$	

表 4.2: 文法 G_2に対する先読み集合 LK_1

Table 4.2: **The set of the lookahead strings LK_1 for G_2**

$LK_1([EA \to EC\cdot], s_4) = \{A\}$	$LK_1([EB \to EC\cdot], s_4) = \{E\}$
$LK_1([E \to \varepsilon\cdot], s_8) = \{A, B, C, E, \$\}$	$LK_1([S \to EBE\cdot], s_6) = \{\$\}$
$LK_1([CE \to AAE\cdot], s_9) = \{\$\}$	$LK_1([CA \to AAC\cdot], s_{10}) = \{A, C, E\}$
$LK_1([A \to a\cdot], s_{11}) = \{A, C, E\}$	$LK_1([B \to a\cdot], s_{11}) = \{E\}$
$LK_1([S' \to \cdot S], s_0) = \{\$\}$	$LK_1([S \to \cdot EBE], s_0) = \{\$\}$
$LK_1([EA \to \cdot EC], s_0) = \{A\}$	$LK_1([EB \to \cdot EC], s_0) = \{E\}$
$LK_1([E \to \cdot\varepsilon], s_0) = \{A, B, C\}$	$LK_1([A \to \cdot a], s_2) = \{A\}$
$LK_1([B \to \cdot a], s_2) = \{B\}$	$LK_1([CA \to \cdot AAC], s_2) = \{A, C, E\}$
$LK_1([CE \to \cdot AAE], s_2) = \{\$\}$	$LK_1([E \to \cdot\varepsilon], s_3) = \{\$\}$
$LK_1([A \to \cdot a], s_5) = \{A, C, E\}$	$LK_1([CA \to \cdot AAC], s_7) = \{A, C, E\}$
$LK_1([CE \to \cdot AAE], s_7) = \{\$\}$	$LK_1([A \to \cdot a], s_7) = \{A\}$

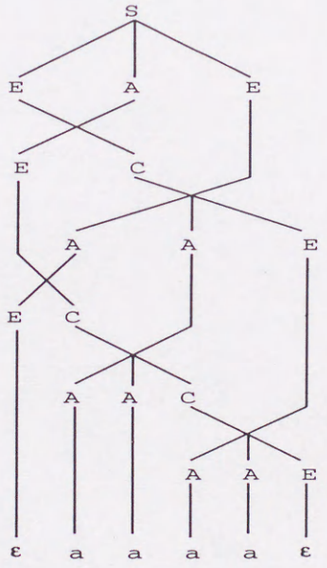


図 4.4: 文法 G_2に対する入力 $I = aaaa$ の構文解析木.

Fig. 4.4: The parsing tree for $I = aaaa$ for G_2.

4.6 unrestricted LR 文法の例

[例] unrestricted LR(2) 文法の例を以下に示す.

$G_3 = (\ \{1 : S \to DE,\ 2 : DE \to ED,\ 3 : E \to ABB,\ 4 : E \to CB,\ 5 : A \to AA,\ 6 : A \to a,\ 7 : C \to CA,\ 8 : C \to a,\ 9 : B \to b,\ 10 : D \to d\ \},\ \{S,\ A,\ B,\ C,\ D,\ E\},\ \{a,\ b,\ d\},\ S\ )$.

これに対する G_3の $LR(0)$ 状態遷移図, 及び, 到達可能性オートマトンを図 4.5, 4.6に示す.

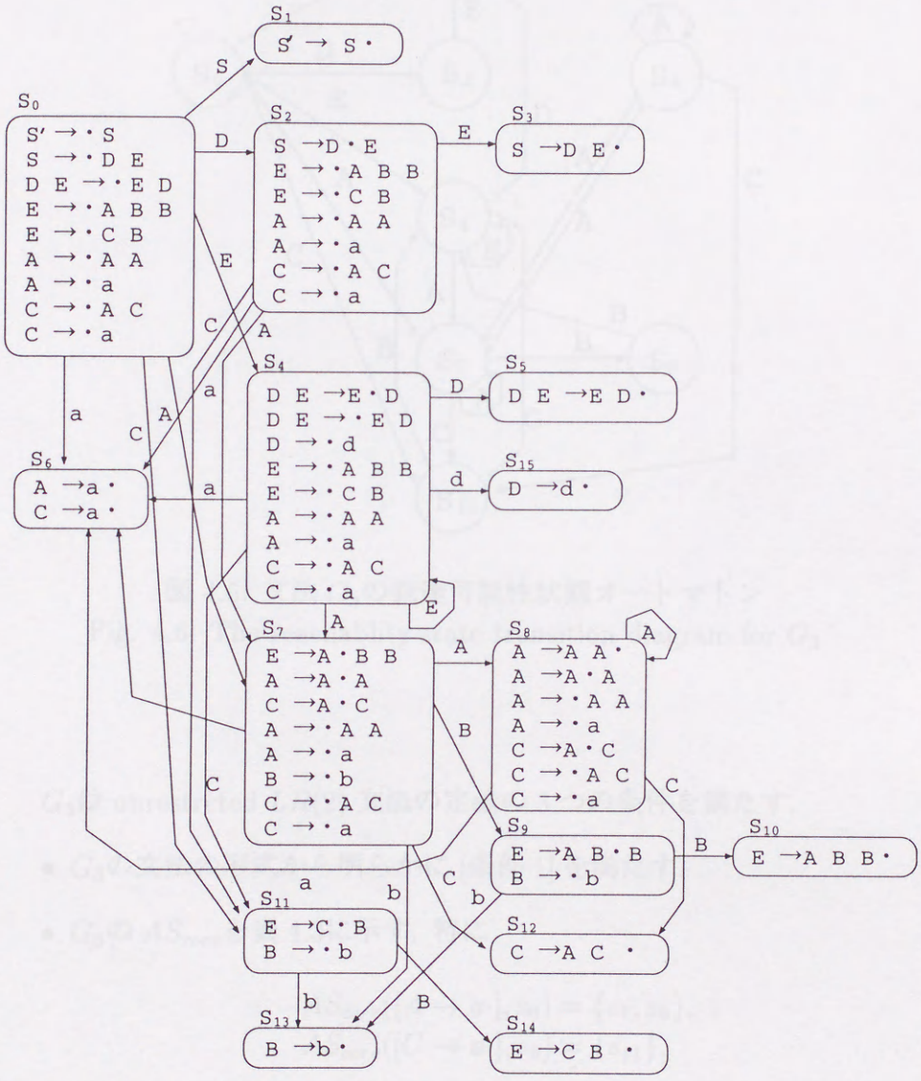


図 4.5: 文法 G_3の LR(0) 状態遷移図

Fig. 4.5: The LR(0) state transition diagram for G_3

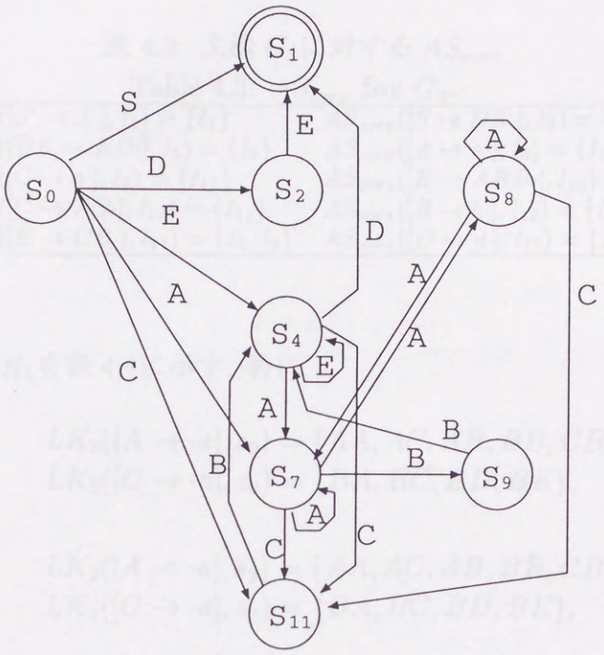


図 4.6: 文法 G_3の到達可能性状態オートマトン
Fig. 4.6: The reachablity state transition diagram for G_3

G_3は unrestrcted $LR(2)$ 文法の定義の 3 つの条件を満たす.

- G_3の文法の形式から明らかに [条件 1] を満たす.
- G_3の AS_{core}を表 4.3に示す. 特に

$$AS_{core}([A \to a\cdot], s_6) = \{s_7, s_8\},$$
$$AS_{core}([C \to a\cdot], s_6) = \{s_{11}\}.$$

であるので, 定義の [条件 2] を満たす.

表 4.3: 文法 G_3に対する AS_{core}

Table 4.3: AS_{core} **for** G_3.

$AS_{core}([S' \to S\cdot], I_1) = \{I_1\}$	$AS_{core}([S \to DE\cdot], I_3) = \{I_1\}$
$AS_{core}([DE \to ED\cdot], I_5) = \{I_1\}$	$AS_{core}([A \to a\cdot], I_6) = \{I_7, I_8\}$
$AS_{core}([C \to a\cdot], I_6) = \{I_{11}\}$	$AS_{core}([E \to ABB\cdot], I_{10}) = \{I_4\}$
$AS_{core}([C \to AC\cdot], I_{12}) = \{I_{11}\}$	$AS_{core}([B \to b\cdot], I_{13}) = \{I_4, I_9\}$
$AS_{core}([E \to CB\cdot], I_{14}) = \{I_4, I_9\}$	$AS_{core}([D \to d\cdot], I_{15}) = \{I_1\}$

- G_3の LR_kを表 4.4に示す. 特に

$$
\begin{aligned}
&LK_2([A \to \cdot a], s_0) = \{AA, AC, AB, BB, CB\},\\
&LK_2([C \to \cdot a], s_0) = \{BA, BC, BD, BE\},
\end{aligned}
$$

$$
\begin{aligned}
&LK_2([A \to \cdot a], s_2) = \{AA, AC, AB, BB, CB\},\\
&LK_2([C \to \cdot a], s_2) = \{BA, BC, BD, BE\},
\end{aligned}
$$

$$
\begin{aligned}
&LK_2([A \to \cdot a], s_4) = \{AA, AC, AB, BB, CB\},\\
&LK_2([C \to \cdot a], s_4) = \{BA, BC, BD, BE\},
\end{aligned}
$$

$$
\begin{aligned}
&LK_2([A \to \cdot a], s_7) = \{AA, AC, AB, BB, CB\},\\
&LK_2([C \to \cdot a], s_7) = \{BA, BC, BD, BE\},
\end{aligned}
$$

$$
\begin{aligned}
&LK_2([A \to \cdot a], s_8) = \{AA, AC, AB, BB, CB\},\\
&LK_2([C \to \cdot a], s_8) = \{BA, BC, BD, BE\}.
\end{aligned}
$$

である.

よって上記の表の結果から定義の [条件 3] を満たす.

表 4.4: G_3に対する先読み集合 LK_2

Table 4.4: **The set of the lookahead strings LK_2 for G_3.**

$LK_2([S \to DE\cdot], I_3) = \{\$\}$
$LK_2([DE \to ED\cdot], I_5) = \{\$\}$
$LK_2([A \to a\cdot], I_6) = \{AA, AB, AC, BB, CB\}$
$LK_2([C \to a\cdot], I_6) = \{BA, BC, BD, BE\}$
$LK_2([A \to AA\cdot], I_8) = \{AA, AB, AC, BB, CB\}$
$LK_2([E \to ABB\cdot], I_{10}) = \{AA, AB, AC, CB, D\$, EA, EC, EE\}$
$LK_2([C \to AC\cdot], I_{12}) = \{BA, BC, BD, BE\}$
$LK_2([B \to b\cdot], I_{13}) = \{AA, AB, AC, BA, BC, BD, BE, CB, D\$, EA, EC, EE\}$
$LK_2([E \to CB\cdot], I_{14}) = \{AA, AB, AC, CB, D\$, EA, EC, EE\}$
$LK_2([D \to d\cdot], I_{15}) = \{\$\}$

$LK_2([S' \to \cdot S], I_0) = \{\$\}$
$LK_2([S \to \cdot DE], I_0) = \{\$\}$
$LK_2([DE \to \cdot ED], I_0) = \{\$\}$
$LK_2([E \to \cdot ABB], I_0) = \{AA, AB, AC, CB, D\$, EA, EC, EE\}$
$LK_2([E \to \cdot CB], I_0) = \{AA, AB, AC, CB, D\$, EA, EC, EE\}$
$LK_2([A \to \cdot AA], I_0) = \{AA, AB, AC, BB, CB\}$
$LK_2([A \to \cdot a], I_0) = \{AA, AB, AC, BB, CB\}$
$LK_2([C \to \cdot a], I_0) = \{BA, BC, BD, BE, CB\}$
$LK_2([C \to \cdot AC], I_0) = \{BA, BC, BD, BE, CB\}$
$LK_2([E \to \cdot ABB], I_2) = \{\$\}$ $LK_2([E \to \cdot CB], I_2) = \{\$\}$
$LK_2([A \to \cdot AA], I_2) = \{AA, AB, AC, BB, CB\}$
$LK_2([C \to \cdot AC], I_2) = \{BA, BC, BD, BE, CB\}$
$LK_2([C \to \cdot a], I_2) = \{BA, BC, BD, BE, CB\}$
$LK_2([DE \to \cdot ED], I_4) = \{\$\}$ $LK_2([D \to \cdot d], I_4) = \{\$\}$
$LK_2([E \to \cdot ABB], I_4) = \{AA, AB, AC, CB, D\$, EA, EC, EE\}$
$LK_2([E \to \cdot CB], I_4) = \{AA, AB, AC, CB, D\$, EA, EC, EE\}$
$LK_2([A \to \cdot AA], I_4) = \{AA, AB, AC, BB, CB\}$
$LK_2([A \to \cdot a], I_4) = \{AA, AB, AC, BB, CB\}$
$LK_2([C \to \cdot AC], I_4) = \{BA, BC, BD, BE, CB\}$
$LK_2([C \to \cdot a], I_4) = \{BA, BC, BD, BE, CB\}$
$LK_2([A \to \cdot AA], I_7) = \{AA, AB, AC, BB, CB\}$
$LK_2([A \to \cdot a], I_7) = \{AA, AB, AC, BB, CB\}$
$LK_2([B \to \cdot b], I_7) = \{BA, BC, BE, BD\}$
$LK_2([C \to \cdot AC], I_7) = \{BA, BC, BD, BE, CB\}$
$LK_2([C \to \cdot a], I_7) = \{BA, BC, BD, BE, CB\}$
$LK_2([A \to \cdot AA], I_8) = \{AA, AB, AC, BB, CB\}$
$LK_2([A \to \cdot a], I_8) = \{AA, AB, AC, BB, CB\}$
$LK_2([C \to \cdot AC], I_8) = \{BA, BC, BD, BE, CB\}$
$LK_2([C \to \cdot a], I_8) = \{BA, BC, BD, BE, CB\}$
$LK_2([B \to \cdot b], I_9) = \{AA, AB, AC, CB, D\$, EA, EC, EE\}$
$LK_2([B \to \cdot b], I_{11}) = \{AA, AB, AC, CB, D\$, EA, EC, EE\}$

4.7 unrestricted LR 構文解析

unrestricted $LR(k)$ 構文解析 $M = (Q, \Sigma, \Gamma, \delta, q_0, S')$ を定義する (Q, Σ, Γ, δ, q_0, S'は $LR(0)$ 状態遷移図と同様, kは先読み文字数).

特に unrestricted $LR(k)$ 構文解析の動作 **[動作 1] ～ [動作 6]** を次に示す. 但し, unrestricted $LR(k)$ 構文解析の状況 $(\pi, \alpha, \beta\$)$ を**第 4.4 節**の定義と同じとする.

[動作 1] 初期設定
入力をβとする時, 初期状態として unrestricted $LR(k)$ 構文解析の状況を

$$(\{s_0\}, \varepsilon, \beta\$)$$

とする.

[動作 2] シフト動作
文法記号 $X = T \cup N$, $X \neq \$$によってシフト動作を定義する. 構文解析の状況が$\pi = \pi'\{q_1 \ldots q_n\}$, $\beta\$ = X\beta'\$$ であるとする. この時, $(S, r_i) \in \cup_{j=1}^{n} \delta_s(q_j, X)$, $i = 1, \ldots, m$ $\cup_{j=1}^{n} \delta_r(q_j) = \phi$ が成り立つ場合,

$$(\pi, \alpha, \beta\$) \vdash_{\gamma} \ (\pi\{r_1, \ldots, r_m\}, \alpha\{X, \ldots, X\}, \beta'\$)$$

とする.

[動作 3] レデュース (還元) 動作の確定
$\lambda \to \mu \in P$, $\mu = \mu_1$, ..., $\mu_{|\mu|}$, $|\mu| = \mu$の文法記号の数 (但し, εは 1 と数える), をレデュースする動作と定義する. 構文解析の状況が$\pi = \pi'\{q_1, q_2, \ldots, q_n\}$, $\pi = \pi_0\{q'_1, \ldots, q'_m\}\pi'_0$, $\alpha = \alpha_0\Upsilon$, π'_0の状態集合の数 $= |\Upsilon| = |\mu|$, であるとする. この時, $(R, \lambda, \mu) \in \cup_{j=1}^{n} \delta_r(q_j)$, $head_k(\beta\$) \in \cup_{j=1}^{n} LK_k([\lambda \to \mu\cdot], q_j)$, $head_k(\beta\$) \in \cup_{l=1}^{m} LK_k([\lambda \to \cdot\mu], q'_l)$, が成り立つ場合,

$$(\pi, \alpha, \beta\$) \vdash_{\gamma} \ (\pi_0\{q'_1, \ldots, q'_m\}, \alpha_0, \lambda\beta\$)$$

とする.

[動作 4] レデュース (還元) 動作の予測
構文解析の状況が $\pi = \pi'\{q_1, q_2, \ldots, q_n\}$, $\alpha = \alpha_0\{X_1, X_2, \ldots, X_n\}$, $X_i \in$

$T \cup N$, $1 \leq i \leq n$ であるとする. この時, $\cup_{i=1}^{n} \delta_s(q_i, head_1(\beta\$)) = \phi$, $head_k(\beta\$) \notin \cup_{p_j \in P} LK_k([p_j \cdot], q_i)$ $\cup_{j=1}^{n} \delta_s(q_j, \varepsilon) = \phi$, が成り立つ場合,

$$(\pi, \alpha, \beta\$) \vdash (\pi' \{q_1, q_2, \ldots, q_n\} \cup \cup_{i=1}^{n} RS(q_i), \alpha, \beta\$)$$

とする.

[動作 5] ε (空語) のシフトの予測
構文解析の状況が $\pi = \pi' \{q_1, q_2, \ldots, q_n\}$, $\beta = X\beta'$, $X \in T \cup N$であるとする. この時, $\cup_{j=1}^{n} \delta_s(q_j, X) = \phi$, $\cup_{j=1}^{n} \delta_r(q_j) = \phi$, $(S, r_j) \in \cup_{i=1}^{n} \delta_s(q_i, \varepsilon)$, $1 \leq j \leq m$ が成り立つ場合,

$$(\pi, \alpha, \beta\$) \vdash (\pi\{r_1, r_2, \ldots, r_m\}, \alpha\{\varepsilon, \ldots, \varepsilon\}, \beta\$)$$

[動作 6 Stop]
構文解析の状況が $\pi = \{s_0\}$, $\alpha = \varepsilon$, $\beta\$ = S'\$$ である場合, unrestricted $LR(k)$ 構文解析は入力を正しいとして終了する. また, **[動作 1] ～ [動作 5]** に当てはまらない状況になった場合は, unrestricted $LR(k)$ 構文解析は入力にエラーがあるとして終了する.

4.8 unrestricted LR 構文解析の例

文法 G_2に対する入力 $I = a^{2^0} = a$ の構文解析の処理を表 4.5 に示す.

表 4.5: unrestricted 文法 G_2に対する入力 $I = a$ の LR 構文解析の例.

Table 4.5: **An example of unrestricted LR parser for input $I = a$ on G_2.**

Step	π	α	$\beta\$$
1	$\{s_0\}$	ε	$a\$$
2	$\{s_0\}\{s_8\}$	$\{\varepsilon\}$	$a\$$
3	$\{s_0\}\{s_8, s_1, s_2\}$	$\{\varepsilon\}$	$a\$$
4	$\{s_0\}\{s_8, s_1, s_2\}\{s_{11}\}$	$\{\varepsilon\}\{a\}$	$\$$
5	$\{s_0\}\{s_8, s_1, s_2\}$		
	$\{s_{11}, s_3, s_5, s_7\}$	$\{\varepsilon\}\{a\}$	$\$$
6	$\{s_0\}\{s_8, s_1, s_2\}$		
	$\{s_{11}, s_3, s_5, s_7\}\{s_8\}$	$\{\varepsilon\}\{a\}\{\varepsilon\}$	$\$$
7	$\{s_0\}\{s_8, s_1, s_2\}$		
	$\{s_{11}, s_3, s_5, s_7\}$	$\{\varepsilon\}\{a\}$	$E\$$
8	$\{s_0\}\{s_8, s_1, s_2\}$	$\{\varepsilon\}$	$BE\$$
9	$\{s_0\}$	ε	$EBE\$$
10	$\{s_0\}\{s_2\}$	$\{E\}$	$BE\$$
11	$\{s_0\}\{s_2\}\{s_3\}$	$\{E\}\{B\}$	$E\$$
12	$\{s_0\}\{s_2\}\{s_3\}\{s_6\}$	$\{E\}\{B\}\{E\}$	$\$$
13	$\{s_0\}$	$\{\varepsilon\}$	$S\$$
14	$\{s_0\}\{s_1\}$	$\{S\}$	$\$$
15	$\{s_0\}$	$\{\varepsilon\}$	$S'\$$

このうち, [**動作 1**]〜[**動作 6**] の各動作について条件及び構文解析の状況について説明する.

- 構文解析の処理は最初に [**動作 1**] として $(\{s_0\}, \varepsilon, a\$)$ を構文解析の状況とする.

- 次に **Step 1** では, $\delta_r(q_i) = \phi$, $(S, s_8) \in \delta_s(s_0, \varepsilon)$ であるので, [**動作 5**] を実行し構文解析の状況を次のように変化させる.

$$(\{s_0\}, \varepsilon, a\$) \vdash_7 (\{s_0\}\{s_8\}, \{\varepsilon\}, a\$)$$

- **Step 2** では, $\delta(s_8, a) = \phi$, $head_1(a\$) = a \notin LK_k([E \to \varepsilon], s_8) = \{A, B, C, E, \$\}$ であるので, [**動作 4**] を実行し構文解析の状況を次のように変化させる ($\{s_8\}$ に $RS(s_8) = \{s_1, s_2\}$ を追加する).

$$(\{s_0\}\{s_8\}, \{\varepsilon\}, a\$) \vdash_7 (\{s_0\}\{s_8, s_1, s_2\}, \{\varepsilon\}, a\$)$$

- **Step 3** では, $\delta(s_8, a) = \phi$, $\delta(s_1, a) = \phi$, $\delta(s_2, a) = \{s_{11}\}$ であるので, [**動作 2**] を実行し構文解析の状況を次のように変化させる (s_{11} を新たに状態スタックに積み, a を構文スタックに積む).

$$(\{s_0\}\{s_8, s_1, s_2\}, \{\varepsilon\}, a\$) \vdash_7 (\{s_0\}\{s_8, s_1, s_2\}\{s_{11}\}, \{\varepsilon\}\{a\}, \$)$$

- **Step 7** で構文解析の状況が

$$(\{s_0\{s_8, s_1, s_2\}\{s_{11}, s_3, s_5, s_7\}, \{\varepsilon\}\{a\}, E\$)$$

であるとする.

$\delta_r(s_{11}) = \{(R, A, a), (R, B, a)\}$, $head_k(E\$) \in LK_k([B \to a\cdot], s_{11}) = \{E\}$, $head_k(E\$) \in LK_k([B \to \cdot a], s_8) = \{E\}$ であるので, [**動作 3**] を実行し構文解析の状況を次のように変化させる. なお, unrestricted LK(k) 文法の制約により $head_k(E\$) \in LK_k([B \to a\cdot], s_{11}) = \{E\}$ と $head_k(E\$) \in LK_k([B \to \cdot a], s_8) = \{E\}$ を成り立たせるのは生成規則 $B \to a \in P$ のみである.

$$(\{s_0\{s_8, s_1, s_2\}\{s_{11}, s_3, s_5, s_7\}, \{\varepsilon\}\{a\}, E\$) \vdash_7 (\{s_0\}\{s_8, s_1, s_2\}, \{\varepsilon\}, BE\$)$$

- **Step 15** で構文解析の状況が

$$(\{s_0\}, \varepsilon, S'\$)$$

である場合, [**動作 6**] の条件が当てはまるので構文解析は終了する.

4.9 unrestricted $LR(k)$ 文法及び unrestricted $LR(k)$ 構文解析の性質

本節では, unrestricted $LR(k)$ 文法及び unrestricted $LR(k)$ 構文解析の性質について述べる.

[定理 1] unrestricted $LR(k)$ 文法は曖昧ではない.

[証明] unrestricted $LR(k)$ 構文解析の各ステップがそれぞれ排他的な状況で動作することによって, unrestricted $LR(k)$ 文法が曖昧でないことを示す.

まず, [**動作 1**] は unrestricted LR構文解析の初期動作であり, [**動作 6**] は unrestricted LR構文解析の終了動作でので, それぞれ [**動作 2**] から [**動作 5**] における構文解析の条件と明らかに相違する.

一方, [**動作 2**],[**動作 3**], [**動作 4**] 及び [**動作 5**] における構文解析の条件はそれぞれ相違することを以下に示す.

- [**動作 2**] と [**動作 3**]

  [**動作 2**] の条件は $\cup_{j=1}^{n}\delta_r(q_j) = \phi$ である. 一方, [**動作 3**] の条件は $(Rreduce, \lambda, \mu) \in \cup_{j=1}^{n}\delta_r(q_j)$, であるので, それぞれの条件は排他的である.

- [**動作 2**] と [**動作 4**]

  [**動作 2**] の条件は $(Shift, q_i) \in \cup_{j=1}^{n}\delta_s(q_j, X)$ である. 一方 [**動作 3**] の条件は $\cup_{j=1}^{n}\delta_s(q_j, X) = \phi$ であるのでそれぞれの条件は排他的である.

- [**動作 2**] と [**動作 5**]

  [**動作 2**] の条件は $(Shift, q_i) \in \cup_{j=1}^{n}\delta_s(q_j, X)$ である. 一方 [**動作 5**] の条件は $\cup_{j=1}^{n}\delta_s(q_j, X) = \phi$であるのでそれぞれの条件は排他的である.

- **[動作 3] と [動作 4]**

  [動作 3] の条件は $head_k(\beta\$) \in \cup_{j=1}^{n} LK_k([\lambda \to \mu\cdot], q_j)$ である. 一方 **[動作 5]** の条件は $head_k(\beta\$) \notin \cup_{j=1}^{n} \cup_{p_i \in P} LK_k([p_i\cdot], q_i)$ であるのでそれぞれの条件は排他的である.

- **[動作 3] と [動作 5]**

  [動作 3] の条件は $(Reduce, \lambda, \mu) \in \cup_{j=1}^{n} \delta_r(q_j)$ である. 一方 **[動作 5]** の条件は $\cup_{j=1}^{n} \delta_r(q_j) = \phi$ であるのでそれぞれの条件は排他的である.

- **[動作 4] と [動作 5]**

  [動作 4] の条件は $\cup_{j=1}^{n} \delta_s(q_j, \varepsilon) = \phi$ である. 一方 **[動作 5]** の条件は $(Shift, r_i) \in \cup_{j=1}^{n} \delta_s(q_j, \varepsilon)$ であるので, それぞれの条件は排他的である.

以上動作ごとの条件がそれぞれ排他的であるので, unrestricted $LR(k)$ 文法が曖昧でないことが明らかになった. □

[定理 2] もし 文法 $G = (P, S, T, N)$ が unrestricted $LR(k)$ 文法であるならば, 先読み文字列数 kが決定できる.
[証明] $\alpha_1 \to \beta, \alpha_2 \to \beta \in P$, unrestricted LR 構文解析 $M = (Q, \Sigma, \Gamma, \delta, s_0, s_{acc}, S')$ 及び $[\alpha_1 \to \cdot\beta]$, $[\alpha_2 \to \cdot\beta] \in s_i$, $s_i \in Q$ であるとする.
unrestricted $LR(k)$ 文法の定義から LK_kに関して

$$LK_k([\alpha_1 \to \cdot\beta], s_i) \cap LK_k([\alpha_2 \to \cdot\beta], s_i) = \phi \text{である.}$$

また, 先読み文字列 kの長さに非有界でないなら, 以下の式で先読み文字列が定義される.

$$LK([\alpha_1 \to \cdot\beta], s_i) = \{RL(s_j) | s_i \xrightarrow[*]{\alpha_1} s_j, s_i, s_j \in Q\},$$
$$LK([\alpha_2 \to \cdot\beta], s_i) = \{RL(s_l) | s_i \xrightarrow[*]{\alpha_2} s_l, s_i, s_l \in Q\}.$$

$RL(s_j)$ と $RL(s_l)$ も以下の式で定義される.

$$RL(s_j) = \{w \in N^* \mid s_j \xrightarrow[*]{w} s_{acc}\},$$
$$RL(s_l) = \{w \in N^* \mid s_l \xrightarrow[*]{w} s_{acc}\}.$$

よって, $RL(s_j) \cap RL(s_l) = \phi$ であり, $RL(s_j)$ と $RL(s_l)$ に関して以下の式を満たす $s_m \in Q$ が存在する.

$$RL(s_j) = \{\gamma_1\xi \in N^* \mid s_j \xrightarrow[*]{\gamma_1} s_m \xrightarrow[*]{\xi} s_{acc}\},$$
$$RL(s_l) = \{\gamma_2\xi \in N^* \mid s_l \xrightarrow[*]{\gamma_2} s_m \xrightarrow[*]{\xi} s_{acc}\}.$$

上記の $s_m \in Q$ の存在性から先読み文字列長 k は $k \leq min(|\gamma_1|, |\gamma_2|)$ を満たす. それゆえ, 先読み文字列長 kを決定可能である.

但し, $|\gamma_1|, |\gamma_2| = \infty$ である場合は $k = \infty$ と定義される. □

4.10 むすび

本研究では先読み文字列の定義を非終端記号と入力の最後を示す特殊記号\$のみを含むと変更することによって unrestricted $LR(k)$ 文法を新たに定義し, またその文法に対する構文解析法を提案した.

先読み文字列を終端記号に限定している $LR(k)$ 文法の枠組みで, 文脈自由文法の先読み文字列数を予め決定して使用とすると, 先読み文字列数 kを不可能である場合が存在する. それに対して, 先読み文字列を非終端記号のみ含むとした場合, その先読み文字列が常に有限長で表現できるかどうかを今後は調べる必要がある.

第 5 章

GLR構文解析を拡張した$UGLR$構文解析の提案

5.1 はじめに

これまで自然言語処理において, 構文解析は文脈自由文法 [4] に基づいて行なうことがほとんどであった. そのため, 古くから文脈自由文法に対応する構文解析手法が幾つか考案されており, CKY 法 [4, 18, 21], Earley 法 [2, 18, 21]) および Chart 法 [18, 21]) はその代表的な例である. 最近では, 計算機における処理時間の削減や並列処理への適用のため, 自然言語の構文解析法の一つとしてGLR構文解析 [8, 11, 18, 21, 24] が使われるようになってきた. これは, LR構文解析 [4, 8, 10, 18, 21] を元にしており, その特徴は処理途中の動作に複数個の可能性がある場合は, 可能性のある複数個の動作をいずれも実行する点である. 一般にGLR構文解析は自然言語を対象としているが, 文脈依存言語 [4] 以上のクラスの言語に対しては, 明らかに適用することができない.

しかし, 自然言語処理においては文脈自由文法よりも広いクラスの言語を構文解析する必要がしばしばある.

そこで 本論文では, 構文解析を行なう対象とする言語のクラスを句構造文法 [3] の一つである unrestricted 文法 [17] が生成する言語のクラスとし, その言語を構文解析する方法としてGLR構文解析を拡張した$UGLR$構文解析 (unrestricted generalized LR 構文解析 の 略) を新たに提案する. また, この$UGLR$構文解析器が受理できる言語のクラスが文脈自由言語のクラスを含むことを, $UGLR$構文解析に与える文法に制限をつけると, $UGLR$構文解析の動作が Earley 法と一致することによって明らかに

する.

5.2 $UGLR$構文解析

$UGLR$構文解析器の構成および動作について述べる.

5.2.1 $UGLR$構文解析器の構成

本節では, 受理機械として$UGLR$構文解析器 $M = (Q, \Sigma, \Pi, \Gamma, \delta, q_0, q_{acc}, \$)$ を定義し, 各要素を以下のように定義する.

1. Q: $ELR(0)$ 状態遷移図の状態の有限集合. $ELR(0)$ 状態遷移図は, **第 5.2.2 節**で定義する.

2. 入力列の集合 $\Sigma(\subset T^*)$.

3. 状態リスト$\Pi = \pi_0\pi_1 \ldots \pi_n$ は以下に示すリストである.

   - $\pi_i = \{ (q_1, i, 1, \alpha_i, x_1), (q_2, i, 2, \alpha_i, x_2), \ldots, (q_m, i, m, \alpha_i, x_m)\}$, $(q_i, i, j, \alpha_i, x_j) \in \pi_i$ に対して, $q_j \in Q$, $i, j \in Z^+$(0 以上の整数の集合), $\alpha_i \in (T \cup N)$, $x_j =$ t または f, 第 2 要素 i, 第 3 要素 j の組合せは, Q のある状態から文法記号によって遷移してきた先の状態を指すポインタを表す. このポインタは状態リストΠ中の集合π_iの位置 i と集合π_i中の要素の位置 jを表す.
   - 文法の終端記号 Tと非終端記号 Nの和集合 $(T \cup N)$ を文法記号として定義する.
   - フラグ t は一つ手前の入力でその要素が処理済みであることを表し, フラグ f は一つ手前の入力でその要素が処理済みでないことを表す.

4. スタックΓは, 文法記号, 構文解析の動作, 生成規則, および, 生成規則のどの文法記号を処理しているのかを示すフラグの 4 つ組からなるリストで, リストの各要素 (α, x, p, y) は以下のように示される.

$$(\alpha, x, p, y) \in (T \cup N) \times \{\text{shift}, \text{reduce}\} \times P \times \{\text{RIGHT, Not\_RIGHT, LEFT, Not\_LEFT}\}.$$

Γの要素中のフラグ RIGHT, Not_RIGHT, LEFT, Not_LEFT は生成規則中の第一要素の文法記号の位置を表す. それぞれ RIGHT は生成規則の左側の最右の文法記号, Not_RIGHT は 生成規則の左側の最右でない文法記号, LEFT は 生成規則の右側の最左の文法記号, および, Not_LEFT は 生成規則の右側の最左でない文法記号を示す. また, Pは生成規則の集合を表す.

5. 遷移関数δ は以下に示すシフト動作の遷移関数δ_sとレデュース動作の遷移関数δ_rの和 ($\delta = \delta_s \cup \delta_r$) からなり, $\delta_s(q,\alpha)$=$\{(\mathrm{Shift}, q') | q' \in Q, \alpha \in (T \cup N) \cup \{\varepsilon\}\}$, $\delta_r(q) = \{(\mathrm{Reduce}, \lambda, \mu) | \lambda \to \mu \in P\}$ である.

6. 初期状態 $q_0 (\in Q)$.

7. 受理状態 $q_{acc} (\in Q)$.

8. 入力列の終りを表す特殊記号 \$.

5.2.2 $ELR(0)$ 状態遷移図

$UGLR$構文解析で用いる $ELR(0)$ 状態遷移図 (Extended $LR(0)$ 状態遷移図 の略) は, LR構文解析 および GLR構文解析で用いられる $LR(0)$ 状態遷移図 [4, 8, 10, 11, 18, 21, 24] を unrestricted 文法 [17] 用に拡張したものである.

文法 G=(P, S, T, N), P:生成規則の集合, S:開始記号, T:終端記号の集合, N:非終端記号の集合 に対して $LR(0)$ 項と呼ばれる 生成規則$\lambda \to \mu_1\mu_2$ にメタ記号 $\cdot$ をつけた形式 $[\lambda \to \mu_1 \cdot \mu_2]$ ($\lambda \in (T \cup N)^+$, $\mu_1, \mu_2 \in (T \cup N)^*$, $(T \cup N)^+$は Tまたは Nの 1 回以上の繰り返し, $(T \cup N)^*$は Tまたは Nの 0 回以上の繰り返しを表す) を要素に持つ状態と状態間の遷移を作ることによって定義される. 以下に $ELR(0)$ 状態遷移図の作成法を示す.

[作成 1] ダミー記号 S'を用意する. 次に $S' \to S$を Pに追加して $LR(0)$ 項 $[S' \to \cdot S]$ を状態 $q_0 (\in Q)$ の要素とする.

[作成 2] $LR(0)$ 項 $[\lambda \to \mu_1 \cdot \alpha\mu_2]$ が状態 $q_i \in Q$ に存在するなら, 生成規則$\alpha\zeta \to \eta \in P$ の形式を持つ生成規則に対応するすべての $LR(0)$ 項 $[\alpha\zeta \to \cdot\eta]$ を状態 q_iに追加する ($\alpha \in (T \cup N)$, $\lambda \in (T \cup N)^+$, $\mu_1, \mu_2, \zeta, \eta \in (T \cup N)^*$).

[作成 3] $LR(0)$ 項 $[\lambda \to \mu_1 \cdot \alpha\mu_2]$ が状態 $q_i \in Q$ に存在するなら, 新たに状態 q_jを用意し, 状態 q_jが $LR(0)$ 項 $[\lambda \to \mu_1\alpha \cdot \mu_2]$ から成るとする.

[作成 4] 状態 $q_i \in Q$ に $LR(0)$ 項 $[\lambda \to \mu_1 \cdot \alpha\mu_2]$, $\alpha \in (T \cup N)$ があり, 状態 $q_j \in Q$ に $LR(0)$ 項 $[\lambda \to \mu_1\alpha \cdot \mu_2]$ があるなら, αによる状態 q_iから状態 q_jへのシフト動作の遷移を追加する ($\delta_s(q_i, \alpha) := \delta_s(q_i, \alpha) \cup \{(\mathrm{Shift}, q_j)\}$, $\lambda \in (T \cup N)^+$, $\mu_1, \mu_2 \in (T \cup N)^*$, $\alpha \in (T \cup N)$).

[作成 5] 状態 $q_i \in Q$ に $LR(0)$ 項 $[\lambda \to \mu \cdot]$ があるなら, 状態 q_iにおいて, レデュース動作の遷移を追加する ($\delta_r(q_i) := \delta_r(q_i) \cup \{(\mathrm{Reduce}, \lambda, \mu)\}$, $\lambda \in (T \cup N)^+$, $\mu \in (T \cup N)^*$).

5.2.3 $UGLR$構文解析の動作

$UGLR$構文解析の各時点での状況を (状態リスト, 入力列) とし, 1 ステップごとの変化を $\vdash$ で表し, 0 ステップ以上の変化を $\vdash\!\!\!\!\!\!+$ で表す.

上記の状況変化を用いて$UGLR$構文解析の動作 [**動作 1**]〜[**動作 5**] を次に示す.

[動作 1] 初期設定

入力列を $a_1 a_2 \ldots a_n$とする時,

- $UGLR$構文解析の状況を $(\{(q_0, 1, 1, \varepsilon, \mathrm{t})\}, a_1 a_2 \ldots a_n\$)$ と設定する ($\pi_0 := (\{(q_0, 1, 1, \varepsilon, \mathrm{t})\})$).
- スタック $\Gamma := \phi$(空リスト) として, スタックを空にする.

[動作 2] shift 動作 1

ある時点での$UGLR$構文解析の状況を $(\Pi, a_i a_{i+1} \ldots a_n\$)$ として,

$\Pi = \pi_0 \pi_1 \ldots \pi_{i-1}$,

$\pi_{i-1} =$
$\{ (q_1, prev\_list(i-1,1), prev\_node(i-1,1), prev\_symbol(i-1,1), \mathrm{t}),$
$(q_2, prev\_list(i-1,2), prev\_node(i-1,2), prev\_symbol(i-1,2), \mathrm{t}),$
$\ldots,$
$(q_j, prev\_list(i-1,j), prev\_node(i-1,j), prev\_symbol(i-1,j), \mathrm{t}),$
$\ldots,$
$(q_m, prev\_list(i-1,m), prev\_node(i-1,m), prev\_symbol(i-1,m), \mathrm{t})\}$,
Π中のπ_{i-1}の位置を表す $prev\_list(i-1,j) \in Z^+$(0 以上の整数の集合), π_{i-1}中の要素の位置を表す $prev\_node(i-1,j) \in Z$, $q_j \in Q$, 遷移に用いた文法記号を表す $prev\_symbol(i-1,j) \in (T \cup N)$, $\delta_s(q_j, a_i) = (\mathrm{Shift}, q'_j)$, $j = 1, 2, \ldots, m$ とする時,

- 状態リスト Π に新たに 以下に示すπ_iを追加する.

$$\pi_i = \{(q'_1, i, 1, a_i, \mathrm{f})), (q'_2, i, 2, a_i, \mathrm{f}), \ldots, (q'_m, i, m, a_i, \mathrm{f})\}$$

- $UGLA$ 構文解析の状況は次のように変化させる.

$$\begin{array}{ll} & (\Pi, a_i a_{i+1} a_{i+2} \ldots a_n\$) \\ \vdash & (\pi_0 \pi_1 \ldots \pi_i \{(q'_1, i, 1, a_i, \mathrm{f})), (q'_2, i, 2, a_i, \mathrm{f}), \ldots, \\ & (q'_m, i, m, a_i, \mathrm{f})\}, a_{i+1} a_{i+2} \ldots a_n\$) \end{array}$$

[動作 3] shift 動作 2

ある時点での$UGLR$構文解析の状況を $(\Pi, a_i a_{i+1} \ldots a_n\$)$ として, $\Pi = \pi_0 \pi_1 \ldots \pi_i$, (q_j,$prev\_list(i,j)$,$prev\_node(i,j)$, $prev\_symbol(i,j)$,f) $\in \pi_i$に対して $q_j \in Q_{push}$, $q_j \notin Q_{pop}$であるとする時,

- 状態リストπ_iの要素 $(q_j, prev\_list(i,j), prev\_node(i,j), prev\_symbol(i,j),$ f) を $(q_j,\ prev\_list(i,j),\ prev\_node(i,j),\ prev\_symbol(i,j),$ t) とする.

[動作 4] reduce 動作

ある時点での$UGLR$構文解析の状況を $(\Pi, a_i a_{i+1} \ldots a_n\$)$ として, $\Pi = \pi_0 \pi_1 \ldots \pi_i$, $(q_j, prev\_list(i,j), prev\_node(i,j), prev\_symbol(i,j)$,f) $\in \pi_i$に対して

$q_j \in Q_{pop}$, $LR(0)$ 項 $[\lambda \to \mu\cdot] \in q_j$, $\lambda = \alpha_1 \alpha_2 \ldots \alpha_{|\lambda|}$, $\alpha_1, \alpha_2, \ldots, \alpha_{|\lambda|} \in (T \cup N)$, $\mu = \beta_1 \beta_2 \ldots \beta_{|\mu|}$, $\beta_1, \beta_2, \ldots, \beta_{|\mu|} \in (T \cup N)$ であるとする時, 但し, $|\mu| = 0$ の時は, $|\mu| = 1$, $\beta_1 = \varepsilon$とする,

- スタックΓに $(\beta_1$, reduce, $\lambda \to \mu$, LEFT), (β_2,reduce,$\lambda \to \mu$, Not_LEFT), ..., ($\beta_{|\mu|}$,reduce,$\lambda \to \mu$, Not_LEFT) の順に積む.
- 状態リストπ_iの要素 $(q_j, prev_list(i,j), prev_node(i,j), prev_symbol(i,j),$f) を $(q_j, prev\_list(i,j), prev\_node(i,j), prev\_symbol(i,j),$ t) とする.
- $k := i$, $l := j$とする.
- スタックΓがϕ(空) になるまで, 以下に示す **[CASE 1]** から **[CASE 4]** を繰り返す. スタックΓが空でない内に **[CASE 1]** から **[CASE 4]** のいずれの処理も実行できない時は **[動作 5]** に処理を移し異常終了する.

[**CASE 1**] スタックΓのトップの文法記号 $\mathrm{TOP}_{\mathrm{symbol}} = prev\_symbol(q_l)$, スタックΓのトップの動作 $\mathrm{TOP}_{\mathrm{action}} = \mathrm{reduce}$, スタックΓのトップのフラグが $\mathrm{TOP}_{\mathrm{flg}} = \mathrm{Not\_LEFT}$ であるならば,

- スタックΓから要素を一つ取り除く.
- $k := prev\_list(k,l)$, $l := prev\_node(k,l)$ とする.

[**CASE 2**] スタックΓのトップの文法記号 $\mathrm{TOP}_{\mathrm{symbol}} = prev\_symbol(q_l)$, スタックΓのトップの動作 $\mathrm{TOP}_{\mathrm{action}} = \mathrm{reduce}$, スタックΓのトップのフラグが $\mathrm{TOP}_{\mathrm{flg}} = \mathrm{LEFT}$ であるならば,

- スタックΓから要素を一つ取り除く.
- スタックΓに $(\alpha_{|\lambda|}, \mathrm{shift}, \lambda \rightarrow \mu, \mathrm{Not\_RIGHT})$, $(\alpha_{|\lambda-1|}, \mathrm{shift}, \lambda \rightarrow \mu, \mathrm{Not\_RIGHT})$, ..., $(\alpha_1, \mathrm{shift}, \lambda \rightarrow \mu, \mathrm{RIGHT})$ の順で積む.
- $k := prev\_list(k,l)$, $l := prev\_node(k,l)$ とする.

[**CASE 3**] スタックΓのトップの動作 $\mathrm{TOP}_{\mathrm{action}} = \mathrm{shift}$, $\delta_s(q_k, \mathrm{TOP}_{\mathrm{symbol}}) = (\mathrm{Shift}, q')$, $q' \in Q_{push}$, $LR(0)$ 項 $[\lambda \rightarrow \mu\cdot] \notin q'$, $\lambda = \alpha_1\alpha_2 \ldots \alpha_{|\lambda|}$, α_1, α_2, ..., $\alpha_{|\lambda|} \in (T \cup N)$, $\mu = \beta_1\beta_2 \ldots \beta_{|\mu|}$, $\beta_1, \beta_2, \ldots, \beta_{|\mu|} \in (T \cup N)$ であるならば,

- $\pi_i := \pi_i \cup \{(q', k, l, \mathrm{TOP}_{\mathrm{symbol}}, \mathrm{t})\}$,
- $k := i$, $l := |\pi_i| + 1$ とする. $|\pi_i|$ はπ_iの要素数を表す.

[**CASE 4**] スタックΓのトップの動作 $\mathrm{TOP}_{\mathrm{action}} = \mathrm{shift}$, $\delta_s(q_k, \mathrm{TOP}_{\mathrm{symbol}}) = (\mathrm{Shift}, q')$, $q' \in Q_{pop}$ $LR(0)$ 項 $[\lambda \rightarrow \mu\cdot] \in q'$, $\lambda = \alpha_1\alpha_2 \ldots \alpha_{|\lambda|}$, α_1, α_2, ..., $\alpha_{|\lambda|} \in (T \cup N)$, $\mu = \beta_1\beta_2 \ldots \beta_{|\mu|}$, $\beta_1, \beta_2, \ldots, \beta_{|\mu|} \in (T \cup N)$ であるならば,

- $\pi_i := \pi_i \cup \{(q', k, l, \mathrm{TOP}_{\mathrm{symbol}}, \mathrm{t})\}$,
- スタックΓに $(\beta_1, \mathrm{reduce}, \lambda \rightarrow \mu, \mathrm{LEFT})$, $(\beta_2, \mathrm{reduce}, \lambda \rightarrow \mu, \mathrm{Not\_LEFT})$, ..., $(\beta_{|\mu|}, \mathrm{reduce}, \lambda \rightarrow \mu, \mathrm{Not\_LEFT})$ の順に積む.
- $k := i$, $l := |\pi_i| + 1$ とする.

[動作 5] 終了動作

$UGLR$構文解析の状況が $(\Pi, \$)$ となった時, 入力列が与えられた文法で受理できることを示し, 正常終了する. また, $UGLR$構文解析の状況が **[動作 1]** から **[動作 4]** の条件に当てはまらない場合は, 入力列が与えられた文法で受理できないことを示し, 異常終了する.

5.3 $UGLR$構文解析の性質

$UGLR$構文解析器 Mによって受理される言語に対する性質を述べる. ここで,

$$UGLR = \{L(M) | M \text{は} UGLR\text{構文解析器}\}$$

とする.

5.3.1 $UGLR \supseteq$ 文脈自由言語のクラス

(定理 1) $UGLR \supseteq$ **文脈自由言語のクラス**.

(証明) $UGLR$構文解析に対して与えられる文法 G の生成規則$\lambda \to \mu$ が制限 $\lambda = X \in N$, $|\lambda| = 1$ を満たすものとすると $UGLR$構文解析の処理が Earley 法 [2, 18, 21] と同じ処理をしていることを示すことにより, $UGLR \supseteq$ **文脈自由言語のクラス**であることを示す.

そこで, 文脈自由文法 $G = (P, S, T, N)$ と入力列 $a_1 a_2 \ldots a_n$ に対する Earley 法による構文解析の動作を次に示す.

Earley 法

[E.1] $S \to \mu$, $\mu \in (T \cup N)^*$の形を持つ文法規則すべてに対して, $I_0 := I_0 \cup \{[S \to \cdot\mu, 0]\}$.

[E.1.1] $j = 0$ としてクロージャを作成する手続き CLOSURE(I_0) を実行する.

[E.2] $j = 1, 2, \ldots, n$ に対して, パーステーブル I_jを作成する.

[E.2.1] I_{j-1}で $[A \to \mu_1 \cdot a_i \mu_2, i]$ の形を持つアイテムから $[A \to \mu_1 a_i \cdot \mu_2, i]$ を作成し, $I_j := I_j \cup [A \to \mu_1 a_i \cdot \mu_2, i]$.

[**E.2.2**] クロージャを作成する手続き CLOSURE(I_j) を実行する. □

手続き CLOSURE(I_j)

次の処理を I_jに変更がなくなるまで繰り返す.

[**C.1**] $[A \to \mu_1 \cdot B\mu_2] \in I_j$かつ $B \to \mu_3 \in P$ ならば, $I_j := I_j \cup \{[B \to \cdot\mu_3, j]\}$.

[**C.2**] $[A \to \mu\cdot, i] \in I_j$ かつ $[B \to \mu_1 \cdot A\mu_2, k] \in I_i$ ならば, $I_j := I_j \cup \{[B \to \mu_1 A \cdot \mu_2, k]\}$. □

(初期段階 i) *UGLR*構文解析の [**動作 1**] で状態リストΠに対して$\pi_1 = \{(q_0, 1, 1, \varepsilon, \mathrm{t})\}$ を初期設定している. ここで q_0の要素は $[S' \to \cdot S]$ から作られるクロージャである.

一方, [**E.1**] で作成される I_0中のアイテム $[A \to \mu_1 \cdot B\mu_2, i]$ からポインタ i を削除した部分を I'_0とする.

よって,

$$q_0 = I'_0 \cup \{[S' \to \cdot S]\}$$

である.

(帰納段階 ii.a) $i - 1 > 1$ において, *UGLR*構文解析の状態リストΠ中の π_{i-1}中の状態の和を

$$\pi'_{i-1} := \bigcup_{(q_j, k, l, \alpha, \mathrm{t}) \in \pi_{i-1}} q_j,$$

$q_j \in Q, \alpha \in (T \cup N), k, l \in Z$とし, Earley 法で作成されるパーステーブル I_{i-1}から時点表示を除いた部分を I'_{i-1}とする時, I'_{i-1}と π'_{i-1}が等しいと仮定する.

(帰納段階 ii.b) Earley 法で入力 a_{i-1}を処理した後入力 a_iを処理する場合, [**E.2.1**], [**E.2.2**] でパーステーブル I_iが作成される.

一方, *UGLR*構文解析では入力列 a_{i-1}を処理した後, 入力 a_iを処理し, 入力 a_{i+1}を処理する手前までには, [**動作 2**], [**動作 3**], [**動作 4**] を実行する.

[動作 2] では, π_{i-1}中の状態 qから入力 a_iによって遷移する先の状態 q'_jを求めてπ_iとしている.

状態リストπ_i中の状態 q'_j全ての和を

$$\pi'_i := \bigcup_{(q'_j,k,l,a_i,\mathrm{t})\in\pi_i} q'_j$$

とすると, π'_iは Earley 法の **[E.2.1]** で作られるパーステーブル I_iと等しいことは明らかである.

[動作 3] は, 状態リストπ_i中の要素の状態を変化させないので, この証明では不要.

[動作 4] では, $(q_j,k,l,\beta_{|\mu|},\mathrm{f}) \in \pi_i$ の q_jに対して $[\lambda \to \mu\cdot] \in q_j$, $\lambda = X_1(\in N)$,$|\lambda| = 1$, $\mu = \beta_1\beta_2\ldots\beta_{|\mu|}$であるならば, **[CASE 1]** を $|\mu|$ 回実行し, $\beta_{|\mu|}\beta_{|\mu|-1}\ldots\beta_1$ の順に状態リストΠ上をたどり, 要素 $(q'_j,k,l,\alpha,\mathrm{t})$ に戻るものとする. 次に, **[CASE 2]** を実行する. ここで, $|\lambda| = 1$ であるので, スタックΓには $(X_1,\mathrm{shift},\lambda\to\mu,\mathrm{RIGHT})$ のみが積まれる. スタックΓに 1 つのデータを積んだ結果, **[CASE 3]** または **[CASE 4]** が実行される. $ELR(0)$ 状態遷移図の遷移関数$\delta_s(q'_j,X_1) = (\mathrm{shift},q''_j)$ とすると, **[CASE 3]** または **[CASE 4]** のいずれの場合でも状態リストΠのπ_iに $(q''_j,k,l,X_1,\mathrm{f})$ が追加される. また, $ELR(0)$ 状態遷移図の作成法から $[\lambda_1 \to \mu_1 X\cdot\mu_2]$ $\in q''_j$, $\lambda_1,\mu_1,\mu_2 \in (T\cup N)^+$であり, q''_jの要素は $[\lambda_1\to\mu_1X\cdot\mu_2]$ のクロージャである.

Earley 法における **[E.2.2]** から呼び出される**手続き CLOSURE**(I_j) の **[C.2]** を実行し, $[\lambda_1\to\mu_1X\cdot\mu_2,k]$ を I_iに追加する. その後, I_iに変更がなくなるまで **[C.1]** の順で処理を行ない, $[\lambda_1\to\mu_1X\cdot\mu_2,k]$ のクロージャを I_iに追加している.

なお, **手続き CLOSURE** で **[C.1]** を実行し **[C.2]** を実行しない時は, $UGLR$構文解析の **[動作 4]** の **[CASE 3]** を実行するのと同じである. また, **手続き CLOSURE** で **[C.1]** を実行し **[C.2]** を実行する時は, $UGLR$構文解析の **[動作 4]** の **[CASE 4]** を実行するのと同じである.

よって,

> $UGLR$構文解析で作成されるπ'_i =
> Earley 法で作成する I_iから時点表示を除いた部分 I'_i

である.

以上の帰納法から$UGLR$構文解析に対して与えられる文法 G の生成規則$\lambda\to\mu$が制限 $\lambda = X_1 \in N$, $|\lambda| = 1$ を満たすものとすると, $UGLR$

構文解析の処理が Earley 法と同じ処理をしていることが明らかとなり, $|\lambda| \geq 1$ であるなら $UGLR \supseteq$ **文脈自由言語のクラス**であることも明らかとなる. □

また, 構文解析の対象とする言語ごとの $UGLR$構文解析の時間計算量 [4] には 変更はなく, 対象とする言語を文脈自由言語とすると, 入力文字列長 n に対して $O(n^3)$ 時間 [4] 必要となる.

5.3.2 $UGLR \supseteq$ 文脈依存言語のクラス

(定理 2) $UGLR \supseteq$ **文脈依存言語のクラス**.

(証明) $UGLR$構文解析に対して与えられる文法 G の生成規則$\lambda \rightarrow \mu$ が制限 $|\lambda| \leq |\mu|$ を満たすものとすると$UGLR$構文解析の処理が Vold'man の方法 [5] と同じ処理をしていることを示すことにより, $UGLR \supseteq$ **文脈依存言語のクラス** であることを示すことができる. □

また, 構文解析の対象とする言語ごとの $UGLR$構文解析の領域計算量 [4] には 変更はなく, $UGLR$構文解析が対象とする言語を文脈依存言語とすると, 多項式領域困難 [1] となる.

5.4 $UGLR$構文解析の動作例

本章では文脈自由文法の例を用いて, $UGLR$構文解析の動作を例示する.

文法 $G_{ex} = (\{ S \rightarrow NP\ VP, S \rightarrow S\ PP, S \rightarrow S\ and\ S, NP \rightarrow n, NP \rightarrow det\ n, NP \rightarrow NP\ PP, NP \rightarrow NP\ and\ NP, VP \rightarrow v\ NP, VP \rightarrow v\ S, VP \rightarrow v\ S, PP \rightarrow p\ NP \}, S, \{n, det, v, and, p\}, \{S, NP, VP, PP\})$ と文 "*I know Jane and Jack knew it.*" を元にした入力列 "*n v n and n v n*" を$UGLR$構文解析の例として示す. ここで, 文の各単語は次のように変換されるものとする.

$$
\begin{aligned}
&I \Rightarrow n,\\
&know \Rightarrow v,\\
&Jane \Rightarrow n,\\
&and \Rightarrow and,\\
&Jack \Rightarrow n,\\
&knew \Rightarrow v,\\
&it \Rightarrow n.
\end{aligned}
$$

まず, 文法 G_{ex}から得られる $ELR(0)$ 状態遷移図を図 5.1に示す.

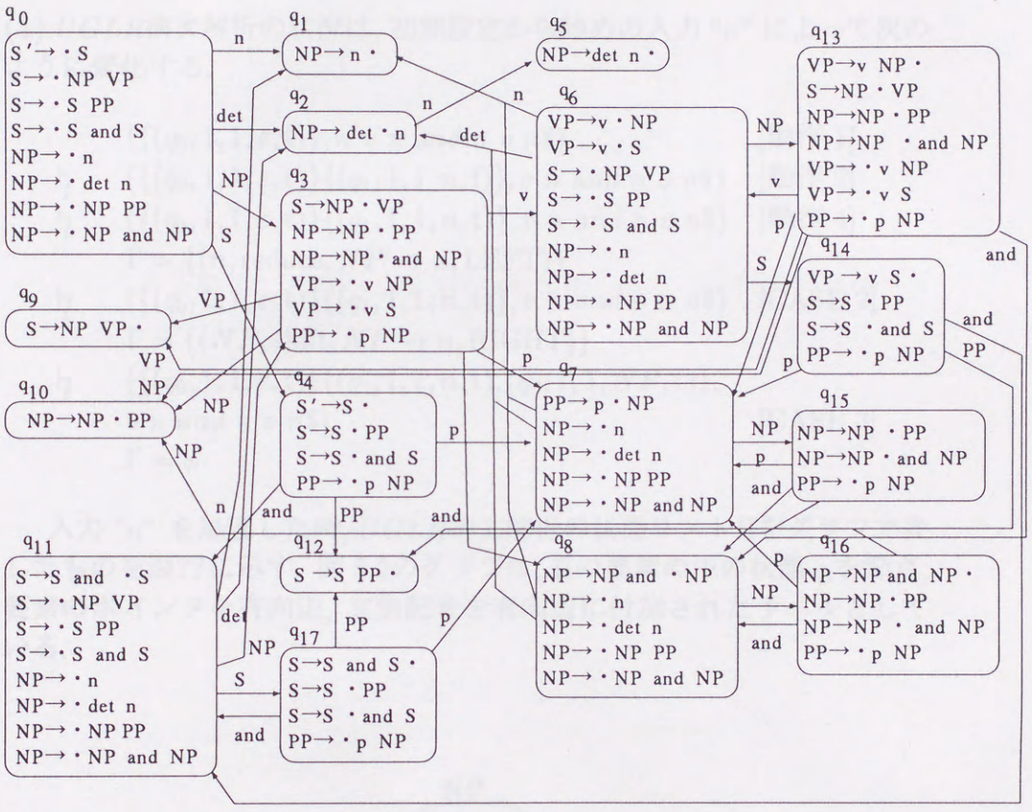


図 5.1: 文法 G_{ex}に対する $ELR(0)$ 状態遷移図.

Fig. 5.1: An LR transition corresponding to G_{ex}.

入力列 "$n\ v\ n\ and\ n\ v\ n$" を$UGLR$構文解析によって処理する時, $UGLR$構文解析の状況は以下のように変化する. 但し, スタックΓが省略してある時は, $\Gamma = \phi$(空) であるものとする.

(1) $UGLR$構文解析の状況は, 初期設定から始めの入力 "n" によって次のように変化する.

$$
\begin{array}{ll}
 & (\{(q_0,1,1,\varepsilon,\mathrm{t})\}, n\ v\ n\ and\ n\ v\ n\$) \quad \text{[動作 1]} \\
\vdash & (\{(q_0,1,1,\varepsilon,\mathrm{t})\}\{(q_1,1,1,n,\mathrm{f})\}, v\ n\ and\ n\ v\ n\$) \quad \text{[動作 2]} \\
\vdash & (\{(q_0,1,1,\varepsilon,\mathrm{t})\}\{(q_1,1,1,n,\mathrm{t})\}, v\ n\ and\ n\ v\ n\$) \quad \text{[動作 4]} \\
 & \Gamma = \{(n, \mathrm{reduce}, NP \rightarrow n, \mathrm{LEFT})\} \\
\vdash & (\{(q_0,1,1,\varepsilon,\mathrm{t})\}\{(q_1,1,1,n,\mathrm{t})\}, v\ n\ and\ n\ v\ n\$) \quad \text{[CASE 2]} \\
 & \Gamma = \{(NP, \mathrm{shift}, NP \rightarrow n, \mathrm{RIGHT})\} \\
\vdash & (\{(q_0,1,1,\varepsilon,\mathrm{t})\}\{(q_1,1,1,n,\mathrm{t}),(q_3,1,1,NP,\mathrm{t})\}, \\
 & v\ n\ and\ n\ v\ n\$) \quad \text{[CASE 3]} \\
 & \Gamma = \phi
\end{array}
$$

入力 "n" を処理した時, $UGLR$構文解析の状態リストΠをグラフで表したものを図??に示す. 図 5.2のグラフは, Πの要素の中の状態 q_iを節点, 要素のポインタを有向辺, 文法記号を有向辺に付加されたラベルとしている.

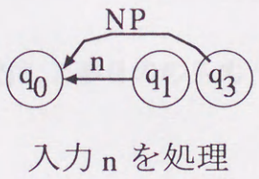


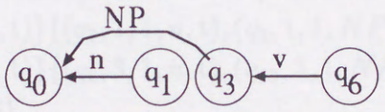
図 5.2: 状態リストΠの変化.

Fig. 5.2: An example of transitions of state list Π.

(2) *UGLR*構文解析の状況は, 2 番目の入力 "v" によって次のように変化する.

$$
\begin{aligned}
& (\{(q_0,1,1,\varepsilon,\mathrm{t})\}\{(q_1,1,1,n,\mathrm{t}),(q_3,1,1,NP,\mathrm{t})\}, \\
& \ v\ n\ and\ n\ v\ n\$) \\
\vdash\!\!\!| \quad & (\{(q_0,1,1,\varepsilon,\mathrm{t})\}\{(q_1,1,1,n,\mathrm{t}),(q_3,1,1,NP,\mathrm{t})\} \\
& \ \{(q_6,2,2,v,\mathrm{f})\},n\ and\ n\ v\ n\$) \qquad \text{[動作 2]} \\
\vdash\!\!\!| \quad & (\{(q_0,1,1,\varepsilon,\mathrm{t})\}\{(q_1,1,1,n,\mathrm{t}),(q_3,1,1,NP,\mathrm{t})\} \\
& \ \{(q_6,2,2,v,\mathrm{t})\},n\ and\ n\ v\ n\$) \qquad \text{[動作 3]}
\end{aligned}
$$

入力 "v" を処理した時, *UGLR*構文解析の状態リストΠをグラフで表したものを図 5.3 に示す.



入力 v を処理

図 5.3: 状態リストΠの変化.

Fig. 5.3: An example of transitions of state list Π.

(3) *UGLR*構文解析の状況は, 3 番目のの入力 "n" によって次のように変化する.

$(\{(q_0,1,1,\varepsilon,\mathrm{t})\}\{(q_1,1,1,n,\mathrm{t}),(q_3,1,1,NP,\mathrm{t})\}$
$\{(q_6,2,2,v,\mathrm{t})\},n\ and\ n\ v\ n\$)$

$\vdash_l$ $(\{(q_0,1,1,\varepsilon,\mathrm{t})\}\{(q_1,1,1,n,\mathrm{t}),(q_3,1,1,NP,\mathrm{t})\}$
$\{(q_6,2,2,v,\mathrm{t})\}\{(q_1,3,1,n,\mathrm{f})\},and\ n\ v\ n\$)$ [動作 2]

$\vdash_l$ $(\{(q_0,1,1,\varepsilon,\mathrm{t})\}\{(q_1,1,1,n,\mathrm{t}),(q_3,1,1,NP,\mathrm{t})\}$
$\{(q_6,2,2,v,\mathrm{t})\}\{(q_1,3,1,n,\mathrm{t})\},and\ n\ v\ n\$)$ [動作 4]

$\Gamma=\{(n,\mathrm{reduce},NP\rightarrow n,\mathrm{LEFT})\}$

$\vdash_l$ $(\{(q_0,1,1,\varepsilon,\mathrm{t})\}\{(q_1,1,1,n,\mathrm{t}),(q_3,1,1,NP,\mathrm{t})\}$
$\{(q_6,2,2,v,\mathrm{t})\}\{(q_1,3,1,n,\mathrm{t})\},and\ n\ v\ n\$)$ [CASE 2]

$\Gamma=\{(NP,\mathrm{shift},NP\rightarrow n,\mathrm{RIGHT})\}$

$\vdash_l$ $(\{(q_0,1,1,\varepsilon,\mathrm{t})\}\{(q_1,1,1,n,\mathrm{t}),(q_3,1,1,NP,\mathrm{t})\}$
$\{(q_6,2,2,v,\mathrm{t})\}\{(q_1,3,1,n,\mathrm{t}),(q_{13},3,1,NP,\mathrm{t})\},$
$and\ n\ v\ n\$)$ [CASE 4]

$\Gamma=\{(v,\mathrm{reduce},VP\rightarrow v\ \ NP,\mathrm{LEFT}),$
$(NP,\mathrm{reduce},VP\rightarrow v\ NP,\mathrm{Not\_LEFT})\}$

$\vdash_l$ $(\{(q_0,1,1,\varepsilon,\mathrm{t})\}\{(q_1,1,1,n,\mathrm{t}),(q_3,1,1,NP,\mathrm{t})\}$
$\{(q_6,2,2,v,\mathrm{t})\}\{(q_1,3,1,n,\mathrm{t}),(q_{13},3,1,NP,\mathrm{t})\},$
$and\ n\ v\ n\$)$ [CASE 1]

$\Gamma=\{(v,\mathrm{reduce},VP\rightarrow v\ NP,\mathrm{LEFT})\}$

$\vdash_l$ $(\{(q_0,1,1,\varepsilon,\mathrm{t})\}\{(q_1,1,1,n,\mathrm{t}),(q_3,1,1,NP,\mathrm{t})\}$
$\{(q_6,2,2,v,\mathrm{t})\}\{(q_1,3,1,n,\mathrm{t}),(q_{13},3,1,NP,\mathrm{t})\},$
$and\ n\ v\ n\$)$ [CASE 2]

$\Gamma=\{(VP,\mathrm{shift},VP\rightarrow v\ \ NP,\mathrm{RIGHT})\}$

$\vdash_l$ $(\{(q_0,1,1,\varepsilon,\mathrm{t})\}\{(q_1,1,1,n,\mathrm{t}),(q_3,1,1,NP,\mathrm{t})\}$
$\{(q_6,2,2,v,\mathrm{t})\}\{(q_1,3,1,n,\mathrm{t}),(q_{13},3,1,NP,\mathrm{t}),$
$(q_9,2,2,VP,\mathrm{t})\},and\ n\ v\ n\$)$ [CASE 4]

$\Gamma=\{(NP,\mathrm{reduce},S\rightarrow VP\ NP,\mathrm{LEFT}),$
$(VP,\mathrm{reduce},S\rightarrow VP\ \ NP,\mathrm{Not\_LEFT})\}$

$\vdash_l$ $(\{(q_0,1,1,\varepsilon,\mathrm{t})\}\{(q_1,1,1,n,\mathrm{t}),(q_3,1,1,NP,\mathrm{t})\}$
$\{(q_6,2,2,v,\mathrm{t})\}\{(q_1,3,1,n,\mathrm{t}),(q_{13},3,1,NP,\mathrm{t}),$
$(q_9,2,2,VP,\mathrm{t})\},and\ n\ v\ n\$)$ [CASE 1]

$\Gamma=\{(NP,\mathrm{reduce},S\rightarrow VP\ \ NP,\mathrm{LEFT})\}$

$\vdash_l$ $(\{(q_0,1,1,\varepsilon,\mathrm{t})\}\{(q_1,1,1,n,\mathrm{t}),(q_3,1,1,NP,\mathrm{t})\}$
$\{(q_6,2,2,v,\mathrm{t})\}\{(q_1,3,1,n,\mathrm{t}),(q_{13},3,1,NP,\mathrm{t}),$
$(q_9,2,2,VP,\mathrm{t}),(q_4,1,1,VP,\mathrm{t})\},$
$and\ n\ v\ n\$)$ [CASE 3]

$\Gamma=\phi$

入力 "n" を処理した時, $UGLR$構文解析の状態リストΠをグラフで表したものを図5.4に示す.

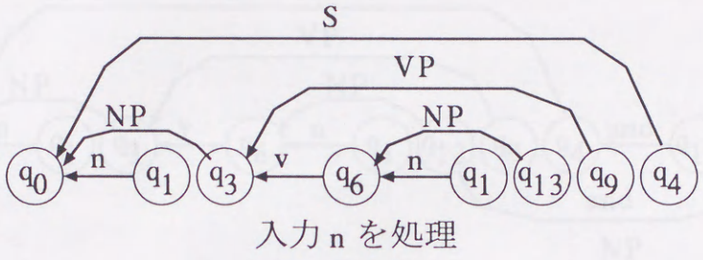


図 5.4: 状態リストΠの変化.

Fig. 5.4: An example of transitions of state list Π.

(4) $UGLR$構文解析の状況は, 4番目のの入力 "and" によって次のように変化する.

$$
\begin{aligned}
&(\{(q_0,1,1,\varepsilon,\mathrm{t})\}\{(q_1,1,1,n,\mathrm{t}),(q_3,1,1,NP,\mathrm{t})\}\\
&\{(q_6,2,2,v,\mathrm{t})\}\{(q_1,3,1,n,\mathrm{t}),(q_{13},3,1,NP,\mathrm{t}),\\
&(q_9,2,2,VP,\mathrm{t}),(q_4,1,1,VP,\mathrm{t})\},\\
&and\ n\ v\ n\$)\\
\vdash\quad&(\{(q_0,1,1,\varepsilon,\mathrm{t})\}\{(q_1,1,1,n,\mathrm{t}),(q_3,1,1,NP,\mathrm{t})\}\\
&\{(q_6,2,2,v,\mathrm{t})\}\{(q_1,3,1,n,\mathrm{t}),(q_{13},3,1,NP,\mathrm{t}),\\
&(q_9,2,2,VP,\mathrm{t}),(q_4,1,1,VP,\mathrm{t})\}\{(q_8,4,2,and,\mathrm{f}),\\
&(q_{11},4,4,and,\mathrm{f})\}\ \mathrm{n}\ \ \mathrm{v}\ \ \mathrm{n}\$) \qquad \text{[CASE 1]}\\
\vdash\quad&(\{(q_0,1,1,\varepsilon,\mathrm{t})\}\{(q_1,1,1,n,\mathrm{t}),(q_3,1,1,NP,\mathrm{t})\}\\
&\{(q_6,2,2,v,\mathrm{t})\}\{(q_1,3,1,n,\mathrm{t}),(q_{13},3,1,NP,\mathrm{t}),\\
&(q_9,2,2,VP,\mathrm{t}),(q_4,1,1,VP,\mathrm{t})\}\{(q_8,4,2,and,\mathrm{t}),\\
&(q_{11},4,4,and,\mathrm{t})\},n\ v\ n\$) \qquad \text{[CASE 2]}
\end{aligned}
$$

and を処理した時, *UGLR*構文解析の状態リストΠをグラフで表したものを図 5.5に示す.

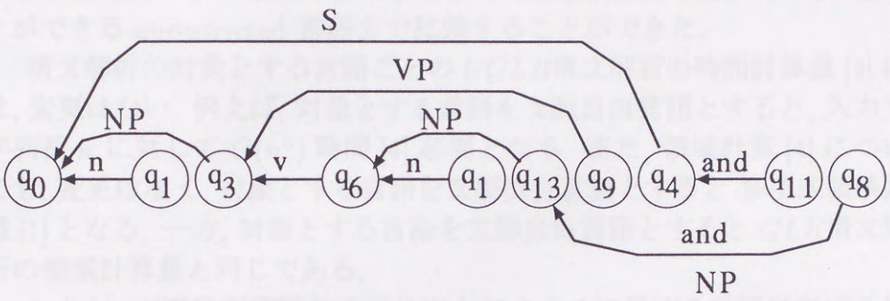


図 5.5: 状態リストΠの変化.

Fig. 5.5: An example of transitions of state list Π.

*UGLR*構文解析の状態リストΠをグラフで表したものを図 5.6に示す. 図 5.6のグラフは, Πの要素の中の状態 q_iを節点, 要素のポインタを有向辺, 文法記号を有向辺に付加されたラベルとしている.

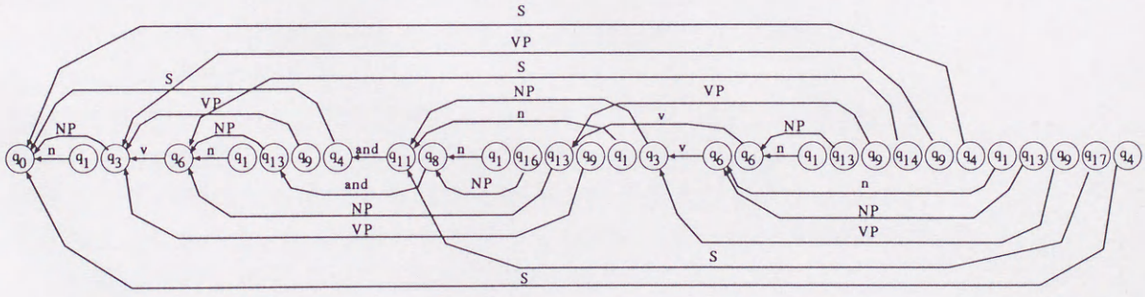


図 5.6: 入力列 "*n v n and n v n*" に対する状態リストΠの例.

Fig. 5.6: An example of list Π for an input sequence .

5.5 おわりに

$UGLR$構文解析はGLR構文解析を拡張することによって, 構文解析ができる言語のクラスを文脈自由言語から unrestricted 文法で生成することができる unrestricted 言語まで拡張することができた.

構文解析の対象とする言語ごとの$UGLR$構文解析の時間計算量 [4] には, 変更はない. 例えば, 対象とする言語を文脈自由言語とすると, 入力文字列長 n に対して $O(n^3)$ 時間 [4] 必要となる. また, 領域計算 [4] についても, 変更はなく, 対象とする言語を文脈依存言語とすると 多項式領域困難 [1] となる. 一方, 対象とする言語を文脈自由言語とするとGLR構文解析の領域計算量と同じである.

しかし, 文脈依存言語のクラス以上のクラスに属する言語に対する構文解析の実際の計算機へのインプリメントを考えると, 時間計算量のほかにも領域計算量への考慮が必要となる.

第 6 章

結論

6.1 本論文のまとめ

本論文では, 自然言語処理やコンパイラなどで使われる構文解析の高速化について述べた. 特に, 文脈自由言語より小さいクラスの言語に対しては, より少ないプロセッサを用いる並列アルゴリズムを提案し, 文脈自由言語より大きいクラスの言語に対しては, まず文法と言語を定義し, それに対する逐次の構文解析法を提案した.

第 2 章では, 単純順位文法の生成する言語に対して, 予測される構文解析木の節点に当たる非終端記号と入力に当たる終端記号との優先順位関係を表す構文解析テーブルを作成することによって, $O(n^2)$ 個のプロセッサを用いて $O(\log^2 n)$ 時間で処理する並列構文解析のアルゴリズムを提案し, 妥当性を明らかにした.

第 3 章では, LR 文法の生成する言語に対して, 入力から得られる予測される LR 状態遷移図の状態と予測される構文解析木の節点に当たる非終端記号との関係から得られる動作の種類を表す構文解析テーブルを作成することによって, $O(n^3)$ 個のプロセッサを用いて $O(\log n)$ 時間で処理する並列構文解析のアルゴリズムを提案し, 妥当性を明らかにした.

第 4 章では, 句構造文法の一種である unrestricted 文法の部分クラスである unrestricted $SLR(1)$ 文法 や unrestricted $LALR(1)$ 文法を, 先読み文字列を終端記号から非終端記号に変更することによって拡張した unresticted $LR(k)$ 文法を提案し, それに対して構文解析の処理がバックトラックできるようにする unrestricted $LR(k)$ 構文解析法を提案した.

また, 第 5 章では, 構文解析を行なう対象とする言語のクラスを句構造文法 の一つである unrestricted 文法 が生成する言語のクラスとし, そ

の言語を構文解析する方法として GLR構文解析を拡張した $UGLR$構文解析を新たに提案した.

6.2 今後の課題

本論文は, まず理論的なモデルである P-RAM 上における構文解析の高速化について述べた. 本論文で提案した P-RAM モデル上での並列構文解析は, 単純順位文法に対しても LR文法に対しても共通して構文解析表を作成する方法を用いている. そのため, 誤挿入, 置換, 及び, 削除がある入力文字列を訂正するなどの非文解析などへの応用や確率文脈自由文法モデルの並列解析に対して適用することは難しいと考えられる. 特に, アルゴリズムとして応用面を考えると構文解析表の柔軟な構築が必要と考えられる. 一方, 文脈自由言語より広い言語のクラスを構文解析する問題では, 定義した unrestricted $LR(k)$ 文法が生成できる言語のクラスを明確化することが必要であり, LR文法の場合と同様に unrestricted $LR(k)$ 文法から unrestricted $LR(1)$ 文法への変換可能性を調べる必要がある.

また, 自然言語やコンパイラの処理は, 構文解析だけではなく, 意味解析も重要な処理である. 意味解析モデルの構築は, その表現能力以外に, 意味解析の処理速度にも大きな影響を与える. 言語理論における構文論である GB 理論, GPSG, 及び, LFG などの処理方式は, 一般的な自然言語の理解を目的としているため, 処理時間を考慮していない. 一方, コンパイラや一部の自然言語処理の意味解析において属性文法モデルが用いられることがある. このモデルは, 構文解析木の文法記号に意味 (属性文法では, これを属性と呼ぶ) を与え, 意味は代入式の形式で各生成規則に付随されている. そのため, 構文解析木の変形を必要としないで, 意味が決定できるため, 理論的に解析がしやすい. 既に, コンパイラの構文解析と同時に意味の評価を進める属性文法のクラスが研究されてきているが, 一般化 LR 構文解析と同時に意味の評価を進める属性文法のクラスや文脈自由文法が生成する言語より広いクラスの言語を生成する文法に付随する形の属性文法や属性の種類を考案することは, 自然言語処理のアルゴリズムの定式化や処理の向上をもたらすと考えられる.

謝辞

本研究を進めるにあたって, 終始熱烈な御指導をいただいた豊橋技術科学大学 工学部 教授 増山繁 先生に深く感謝致します.

本博士論文の執筆にあり, 有益な御討論, 御助言を頂いた 豊橋技術科学大学工学部教授 磯田定宏 先生, 同大学工学部教授 中川聖一 先生, 同大学工学部助教授 河合和久 先生に対し, 深くお礼を申し上げます.

また, 本研究の討論に参加して頂いた増山研究室の皆さんや博士後期課程進学後も常に励ましたくれた平成 5 年度 豊橋技術科学大学 大学院知識情報工学研究科 修了生の皆さんにもお礼を申し上げたい.

最後に, 常に励まし, 勇気づけ, 研究生活を支えてくれた両親に心から感謝致します.

参考文献

[1] Kuroda, S.Y., (1967). *Classes of languages and linear-bounded automata.* Inform. Control, 7, 207-223.

[2] Earley, J. (1970). *An efficient context-free parsing algorithm.* Commun. ACM, 13, 2, 94-102.

[3] Loeckx, J. (1970). *The parsing for general phase-structure grammars.* Inform. Control, 16, 443-464.

[4] Hopcroft, J.E. and Ullman, J.D. (1979). *Introduction to Automata Theory, Languages and Computation.* Addison-Wesley.

[5] Vold'man, G.Sh.(1981). *A parsing algorithm for context-sensiteve grammars.* Program. Comput. Software, 7, 302-307.

[6] G.Sh. Vol'dman: "A parsing algorithm for context-sensitive grammars", Program. Comput. Softw., vol.7, pp.302-307, 1981.

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

[8] Tomita, M. (1984). *LR Parsier for Natural Languages.* Proc. of the International Conference on Computational Linguistics.

[9] Peter Sells: *Lectures on Contemporary Syntactic Theories, (An Introduction to Government-Binding Theory, Generalized Pharase Grammar, and Lexical-Functional Grammar)*, Center for the Study of Language and Information Leland Stanford Junior University(1985), 郡司隆男, 田窪行則, 石川彰 訳, 現代の文法理論 (GB 理論, GPSG, LFG 入門), 産業図書 (1988).

[10] 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).

[11] Tomita, M. (1987). *An Efficient Augmented-Context-Free Parsing Algorithm.* Computational Linguistics, 13,1-2, 31-46.

[12] Harris, L.A. (1987). *SLR(1) and LALR(1) parsing for unrestricted grammar.* Acta Inform., 24, 191-209.

[13] Rytter W.: *On the complexity of parallel parsing of general context-free languages*, Theoretical Computer Science, 47, pp.315-322 (1987).

[14] Rytter W.: *Parallel time log(n) recognition of unambiguous context-free languages*, Information and Computation, 73, pp.75-86 (1987).

[15] Pan V. Y.: " Complexity of Parallel Matrix Computations", Theoretical Computer Science, 54, pp.65-85 (1987).

[16] Coppersmith, D and S. Winograd: "Matrix multiplication via arithmethic progressings", Poc. 19th Ann. ACM Symp. on Theory of Computing pp.65-85 (1987).

[17] Harris L. A. : "SLR(1) and LALR(1) parsing for unrestricted grammar", Acta Inform., vol.24, pp.191-209, 1987.

[18] 野村 浩郷 (1988). 自然言語処理の基礎技術. 電子情報通信学会.

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

[20] 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).

[21] 田中 穂積 (1989), 自然言語解析の基礎, 産業図書.

[22] J. van Leeuwen: *Handbook of Theoretical Computer Science*, Volume A, Algorithms and Complexity, The MIT Press (1990).

[23] Grune D. and Jacobs C. J. H.: "Parsing Techniques: a practical guide", Ellis Horwood Limited 1990.

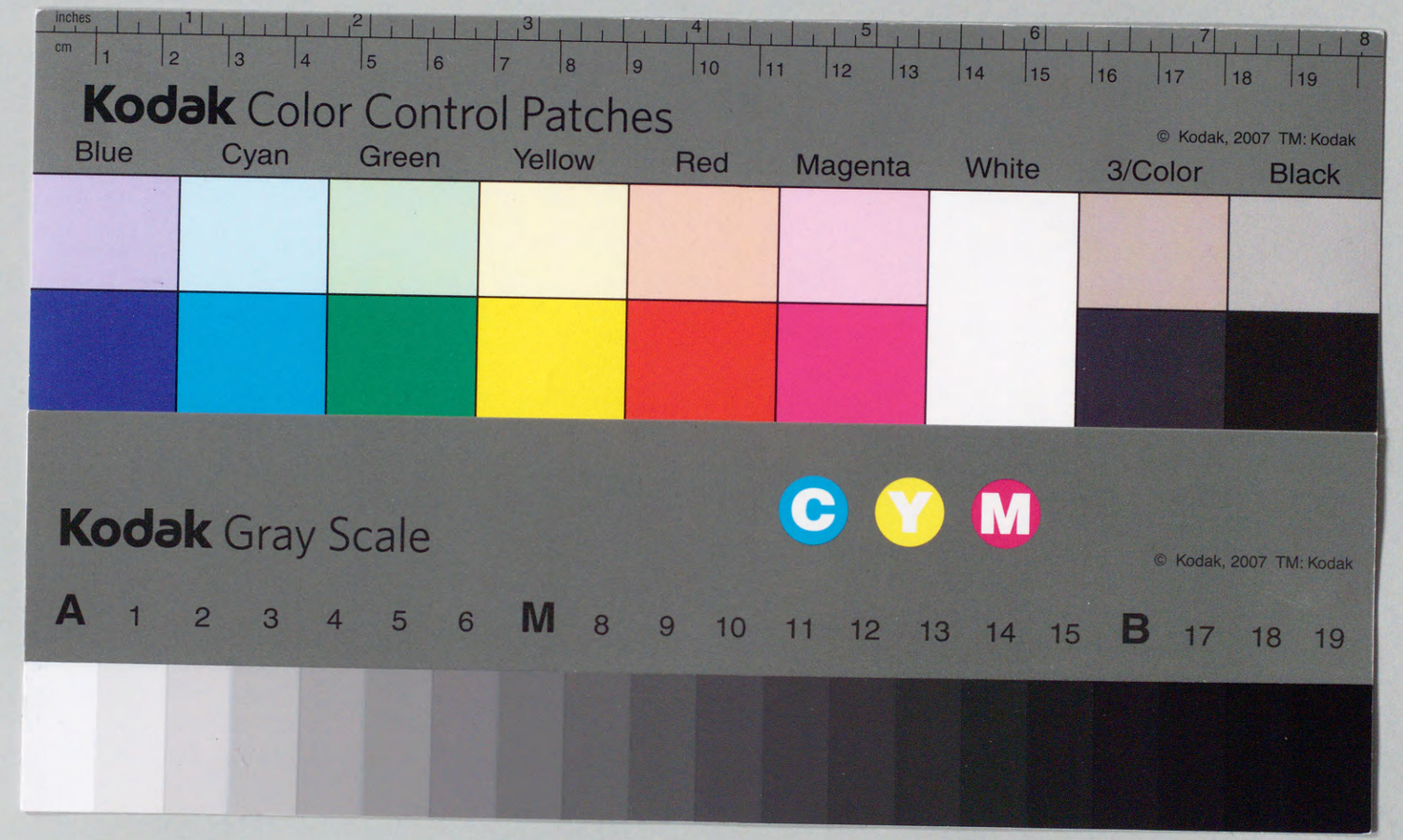
[24] Tomita, M. (1991). *Generalized LR Parsing.* Kluwer Academic Publishers.

[25] 富田悦次, 横森貴: “オートマトン・言語理論”, 森北出版 (1992).

[26] Monien B., Rytter W., Schäpers L.: “Fast recognition of deterministic cfl's with a smaller number of processors”, Theoretical Computer Science, 116, pp.421-429 (1993).

[27] 井上 謙蔵: コンパイラ, 丸善 (1994).

[28] 椎名 広光, 増山 繁, “単純順位文法に対する並列構文解析アルゴリズム”, 信学会 (DI), 採録決定.


inches
cm
Kodak Color Control Patches
© Kodak, 2007 TM: Kodak
Blue
Cyan
Green
Yellow
Red
Magenta
White
3/Color
Black
Kodak Gray Scale
C
Y
M
© Kodak, 2007 TM: Kodak
A 1 2 3 4 5 6 M 8 9 10 11 12 13 14 15 B 17 18 19