あいまい性が強い文脈自由文法の枝刈りに基づく効率的な構文解析

森本真一[†] 石畑 清[††] 疋田輝雄[††]

本論文では，強いあいまい性を持つ文脈自由文法に対しても効率的な構文解析アルゴリズムを示す．本アルゴリズムはデータ構造として，Tomita ら (1991)，Kipps (1991)，Nederhof (1993) と同様に，グラフ構造スタックを用いる．ここでのグラフ構造スタックは，項と入力列の組を頂点とし，各頂点から，解析木でいう親頂点に対応する頂点への辺（親ポインタ）を持つ有向グラフである．本アルゴリズムの特徴は，還元時に項の部分が同じ頂点への親ポインタを枝刈りすることである．枝刈りはすべての文脈自由文法に対して可能ではないが，あいまいさの強い文法の多くは枝刈り可能である．また枝刈りすることにより，すべての解析木を求めることはできなくなる．しかし同じ頂点からの親ポインタの指す頂点の項部分はすべて異なるので，各頂点からの親ポインタの個数は入力列の長さ n に依存しない定数で抑えられる．これにより，枝刈りが可能な文脈自由文法に対する構文解析の時間計算量は $O(n^2)$ となる．本論文ではまず本アルゴリズムの定義を述べ，正当性すなわち入力列が文法の語であるかを正しく判定できることを示し，さらに時間計算量の解析を行う．本アルゴリズムの適用例として，Kipps によって与えられているあいまいさの強い典型的な文法（Kipps の構文解析法による時間計算量は $O(n^3)$）に対して，本アルゴリズムでは $O(n^2)$ であることを示す．最後に，本アルゴリズムで扱える枝刈り可能な文脈自由文法の範囲について，いくつかの文法例に基づく定性的な考察を行い，これらが十分に広い範囲を占めることを示す．

An Efficient Parsing for Highly Ambiguous Context-free Grammars Based on Pruning

Shin-ichi Morimoto,[†] Kiyoshi Ishihata[††] and Teruo Hikita[††]

We introduce an efficient parsing algorithm even for highly ambiguous context-free grammars. It uses a graph-structured stack as those by Tomita (1991), Kipps (1991), and Nederhof (1993). Here, a graph-structured stack is a directed graph whose nodes are pairs of an item and an input string, and whose edges are pointers (parent pointers), each pointing to a parent node in the parse tree. Our new technique here is to prune the parent pointers pointing to the nodes having the same item part. By this pruning, we cannot obtain all parse trees, but we can reduce the size of the set of parent pointers to a constant, thus making the time complexity of this algorithm $O(n^2)$. In this paper we first define the algorithm and show its correctness, and then we analyze its time complexity. Then we show that by our parsing a class of typical highly ambiguous grammars given by Kipps (1991) that need $O(n^3)$ by his algorithm can be parsed in $O(n^2)$. Finally some study is made on the range of prunable grammars.

1. はじめに

あいまい性を持つ文脈自由文法の構文解析に対して，これまで提案されたアルゴリズムは，索表方式 (tabular method)[1),3)] とスタック方式[2),4),9)] に大別できる．索表方式は，$n \times n$ の行列を用いて入力列 $a_1 \cdots a_n$ の解析を行う．この行列の (i, j) 要素は，入力列の $a_{i+1} \cdots a_j$ に対応しうる項の集合を表す．

スタック方式は，解析スタック（parse stack）とよばれるスタックを用いて解析を行う．一般の文脈自由文法に対するスタック方式のアルゴリズムとしては，Tomita らのアルゴリズム[9)] と，それを改良した Kipps[2)]，Nederhof[4)] がある．これらのスタック方式では，あいまいな文法を解析するために，解析スタックの集合，または解析スタックの集合を 1 つの有向グラフとして表現したグラフ構造スタックを用いる．グラフ構造スタックとは，各解析スタックの要素を頂点とし各頂点からその 1 つ前の要素に対応する頂点へのポインタ（親ポインタ）を辺とする有向グラフである．例として，D をトップ要素とする 3 つのスタックとそ

† 株式会社 NEC 航空宇宙システム
NEC Aerospace Systems, Ltd.
†† 明治大学理工学部情報科学科
Department of Computer Science, Meiji University

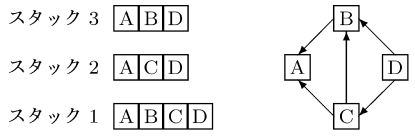

図 1 スタックの集合と対応するグラフ構造スタック
Fig. 1 Stack set and corresponding graph structured stack.

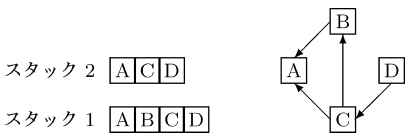

図 2 枝刈り後のグラフ構造スタック
Fig. 2 Graph-structured stack after pruning.

れらに対応するグラフ構造スタックを**図 1** に示す．

時間計算量は，索表方式では $O(n^3)$ であり，スタック方式では Tomita らのアルゴリズム[9] では $O(n^{p+1})$（p は右辺が最も長い構文規則の長さ）なので，索表方式のほうが優れている．ただし文献 2) では索表により，文献 4) ではメモ関数により，それぞれ構文解析の過程で同じ計算を重複して行わないようにすることで，時間計算量 $O(n^3)$ のアルゴリズムを示している．実際に適用する場合はスタック方式のほうが索表方式よりも効率的であるという報告がある[6]．

本論文のアルゴリズムはグラフ構造スタックを用いるスタック方式であり，文献 2), 4) をさらに改良したものである．本アルゴリズムでは，構文解析の過程でグラフ構造スタックの新しい頂点を生成する際に，可能な場合は新しい頂点の親ポインタの枝刈り（pruning）を行う．親ポインタの枝刈りを行った例を**図 2** に示す．図 2 は図 1 の頂点 D の親ポインタ（D を始点とする辺）のうち，頂点 B への辺に対応する親ポインタを枝刈りにより削除した場合のグラフ構造スタックと対応するスタックの集合を表したものである．親ポインタの枝刈りを行うことは，枝刈りされた親ポインタに対応するスタックを削除することを意味する．図 2 において枝刈りにより頂点 D から頂点 B への辺（親ポインタ）を削除することは，グラフ構造スタックに対応するスタックの集合から図 1 のスタック 3 に対応するスタックを削除することを意味する．

本アルゴリズムで用いるグラフ構造スタックの頂点は，LR 構文解析における LR(0) 項に対応する部分とこれまで読み込んだ（解析した）入力列を示す部分から構成される．グラフ構造スタックの各頂点の親ポインタが指す頂点の集合をその頂点の Parent 集合というとき，本アルゴリズムでは各頂点の Parent 集合に LR(0) 項の部分が同じである要素が複数含まれないように，親ポインタの枝刈りを行う．この枝刈りにより Parent 集合の大きさはつねに入力の長さに依存しない定数（LR(0) 項の個数）以下に抑えられる．ただし枝刈りにより，可能なすべての解析木に対応する構文解析を行えなくなる．

本アルゴリズムでは，枝刈りにより各頂点の Parent 集合の大きさを $O(1)$ とし，構文解析での多くの処理を $O(1)$ で行うことにより，時間計算量を $O(n^2)$ としている．

2 章では用語の定義を述べ，3 章で解析スタックの集合を用いたアルゴリズムを，4 章でグラフ構造スタックを用いたアルゴリズムを述べる．3 章と 4 章のアルゴリズムは文献 2) や 4) のアルゴリズムと基本的に同じであるが，従来の論文では形式化が十分でなく直観に頼る部分があったので，5 章で述べる枝刈りアルゴリズムの記述の準備として形式化を行う．5 章では本論文の主題である枝刈り式グラフ構造スタック法の定義と正当性を述べ，6 章でその時間計算量の解析を行う．7 章では枝刈り式グラフ構造スタック法の適用例として，構文解析の時間計算量が Tomita らのアルゴリズム[9] では $O(n^{m+1})$，それを改良したアルゴリズム[2] では $O(n^3)$ である文法 S_m（$m \geq 3$）に対して，枝刈り式グラフ構造スタック法では $O(n^2)$ で構文解析が行えることを示す．最後に 8 章で枝刈り式グラフ構造スタック法が適用できる文法と適用できない文法の条件の例を示す．

2. 術語の定義

以下では，一般の列 η に対して，

$\sharp\eta$：列 η の要素数，

η_j：列 η の j 番目の要素，

$\eta_\sharp$：列 η の $\sharp\eta$ 番目（末尾）の要素

を表す．また列 η，η'，要素 a に対して，

$\eta \cdot \eta'$：η に η' を連結した列，

$\eta \cdot a$：η に a を連結した列

を表す．ε は空列を表す．

定義 1（文脈自由文法） 文脈自由文法 G は，4 つ組 (V_N, V_T, S, P) で定義される．ただし V_N は非終端記号の集合，V_T は終端記号の集合，S は開始記号，P は構文規則の集合である．$V = V_N \cup V_T$ とする．また $r \in P$ に対して，r の右辺の構文記号の個数を $\sharp r$，r の j 番目の構文記号を r_j と表す．これにより r は $r_0 : r_1\ r_2 \cdots r_{\sharp r}$ と表せる．

本論文では，構文記号 S'，$\dashv$ と構文規則 $S' : S\ \dashv$ を追加した文脈自由文法を考える．また本論文で扱う

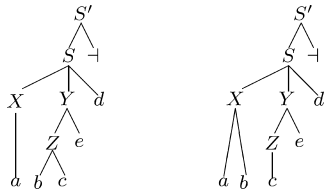

図 3 $G1$ の解析木
Fig. 3 Parse trees for $G1$.

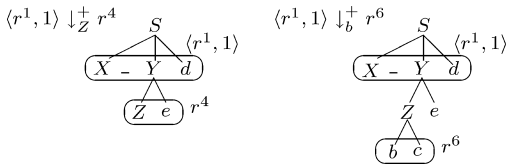

図 4 $\downarrow_w^+$ の例
Fig. 4 Examples of $\downarrow_w^+$.

文脈自由文法は,
(1) ε 規則（$\sharp r = 0$ となる構文規則）を含まない,
(2) 単一規則（右辺が非終端記号 1 個だけである構文規則）だけからなるループは存在しない,
とする．(2) より本論文で扱う文脈自由文法は，たとえば $A : B$，$B : C$，$C : A$ という構文規則の集合は含まない．

例 1 $G1$ を次の構文規則を持つ文脈自由文法とする．

$(r^0)\ S' : S\ \dashv$
$(r^1)\ S : X\ Y\ d \quad (r^2)\ X : a \quad (r^3)\ X : a\ b$
$(r^4)\ Y : Z\ e \quad (r^5)\ Z : c \quad (r^6)\ Z : b\ c$

語 $abced$ に対する $G1$ のすべての解析木を**図 3** に示す．

本論文では $G1$ を文脈自由文法の例として用いる．より複雑な文脈自由文法に適用した例は 7 章に示す．

定義 2（項） $r \in P$ と $0 \le n \le \sharp r$ に対して，$\langle r, n\rangle$ を項（item）といい，項全体の集合を I_G で表す．r が $r_0 : r_1 r_2 \ldots r_{\sharp r}$ と表されるとき，$\langle r, n\rangle$ を

$$r_0 : r_1 r_2 \ldots r_n \_ r_{n+1} \ldots r_{\sharp r}$$

と表すことがある．$S' : \_S\ \dashv$ という項を i_0 で表す．

項は LR 構文解析における LR(0) 項と同じである．

定義 3（拡張項） $t \in I_G$，$\alpha \in {V_T}^*$ に対して，$\langle t, \alpha\rangle$ を拡張項（extended item）といい，拡張項全体の集合を J_G で表す．$x \in J_G$ に対して，x の I_G の部分を body 部といい $\mathrm{body}(x)$ で表し，${V_T}^*$ の部分を input 部といい $\mathrm{input}(x)$ で表す．$\langle i_0, \varepsilon\rangle$ という拡張項を e_0 で表す．

本論文では，α は構文解析においてそれまでに読み込んだ（解析した）入力文字列である．拡張項 $\langle t, \alpha\rangle$ は，読み込み済み入力列が α である時点での状態が t であることを表す．たとえば，$e_0 (= \langle i_0, \varepsilon\rangle)$ は入力を読み込まない時点での状態が i_0 であることを表し，$\langle Z : b\_c, ab\rangle$ は入力 ab を読み込んだ時点での状態が $Z : b\_c$ であることを表す．

Kipps [2] では，$\langle i, s, l\rangle$（i は読み込んだ文字数，s は解析状態，l は Parent 集合）という拡張項と同様の集合を用いてアルゴリズムを定義している．

定義 4（$\downarrow_w^+$） $\langle r, n\rangle \in I_G$ $(0 \le n < \sharp r)$，$r' \in P$ に対して，$r_{n+1} = {r'}_0$ が成り立つとき，$\langle r, n\rangle \downarrow r'$ と定義する．さらに $w \in V$ に対して
(1) $\langle r, n\rangle \downarrow r^{m_1}$,
(2) $\langle r^{m_j}, 0\rangle \downarrow r^{m_{j+1}}, (r^{m_j})_1 \ne w$ $(1 \le j < k)$,
(3) $(r^{m_k})_1 = w$
を満たす $r^{m_1}, \ldots, r^{m_k} = r' \in P$（$k \ge 1$）が存在するとき，$\langle r, n\rangle \downarrow_w^+ r'$ と定義する．

$\langle r, n\rangle \downarrow r'$ は，r の右辺の $n+1$ 番目の構文記号から r' が展開できることを表す．$\langle r, n\rangle \downarrow_w^+ r'$ は，r の右辺の $n+1$ 番目の構文記号から最左導出を繰り返すことにより r' が展開でき，r' の右辺の先頭の構文記号が w であることを表す．

例 2 例 1 の $G1$ において，$r^1 = S : XYd$，$r^4 = Y : Ze$，$r^6 = Z : bc$ だから $\langle r^1, 1\rangle \downarrow r^4$，$\langle r^1, 1\rangle \downarrow_Z^+ r^4$，$\langle r^1, 1\rangle \downarrow_b^+ r^6$ が成り立つ．$\langle r^1, 1\rangle$，r^4，r^6 の関係を**図 4** に示す．

Nederhof [4] では，$B \angle A$ という $\langle r, n\rangle \downarrow r'$ と類似の概念を用いてアルゴリズムを記述している．$B \angle A$ は，$A : B\alpha$ という構文規則が存在することを示す．

定義 5（シフト可能集合） $a \in V_T$ に対して，

$$\mathrm{S0eitem}(a) = \{\langle\langle r, n\rangle, \alpha\rangle \mid r_{n+1} = a, \text{ ただし } r \in P, 0 \le n < \sharp r, \alpha \in V_T^*\},$$
$$\mathrm{S1eitem}(a) = \{\langle\langle r, n\rangle, \alpha\rangle \mid \langle r, n\rangle \downarrow_a^+ r', \text{ ただし } r, r' \in P, 0 \le n < \sharp r, \alpha \in V_T^*\}$$

と定義する．

S0eitem(a)，S1eitem(a) は，いずれも次に a を読み込む可能性のある拡張項の集合である．S0eitem(a) の要素は a を読み込んだ後の状況に対応する拡張項の body 部の構文規則が読み込む前と変わらない拡張項であり，S1eitem(a) の要素は a を読み込んだ後の状況に対応する拡張項の body 部の構文規則が読み込む前と異なる拡張項である．

定義 6（シフト集合） $a \in V_T$，$x = \langle\langle r, n\rangle, \alpha\rangle \in \mathrm{S0eitem}(a)$ に対して，

$$\mathrm{Shift0}(x, a) = \{\langle\langle r, n+1\rangle, \alpha \cdot a\rangle\}$$

と定義する．$a \in V_T$，$x = \langle\langle r, n\rangle, \alpha\rangle \in \mathrm{S1eitem}(a)$ に対して，

$$\mathrm{Shift1}(x, a) = \{\langle\langle r', 1\rangle, \alpha \cdot a\rangle \mid \langle r, n\rangle \downarrow_a^+ r'\}$$

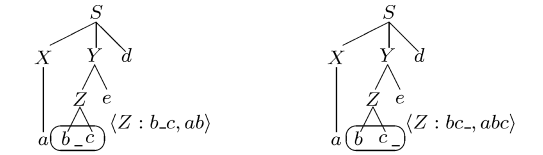

図 5 S0eitem(c) と Shift0(x, c) の例
Fig. 5 Example of S0eitem(c) and Shift0(x, c).

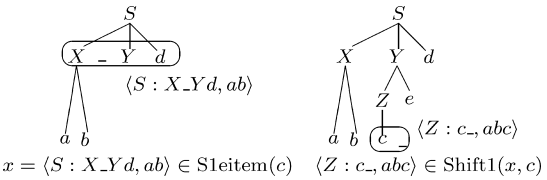

図 6 S1eitem(c) と Shift1(x, c) の例
Fig. 6 Example of S1eitem(c) and Shift1(x, c).

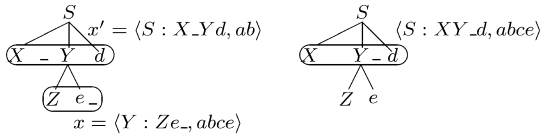

図 7 R0eitem(x) と Reduce0(x', x) の例
Fig. 7 Example of R0eitem(x) and Reduce0(x', x).

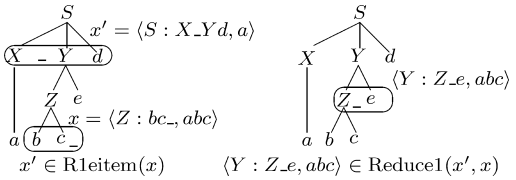

図 8 R1eitem(x) と Reduce1(x', x) の例
Fig. 8 Example of R1eitem(x) and Reduce1(x', x).

と定義する．

Shift0(x, a) の要素は S0eitem(a) の要素 x に対応する状況で a を読み込んだ状況に対応する拡張項である．Shift1(x, a) の要素は S1eitem(a) の要素 x に対応する状況で a を読み込んだ状況に対応する拡張項である．

例 3 例 1 の $G1$ において，$\langle Z : b\_c, ab\rangle \in$ S0eitem(c) に対して，

$$\text{Shift0}(\langle Z : b\_c, ab\rangle, c) = \{\langle Z : b\ c\_, abc\rangle\}.$$

これらの関係を**図 5** に示す．

$\langle S : X\_Y\ d, ab\rangle \in$ S1eitem(c) に対して，

$$\text{Shift1}(\langle S : X\_Y\ d, ab\rangle, c) = \{\langle Z : c\_, abc\rangle\}.$$

これらの関係を**図 6** に示す．

定義 7（Redex）

Redex $= \{x \in J_G \mid \text{body}(x) = \langle r, \sharp r\rangle$，ただし $r \in P\}$

と定義する．

定義 8（還元進行可能集合） $x = \langle\langle r, \sharp r\rangle, \alpha\rangle \in$ Redex に対して，

$$\text{R0eitem}(x) = \{\langle\langle r', n'\rangle, \alpha'\rangle \mid r'_{n'+1} = r_0\},$$
$$\text{R1eitem}(x) = \{\langle\langle r', n'\rangle, \alpha'\rangle \mid \langle r', n'\rangle{\downarrow^+}_{r_0}\ r'',$$
$$\text{ただし } r'' \in P\}$$

と定義する．

Redex は body 部が還元可能な拡張項の集合である．body 部が還元可能な拡張項 x に対して R0eitem(x), R1eitem(x) は，いずれも x の Parent 集合の要素となる可能性のある拡張項の集合である．R0eitem(x) の要素は x の還元後の状況に対応する拡張項の body 部の構文規則が還元前の Parent 集合の要素と変わらない拡張項であり，R1eitem(x) の要素は x の還元後の状況に対応する拡張項の body 部の構文規則が還元前の Parent 集合の要素と異なる拡張項である．

定義 9（還元集合） $x = \langle\langle r, \sharp r\rangle, \alpha\rangle \in$ Redex, $x' = \langle\langle r', n'\rangle, \alpha'\rangle \in$ R0eitem(x) に対して，

$$\text{Reduce0}(x', x) = \{\langle\langle r', n' + 1\rangle, \alpha\rangle\}$$

と定義する．$x = \langle\langle r, \sharp r\rangle, \alpha\rangle \in$ Redex, $x' = \langle\langle r', n'\rangle, \alpha'\rangle \in$ R1eitem(x) に対して，

$$\text{Reduce1}(x', x) = \{\langle\langle r'', 1\rangle, \alpha\rangle \mid \langle r', n'\rangle{\downarrow^+}_{r_0}\ r''\}$$

と定義する．

Reduce0(x', x) の要素は x の Parent 集合の要素 x' が R0eitem(x) に含まれる場合に x の還元後の状況に対応する拡張項である．Reduce1(x', x) の要素は x の Parent 集合の要素 x' が R1eitem(x) に含まれる場合に x の還元後の状況に対応する拡張項である．

例 4 例 1 の $G1$ において，$x = \langle Y : Z\ e\_, abce\rangle$, $x' = \langle S : X\_Y\ d, ab\rangle$ とすると，$x \in$ Redex, $x' \in$ R0eitem(x),

$$\text{Reduce0}(x', x) = \{\langle S : X\ Y\_d, abce\rangle\}$$

が成り立つ．これらの関係を**図 7** に示す．

また，$x = \langle Z : b\ c\_, abc\rangle$, $x' = \langle S : X\_Y\ d, a\rangle$ とすると，$x \in$ Redex, $x' \in$ R1eitem(x),

$$\text{Reduce1}(x', x) = \{\langle Y : Z\_e, abc\rangle\}$$

が成り立つ．これらの関係を**図 8** に示す．

3. 解析スタック法

本章では一般の文脈自由文法に対して，解析スタックの集合を用いた構文解析法を示す．解析スタックの集合を用いた既存の構文解析法としては，Tomita ら[9]，Kipps[2]，Nederhof[4] がある．文献 9)，2) の解析法ではスタックの要素は LR 集合であり，スタックの構成は通常の LR 解析法と同じである．文献 4) の解析法ではスタックの要素は構文記号または LR(0) 項であ

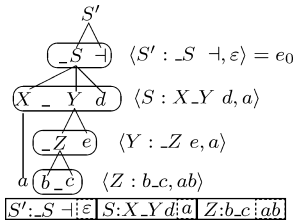

図 9 $G1$ の解析木と対応する解析スタック
Fig. 9 Parse tree and corresponding parse stack.

り，スタックの構成は解析木でルートから現在読み込んだ構文記号までのパス上の解析木の要素に対応する．本論文の構文解析法で用いる解析スタックは文献 4) の解析法の解析スタックと本質的に同じであるが，以下の 2 点が異なっている．

- 単一規則だけからなるループや ε 規則を含む文法は対象としない．
- スタックの要素は拡張項である．

この構文解析方法は与えられた語に対するすべての解析木を生成できる．

例 5 例 1 の $G1$ において入力列が ab の場合の解析木の 1 つ（図 3 の左側の解析木）とそれに対応する本方式の解析スタックを**図 9** に示す．解析木の S'（ルート）から読み込んだ構文記号 b までのパス上のノードに対応する拡張項も示す．本方式の解析スタックはこれらの拡張項を並べたものである．ただし body 部が non-kernel 項である拡張項は e_0 以外はスタックに含めない．図 9 では $\langle Y : \_Z\ e, a\rangle$ は，body 部が non-kernel 項なのでスタックに含めない．

アルゴリズム 1（解析スタック法） 入力列 $\alpha \in {V_T}^*$ と入力記号 $a \in V_T$ に対して，$\alpha \cdot a$ の構文解析によって得られる拡張項の列（解析スタックという）の集合 $L^{\alpha\cdot a} \subseteq {J_G}^*$ を α に対する解析スタックの集合 $L^\alpha \subseteq {J_G}^*$ から次のように帰納的に定義する．

(1) $L^\varepsilon = \{e_0\}$.

(2) $\mu{\cdot}x \in L^\alpha$ が $x \in \mathrm{S0eitem}(a)$, $y \in \mathrm{Shift0}(x, a)$ を満たすとき $\mu \cdot y \in L^{\alpha\cdot a}$.

(3) $\mu{\cdot}x \in L^\alpha$ が $x \in \mathrm{S1eitem}(a)$, $y \in \mathrm{Shift1}(x, a)$ を満たすとき，$\mu \cdot x \cdot y \in L^{\alpha\cdot a}$．ここで

$$\mathrm{Reduce}^{(0)}(L^{\alpha\cdot a}) = \bigcup_{\mu\cdot x\in L^\alpha}\Big(\bigcup_{x\in\mathrm{S0eitem}(a),y\in\mathrm{Shift0}(x,a)}\{\mu\cdot y\} \cup \bigcup_{x\in\mathrm{S1eitem}(a),y\in\mathrm{Shift1}(x,a)}\{\mu\cdot x\cdot y\}\Big)$$

と定義する（$\mathrm{Reduce}^{(0)}(L^{\alpha\cdot a})$ は，場合 (2) と (3) で生成される解析スタックの集合である）．

(4) $\mu \cdot x' \cdot x \in \mathrm{Reduce}^{(n)}(L^{\alpha\cdot a})$ $(n \geq 0)$ が $x \in \mathrm{Redex}$, $x' \in \mathrm{R0eitem}(x)$, $y \in \mathrm{Reduce0}(x', x)$ を満たすとき $\mu \cdot y \in L^{\alpha\cdot a}$.

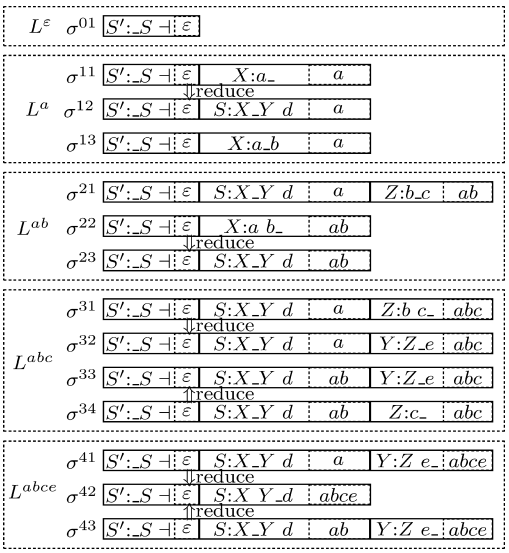

図 10 $G1$ における L^α の例
Fig. 10 Example of L^α in $G1$.

(5) $\mu \cdot x' \cdot x \in \mathrm{Reduce}^{(n)}(L^{\alpha\cdot a})$ $(n \geq 0)$ が $x \in \mathrm{Redex}$, $x' \in \mathrm{R1eitem}(x)$, $y \in \mathrm{Reduce1}(x', x)$ を満たすとき $\mu \cdot x' \cdot y \in L^{\alpha\cdot a}$．ここで

$$\mathrm{Reduce}^{(n+1)}(L^{\alpha\cdot a}) = \bigcup_{\mu\cdot x'\cdot x\in\mathrm{Reduce}^{(n)}(L^{\alpha\cdot a})}\Big(\bigcup_{x\in\mathrm{Redex},x'\in\mathrm{R0eitem}(x),y\in\mathrm{Reduce0}(x',x)}\{\mu\cdot y\} \cup \bigcup_{x\in\mathrm{Redex},x'\in\mathrm{R1eitem}(x),y\in\mathrm{Reduce1}(x',x)}\{\mu\cdot x'\cdot y\}\Big)$$

と定義する．

$L^{\alpha\cdot a}$ は，場合 (2)–(5) で定義される最小の集合である．

$L^{\alpha\cdot a}$ の要素は，L^α の要素から shift によって生成される要素（場合 (2)，(3) に対応）と $L^{\alpha\cdot a}$ の要素から reduce によって生成される要素（場合 (4)，(5) に対応）に分けられる．reduce によって生成される要素はすでに reduce によって生成された要素からさらに reduce によって生成されるものも存在する．そこでアルゴリズム 1 では，shift によって生成される要素から reduce を n 回行って生成される $L^{\alpha\cdot a}$ の要素の集合を $\mathrm{Reduce}^{(n)}(L^{\alpha\cdot a})$ とし，n に関する帰納法により reduce によって生成される要素を定義している．解析スタックから解析木を構成することは容易である．$\alpha \cdot a$ が構文的に正しくない入力である場合，$L^{\alpha\cdot a} = \emptyset$ である．

例 6 例 1 の文法 $G1$ において入力列が $abced$ の場合の解析の進行を**図 10** に示す．L^α の各要素 $(\sigma^{01}, ..., \sigma^{43})$ は解析スタックである．たとえば σ^{21} は，例 5 で示されたスタックであり図 3 の左側の解析

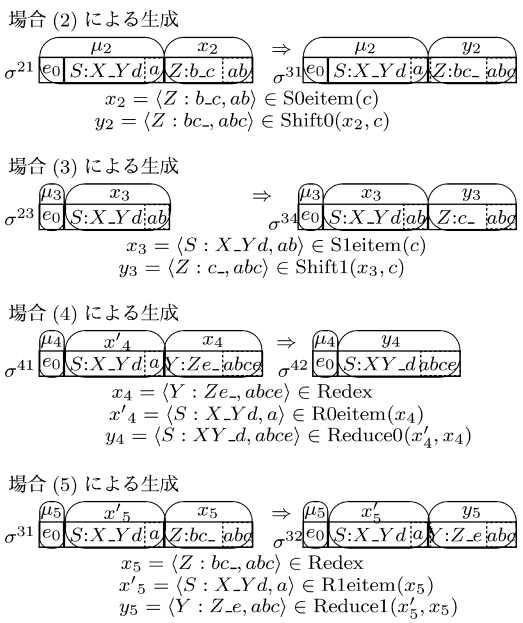

図 11 場合 (2)–場合 (5) による解析スタックの生成例
Fig. 11 Example of parse stack generation.

木に対応する．同様に σ^{23} は，図 3 の右側の解析木に対応する．図 10 の ⇓reduce や ⇑reduce は，アルゴリズム 1 の場合 (4) と (5) による解析スタックの生成を示す．

$\mu_1 = \varepsilon$，$x_1 = e_0$，$y_1 = \langle X : a\_, a\rangle$ とすると，$\mu_1 \cdot x_1 \in L^{\varepsilon}$，$x_1 \in \mathrm{S1eitem}(a)$，$y_1 \in \mathrm{Shift1}(x_1, a)$ を満たすので，アルゴリズム 1 の場合 (3) より，$\mu_1 \cdot x_1 \cdot y_1 \in L^{a}$．これは，図 10 の L^{ε} の解析スタック（σ^{01}）から L^{a} の上の解析スタック（σ^{11}）を生成する場合に対応する．

$\mu_2 = e_0 \cdot \langle S : X\_Y\ d, a\rangle$，$x_2 = \langle Z : b\_c, ab\rangle$，$y_2 = \langle Z : bc\_, abc\rangle$ とすると，$\mu_2 \cdot x_2 \in L^{ab}$，$x_2 \in \mathrm{S0eitem}(c)$，$y_2 \in \mathrm{Shift0}(x_2, c)$ を満たすので，アルゴリズム 1 の場合 (2) より，$\mu_2 \cdot y_2 \in L^{abc}$．これは，図 10 の L^{ab} の最も上の解析スタック（σ^{21}）から L^{abc} の最も上の解析スタック（σ^{31}）を生成する場合に対応する．

$\mu_3 = e_0$，$x_3 = \langle S : X\_Y\ d, ab\rangle$，$y_3 = \langle Z : c\_, abc\rangle$ とすると，$\mu_3 \cdot x_3 \in L^{ab}$，$x_3 \in \mathrm{S1eitem}(c)$，$y_3 \in \mathrm{Shift1}(x_3, c)$ を満たすので，アルゴリズム 1 の場合 (3) より，$\mu_3 \cdot x_3 \cdot y_3 \in L^{abc}$．これは，図 10 の L^{ab} の最も下の解析スタック（σ^{23}）から L^{abc} の最も下の解析スタック（σ^{34}）を生成する場合に対応する．

$\mu_4 = e_0$，$x_4 = \langle Y : Z\ e\_, abce\rangle$，$x'_4 = \langle S : X\_Y\ \ d, a\rangle$，$y_4 = \langle S : X\ \ Y\_d, abce\rangle$ とすると，$\mu_4 \cdot x'_4 \cdot x_4 \in \mathrm{Reduce}^{(0)}(L^{abce})$，$x_4 \in \mathrm{Redex}$，$x'_4 \in \mathrm{R0eitem}(x_4)$，$y_4 \in \mathrm{Reduce0}(x'_4, x_4)$ を満たすので，アルゴリズム 1 の場合 (4) より，$\mu_4 \cdot y_4 \in L^{abce}$．これは，図 10 の L^{abce} の最も上の解析スタック（σ^{41}）から L^{abce} の中央の解析スタック（σ^{42}）を生成する場合に対応する．

$\mu_5 = e_0$，$x_5 = \langle Z : b\ \ c\_, abc\rangle$，$x'_5 = \langle S : X\_Y\ d, a\rangle$，$y_5 = \langle Y : Z\_e, abc\rangle$ とすると，$\mu_5 \cdot x'_5 \cdot x_5 \in \mathrm{Reduce}^{(0)}(L^{abc})$，$x_5 \in \mathrm{Redex}$，$x'_5 \in \mathrm{R1eitem}(x_5)$，$y_5 \in \mathrm{Reduce1}(x'_5, x_5)$ を満たすので，アルゴリズム 1 の場合 (5) より，$\mu_5 \cdot x'_5 \cdot y_5 \in L^{abc}$．これは，図 10 の L^{abc} の最も上の解析スタック（σ^{31}）から L^{abc} の上から 2 番目の解析スタック（σ^{32}）を生成する場合に対応する．

これらの解析スタックの生成状況を**図 11** に示す．図 11 では，$\langle S' : \_S\ \dashv, \varepsilon\rangle$ を e_0 と略す．

4. グラフ構造スタック法

本章ではグラフ構造スタック[2),4),9)] に基づく構文解析アルゴリズムを示す．これは 3 章の解析スタックの集合を有向グラフで表現したグラフ構造スタックを対象としているので，3 章の解析スタック法が文献 4) の解析法と本質的に同じなのと同様に，本章のアルゴリズムも文献 4) の解析法と本質的に同じである．本章では，グラフ構造スタックの要素 x の Parent 集合（親ポインタの集合）を $\mathrm{Parent}(x)$ と表す．

アルゴリズム 2（グラフ構造スタック法） 入力列 $\alpha \in {V_T}^*$ と入力記号 $a \in V_T$ に対して，$\alpha \cdot a$ の構文解析によって得られる拡張項の集合 $E_{\alpha\cdot a} \subseteq J_G$ と $E_{\alpha\cdot a}$ の要素の Parent 集合を α に対する拡張項の集合 $E_\alpha \subseteq J_G$ から次のように帰納的に定義する．

(1) $E_\varepsilon = \{e_0\}$，$\mathrm{Parent}(e_0) = \emptyset$.

(2) $x \in \mathrm{S0eitem}(a)$ を満たす $x \in E_\alpha$ に対して，$\mathrm{Shift0}(x, a) \subseteq E_{\alpha\cdot a}$.
$y \in \mathrm{Shift0}(x, a)$ に対して $\mathrm{Parent}(y)$ を
$\mathrm{Parent}(y) = \{w \mid w \in \mathrm{Parent}(z), z \in E_\alpha \cap \mathrm{S0eitem}(a), y \in \mathrm{Shift0}(z, a)\}$
と定義する．

(3) $x \in \mathrm{S1eitem}(a)$ を満たす $x \in E_\alpha$ に対して，$\mathrm{Shift1}(x, a) \subseteq E_{\alpha\cdot a}$.
$y \in \mathrm{Shift1}(x, a)$ に対して $\mathrm{Parent}(y)$ を
$\mathrm{Parent}(y) = \{z \mid z \in E_\alpha \cap \mathrm{S1eitem}(a), y \in \mathrm{Shift1}(z, a)\}$
と定義する．ここで

$$\mathrm{Reduce}_{(0)}(E_{\alpha\cdot a}) = \bigcup_{x \in E_\alpha \cap \mathrm{S0eitem}(a)} \mathrm{Shift0}(x, a)$$

$$\cup \bigcup_{x\in E_\alpha\cap \mathrm{S1eitem}(a)} \mathrm{Shift1}(x,a)$$

と定義する（$\mathrm{Reduce}_{(0)}(E_{\alpha\cdot a})$ は，場合 (2) と (3) で生成される拡張項の集合である）．

(4) $x \in \mathrm{Redex}$ を満たす $x \in \mathrm{Reduce}_{(n)}(E_{\alpha\cdot a})$ に対して，$x' \in \mathrm{R0eitem}(x) \cap \mathrm{Parent}(x)$ とすると，
$\mathrm{Reduce0}(x', x) \subseteq E_{\alpha\cdot a}$．
$y \in \mathrm{Reduce0}(x', x)$ に対して $\mathrm{Parent}(y)$ を
$\mathrm{Parent}(y) = \{w \mid w \in \mathrm{Parent}(z'),$
$z' \in \mathrm{Parent}(z) \cap \mathrm{R0eitem}(z),$
$z \in \mathrm{Reduce}_{(n)}(E_{\alpha\cdot a}) \cap \mathrm{Redex},\ y \in \mathrm{Reduce0}(z', z)\}$
と定義する．

(5) $x \in \mathrm{Redex}$ を満たす $x \in \mathrm{Reduce}_{(n)}(E_{\alpha\cdot a})$ に対して，$x' \in \mathrm{R1eitem}(x) \cap \mathrm{Parent}(x)$ とすると，$\mathrm{Reduce1}(x', x) \subseteq E_{\alpha\cdot a}$．
$y \in \mathrm{Reduce1}(x', x)$ に対して $\mathrm{Parent}(y)$ を
$\mathrm{Parent}(y) = \mathrm{P}'(y)$
ただし，$\mathrm{P}'(y) =$
$\{z' \mid z' \in \mathrm{Parent}(z) \cap \mathrm{R1eitem}(z), z \in \mathrm{Reduce}_{(n)}(E_{\alpha\cdot a}) \cap \mathrm{Redex}, y \in \mathrm{Reduce1}(z', z)\}$
と定義する．ここで

$$\begin{aligned}&\mathrm{Reduce}_{(n+1)}(E_{\alpha\cdot a})\\ =&\bigcup_{x\in \mathrm{Reduce}_{(n)}(E_{\alpha\cdot a})\cap \mathrm{Redex}}\\ &\Big(\bigcup_{x'\in \mathrm{Parent}(x)\cap \mathrm{R0eitem}(x)} \mathrm{Reduce0}(x', x)\\ &\cup \bigcup_{x'\in \mathrm{Parent}(x)\cap \mathrm{R1eitem}(x)} \mathrm{Reduce1}(x', x)\Big)\end{aligned}$$

と定義する．

$E_{\alpha\cdot a}$ は，場合 (2)–(5) で定義される最小の集合である．

$\mathrm{Reduce}_{(n)}(E_{\alpha\cdot a})$ は，アルゴリズム 1 の $\mathrm{Reduce}^{(n)}(L^{\alpha\cdot a})$ と同様に shift によって生成される要素から reduce を n 回行って生成される $E_{\alpha\cdot a}$ の要素の集合であり，アルゴリズム 2 では $\mathrm{Reduce}_{(n)}(E_{\alpha\cdot a})$ を用いて reduce によって生成される要素をアルゴリズム 1 と同様に n に関する帰納法により定義している．

$y \in \mathrm{Shift0}(x, a)$ の場合は，x を $\langle\langle r', n'\rangle, \alpha'\rangle$，$y$ を $\langle\langle r, n\rangle, \alpha\rangle$ と表現すると，$x \in E_{\alpha'}$（$\alpha' \neq \varepsilon$）ならば $n' \geq 1$ が成り立つので，$r_n \in V_T$，$n \geq 2$ が成り立つ．同様に定義 6 および定義 9 より
$y \in \mathrm{Shift1}(x, a)$ の場合は，$r_n \in V_T, n = 1$,
$y \in \mathrm{Reduce0}(x', x)$ の場合は，$r_n \in V_N, n \geq 2$,
$y \in \mathrm{Reduce1}(x', x)$ の場合は，$r_n \in V_N, n = 1$

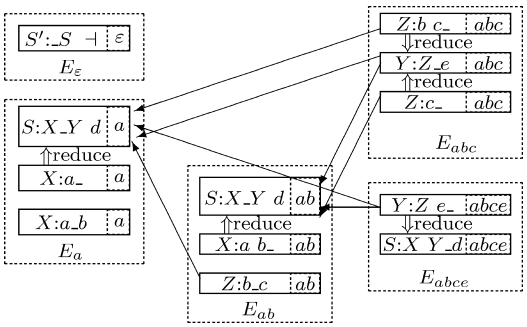

図 12 $G1$ における E_α の例
Fig. 12 Example of E_α in G1.

がそれぞれ成り立つから，$\mathrm{Shift0}(x, a)$，$\mathrm{Shift1}(x, a)$，$\mathrm{Reduce0}(x', x)$，$\mathrm{Reduce1}(x', x)$ は互いに排他的な集合である．よって $\mathrm{Parent}(y)$ の定義がアルゴリズム 2 の (2)–(5) の 2 つ以上の項にまたがることはない．

定義 10（E） $E = \bigcup_{\alpha\in V_T{}^*} E_\alpha$ と定義する．

例 7 例 1 の $G1$ において入力列が $abce$ の場合の E_α を**図 12** に示す．図 12 では $\mathrm{Parent}(x) \neq \{e_0\}$ の場合のみ，x から $\mathrm{Parent}(x)$ の要素への矢印を表示する．たとえば $x = \langle Y : Z\_e,\ abc\rangle$ の場合は，$\mathrm{Parent}(x) = \{\langle S : X\_Yd,\ a\rangle, \langle S : X\_Yd,\ ab\rangle\}$ となるので，x から $\langle S : X\_Yd,\ a\rangle$ と $\langle S : X\_Yd,\ ab\rangle$ への矢印は表示するが，$x' = \langle S : X\_Yd,\ ab\rangle$ の場合は $\mathrm{Parent}(x') = \{e_0\}$ なので，x' からの矢印は表示しない．

$x_2 = \langle Z : b\_c, ab\rangle$，$y_2 = \langle Z : b\ c\_, abc\rangle$ とすると，$x_2 \in E_{ab} \cap \mathrm{S0eitem}(c)$，$y_2 \in \mathrm{Shift0}(x_2, c)$ を満たすので，アルゴリズム 2 の場合 (2) より $y_2 \in E_{abc}$ が成り立つ．これは図 12 の E_{ab} の最も下の拡張項（$\langle Z : b\_c, ab\rangle$）から E_{abc} の最も上の拡張項（$\langle Z : b\ c\_, abc\rangle$）を生成する場合に対応する．このとき場合 (2) の $\mathrm{Parent}(y)$ の定義より，

$\mathrm{Parent}(y_2) = \{w \mid w \in \mathrm{Parent}(z),$
$z \in E_{ab} \cap \mathrm{S0eitem}(c),\ y_2 \in \mathrm{Shift0}(z, c)\}$

であり，$E_{ab} \cap \mathrm{S0eitem}(c) = \{x_2\}$，$y_2 \in \mathrm{Shift0}(x_2, c)$ だから

$\mathrm{Parent}(y_2) = \{w \mid w \in \mathrm{Parent}(x_2)\}$
$= \mathrm{Parent}(x_2) = \{\langle S : X\_Y\ d, a\rangle\}.$

$x_3 = \langle S : X\_Y\ d, ab\rangle$，$y_3 = \langle Z : c\_, abc\rangle$ とすると，$x_3 \in E_{ab} \cap \mathrm{S1eitem}(c)$，$y_3 \in \mathrm{Shift1}(x_3, c)$ を満たすので，アルゴリズム 2 の場合 (3) より $y_3 \in E_{abc}$ が成り立つ．これは図 12 の E_{ab} の最も上の拡張項（$\langle S : X\_Y\ d, ab\rangle$）から E_{abc} の最も下の拡張項（$\langle Z : c\_, abc\rangle$）を生成する場合に対応する．

このとき場合 (3) の $\mathrm{Parent}(y)$ の定義より，

Parent(y_3)
$= \{z \mid z \in E_{ab} \cap \mathrm{S1eitem}(c),\ y_3 \in \mathrm{Shift1}(z, c)\}$
であり，$E_{ab} \cap \mathrm{S1eitem}(c) = \{x_3\}$，$y_3 \in \mathrm{Shift1}(x_3, c)$ だから $\mathrm{Parent}(y_3) = \{x_3\}$.

$x_4 = \langle Y : Z\ e\_, abce\rangle$，$x'_4 = \langle S : X\_Y\ d, a\rangle$，$y_4 = \langle S : X\ Y\_d, abce\rangle$ とすると，$x_4 \in \mathrm{Reduce}_{(0)}(E_{abce}) \cap \mathrm{Redex}$，$x'_4 \in \mathrm{Parent}(x_4) \cap \mathrm{R0eitem}(x_4)$，$y_4 \in \mathrm{Reduce0}(x'_4, x_4)$ を満たすので，アルゴリズム 2 の場合 (4) より $y_4 \in E_{abce}$ が成り立つ．これは図 12 の E_{abce} の上の拡張項（$\langle Y : Z\ e\_, abce\rangle$）から E_{abce} の下の拡張項（$\langle S : X\ Y\_d, abce\rangle$）を生成する場合に対応する．このとき場合 (4) の $\mathrm{Parent}(y)$ の定義より，

Parent(y_4)
$= \{w \mid w \in \mathrm{Parent}(z'), z' \in \mathrm{Parent}(z)$
$\cap \mathrm{R0eitem}(z), z \in \mathrm{Reduce}_{(0)}(E_{abce}) \cap \mathrm{Redex},$
$y_4 \in \mathrm{Reduce0}(z', z)\}$
であり，$x''_4 = \langle S : X\_Y\ d, ab\rangle$ とすると
$\mathrm{Reduce}_{(0)}(E_{abce}) \cap \mathrm{Redex} = \{x_4\}$,
$\mathrm{Parent}(x_4) \cap \mathrm{R0eitem}(x_4) = \{x'_4, x''_4\}$,
$\mathrm{Reduce0}(x'_4, x_4) = \mathrm{Reduce0}(x''_4, x_4) = \{y_4\}$,
$\mathrm{Parent}(x'_4) = \mathrm{Parent}(x''_4) = \{e_0\}$
だから，
$\mathrm{Parent}(y_4) = \mathrm{Parent}(x'_4) \cup \mathrm{Parent}(x''_4) = \{e_0\}$.

$x_5 = \langle Z : b\ c\_, abc\rangle$，$x'_5 = \langle S : X\_Y\ d, a\rangle$，$y_5 = \langle Y : Z\_e, abc\rangle$ とすると，$x_5 \in \mathrm{Reduce}_{(0)}(E_{abc}) \cap \mathrm{Redex}$，$x'_5 \in \mathrm{Parent}(x_5) \cap \mathrm{R1eitem}(x_5)$，$y_5 \in \mathrm{Reduce1}(x'_5, x_5)$ を満たすので，アルゴリズム 2 の場合 (5) より，$y_5 \in E_{abc}$ が成り立つ．これは図 12 の E_{abc} の最も上の拡張項（$\langle Z : b\ c\_, abc\rangle$）から E_{abc} の中央の拡張項（$\langle Y : Z\_e, abc\rangle$）を生成する場合に対応する．

このとき場合 (5) の $\mathrm{Parent}(y)$ の定義より，
$\mathrm{Parent}(y_5) =$
$\{z' \mid z' \in \mathrm{Parent}(z) \cap \mathrm{R1eitem}(z),$
$z \in \mathrm{Reduce}_{(0)}(E_{abc}) \cap \mathrm{Redex}, y_5 \in \mathrm{Reduce1}(z', z)\}$
であり，$z_5 = \langle Z : c\_, abc\rangle$，$z'_5 = \langle S : X\_Y\ d, ab\rangle$ とすると，
$\mathrm{Reduce}_{(0)}(E_{abc}) \cap \mathrm{Redex} = \{x_5, z_5\}$,
$\mathrm{Parent}(x_5) \cap \mathrm{R1eitem}(x_5) = \{x'_5\}$，$\mathrm{Parent}(z_5) \cap \mathrm{R1eitem}(z_5) = \{z'_5\}$,
$\mathrm{Reduce1}(x'_5, x_5) = \mathrm{Reduce1}(z'_5, z_5) = \{y_5\}$
だから，
$\mathrm{Parent}(y_5) = \{x'_5, z'_5\}$.

これらのグラフ構造スタックの要素の生成状況を**図 13** に示す．

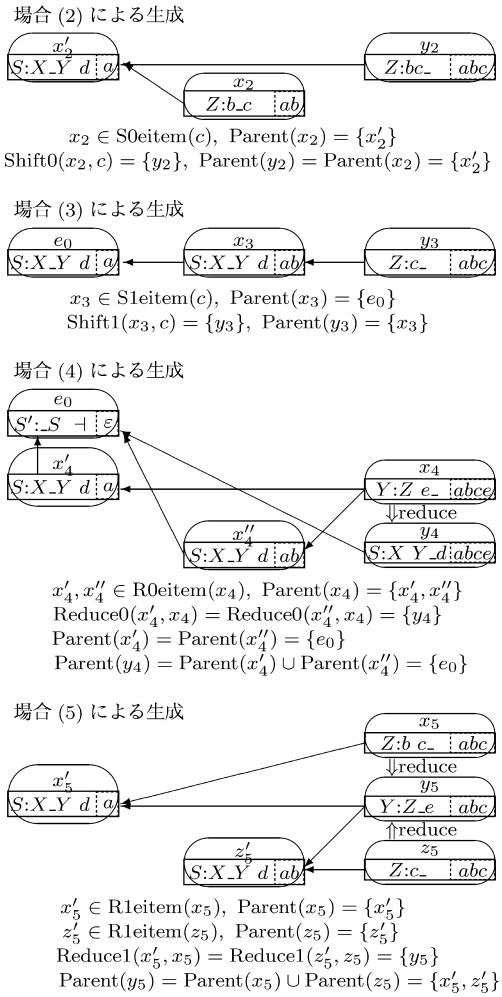

図 13 グラフ構造スタックの生成例
Fig. 13 Example of Graph Structured Stack Generation.

補題 1 $a \in V_T$，$\alpha \in V_T{}^*$，$x \in E_\alpha$ に対して，$x \in \mathrm{S0eitem}(a)$ および $y \in \mathrm{Shift0}(x, a)$ が成り立つ場合は，
$\mathrm{Parent}(x) = \mathrm{Parent}(y)$
が成り立つ．

証明： $y = \langle\langle r, n\rangle, \alpha \cdot a\rangle$ と表現すると，x は $\langle\langle r, n-1\rangle, \alpha\rangle$ と表せるので，y に対して一意に定まる．つまり x はアルゴリズム 2 の場合 (2) の $\mathrm{Parent}(y)$ の定義中の，z の条件を満たす唯一の拡張項である． □

定義 11（E^α） $\alpha \in V_T{}^*$ に対して，E^α を次のように定義する．
$E^\alpha = \{\sigma \mid \mathrm{Parent}(\sigma_1) = \emptyset,$
$\sigma_j \in \mathrm{Parent}(\sigma_{j+1})(1 \le j < \sharp\sigma), \sigma_\sharp \in E_\alpha\}$.
E^α の要素は，E_α の Parent 集合の要素をたどることにより構成される拡張項の列である．

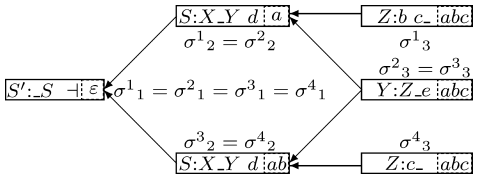

図 14 $G1$ における $E^{abc} = \{\sigma^1, \sigma^2, \sigma^3, \sigma^4\}$ の要素
Fig. 14 $E^{abc} = \{\sigma^1, \sigma^2, \sigma^3, \sigma^4\}$ in $G1$.

例 8 例 1 の $G1$ において，E^{abc}=$\{\sigma^1, \sigma^2, \sigma^3, \sigma^4\}$，ただし

$$\sigma^1 = ({\sigma^1}_1, {\sigma^1}_2, {\sigma^1}_3) = (e_0, \langle S : X\_Y\ d, a\rangle, \langle Z : b\ c\_, abc\rangle),$$
$$\sigma^2 = ({\sigma^2}_1, {\sigma^2}_2, {\sigma^2}_3) = (e_0, \langle S : X\_Y\ d, a\rangle, \langle Y : Z\_e, abc\rangle),$$
$$\sigma^3 = ({\sigma^3}_1, {\sigma^3}_2, {\sigma^3}_3) = (e_0, \langle S : X\_Y\ d, ab\rangle, \langle Y : Z\_e, abc\rangle),$$
$$\sigma^4 = ({\sigma^4}_1, {\sigma^4}_2, {\sigma^4}_3) = (e_0, \langle S : X\_Y\ d, ab\rangle, \langle Z : c\_, abc\rangle).$$

σ^1，σ^2，σ^3，σ^4 の要素の関係を**図 14** に示す．この図では e_0 への矢印も表示する．この例の σ^1，σ^2，σ^3，σ^4 は，図 10 の L^{abc} の σ^{31}，σ^{32}，σ^{33}，σ^{34} にそれぞれ対応するので，E^{abc} は L^{abc} と集合として等しい．

L^α と E^α に関する基本的な定理を次に示す．これらはアルゴリズム 1 とアルゴリズム 2 の定義に直接対応しているので，定義に基づき帰納的に証明できる．

定理 1 $\alpha \in {V_T}^*$ に対して，$\lambda \in L^\alpha$ ならば $\lambda \in E^\alpha$.

定理 2 $\alpha \in {V_T}^*$ に対して，$\lambda \in E^\alpha$ ならば $\lambda \in L^\alpha$.

定理 1 と定理 2 から，L^α と E^α は集合として等しいことが分かる．また定理 1 と定理 2 から，アルゴリズム 1 とアルゴリズム 2 は同じ構文解析能力を持ち，構文解析の過程も同じである．

5. 枝刈り式グラフ構造スタック法

本章では本論文の主題である枝刈り式グラフ構造スタック法を述べる．5.1 節の命題 2 が示すように，拡張項の集合が分離的な（すべての要素の body 部が異なる）場合は，その集合の要素の数は固定値（I_G の要素数）以下となる．このため Parent 集合がすべて分離的である場合，グラフ構造スタック法は効率的である．本章では分離的でない Parent 集合を，還元時に枝刈りを行って分離的にすることにより，効率的に構文解析を行うアルゴリズムを述べる．ただしこのような枝刈りを行うことができない文脈自由文法も存在する．

5.1 枝刈り式グラフ構造スタック法の概要

有限集合 H の大きさ（要素数）を $|H|$ で表す．

定義 12（Succ） 項 $i = \langle r, n\rangle \in I_G$ $(0 \le n < \sharp r)$ に対して

$$\mathrm{Succ}(i) = \langle r, n+1\rangle$$

と定義する．

2 つの拡張項に対して，一方から他方がアルゴリズム 2 の場合 (2)–場合 (5) によって生成されることを表す二項関係を以下に定義する．

定義 13（$\mapsto$） 拡張項 x，$y \in E$ に対して

(1) $x \in \mathrm{S0eitem}(a)$，$y \in \mathrm{Shift0}(x, a)$，
(2) $x \in \mathrm{S1eitem}(a)$，$y \in \mathrm{Shift1}(x, a)$，
(3) $x \in \mathrm{Redex}$，$y \in \mathrm{Reduce0}(x', x)$

ただし x' は $x' \in \mathrm{R0eitem}(x) \cap \mathrm{Parent}(x)$ を満たす，
(4) $x \in \mathrm{Redex}$，$y \in \mathrm{Reduce1}(x', x)$

ただし x' は $x' \in \mathrm{R1eitem}(x) \cap \mathrm{Parent}(x)$ を満たす，

のいずれかが成り立つとき，$x \mapsto y$ と表す．$\mapsto$ の推移的閉包を $\mapsto^*$ で表す．

例 9 例 7 で述べたように，例 1 の $G1$ ではアルゴリズム 2 の場合 (2) により，$\langle Z : b\_c, ab\rangle$ から $\langle Z : bc\_, abc\rangle$ が生成されるから，$\langle Z : b\_c, ab\rangle \mapsto \langle Z : bc\_, abc\rangle$ が成り立つ．同様に，

$$e_0 \mapsto \langle X : a\_, a\rangle \mapsto \langle S : X\_Y d, a\rangle \mapsto \langle Z : b\_c, ab\rangle \mapsto \langle Z : bc\_, abc\rangle \mapsto \langle Y : Z\_e, abc\rangle$$

が成り立つので，

$\langle S : X\_Y d, a\rangle \mapsto^* \langle Y : Z\_e, abc\rangle$ が成り立つ．

$\mapsto$ と $\mathrm{Parent}(x)$ の定義より次の命題が成り立つ．

命題 1 $y \in \mathrm{Parent}(x)$ ならば $y \mapsto^* x$.

定義 14（分離的） 拡張項の集合 $D \subseteq J_G$ は，$\mathrm{body}(x) = \mathrm{body}(y)$ を満たす任意の x，$y \in D$ に対して $x = y$ が成り立つとき，分離的（separate）であるという．

アルゴリズム 2 で定義する E_α はすべての要素の input 部が同一（α）なので，E_α は分離的である．

アルゴリズム 2 の場合 (4) における $\mathrm{Parent}(x)$ が分離的な場合は，次の補題が成り立つ．

補題 2 x，x'，y が $x \in \mathrm{Redex}$，$x' \in \mathrm{R0eitem}(x) \cap \mathrm{Parent}(x)$，$y \in \mathrm{Reduce0}(x', x)$ を満たす場合に，$\mathrm{Parent}(x)$ が分離的であれば，$\mathrm{Parent}(y) = \mathrm{Parent}(x')$ が成り立つ．

証明： $z' \in \mathrm{R0eitem}(x) \cap \mathrm{Parent}(x)$，$y \in \mathrm{Reduce0}(z', x)$ を満たす z' が存在したとすると，

$y \in \mathrm{Reduce0}(x', x)$，$y \in \mathrm{Reduce0}(z', x)$ より $\mathrm{body}(x') = \mathrm{body}(z')$ が成り立つので，$\mathrm{Parent}(x)$ が分離的であれば，$x' = z'$ となる．つまりこの場合は $y \in \mathrm{Reduce0}(x', x)$ を満たす x' はただ 1 つなので，アルゴリズム 2 の場合 (4) における $\mathrm{Parent}(y)$ の定義より，$\mathrm{Parent}(y) = \mathrm{Parent}(x')$ が成り立つ． □

D が分離的であれば，各要素の input 部を無視することにより，D は LR(0) 項の集合 I_G の内部に 1 対 1 に自然に対応させることができる．たとえば E_{abc} は分離的なので，E_{abc} の要素の input 部を無視した要素の集合 $\{Z : bc\_, Y : Z\_e, Z : c\_\}$ は I_G の部分集合となり，I_G に自然に対応させることができる．このことから次の命題が成り立つ．

命題 2 $D \subseteq J_G$ に対して，D が分離的であれば $|D| \leq |I_G|$．つまり分離的な集合は有限集合である．

命題 2 より，分離的な集合の大きさは，$|I_G|$ つまり文法 G には依存するが入力列の長さ n には依存しない定数値以下である．

定義 14 より E_α は分離的である．さらに E_α のすべての要素 x に対して $\mathrm{Parent}(x)$ が分離的であれば，グラフ構造スタック法は効率的である．しかし残念ながら分離的でない $\mathrm{Parent}(x)$ も存在する．

例 10 図 12 に示されるように，$\mathrm{Parent}(\langle Y : Z\_e,\ abc\rangle)$ は，$\langle S : X\_Yd,\ a\rangle$ と $\langle S : X\_Yd,\ ab\rangle$ の両方を含むので分離的でない．

分離的でない Parent 集合には次の定理が成り立つ．

定理 3 $x_1, x_2 \in \mathrm{Parent}(x)$, $\mathrm{body}(x_1) = \mathrm{body}(x_2)$, $\mathrm{input}(x_1) \neq \mathrm{input}(x_2)$ を満たす x，x_1，x_2 に対して，$\mathrm{body}(x_1) = \mathrm{body}(x_2) = i$ とすると，$x \mapsto^* \langle \mathrm{Succ}(i), \alpha\rangle$ を満たす $\alpha \in {V_T}^*$ が存在すれば，

$x_1 \mapsto^* \langle \mathrm{Succ}(i), \alpha\rangle$,
$x_2 \mapsto^* \langle \mathrm{Succ}(i), \alpha\rangle$

が成り立つ．

証明：x_1，$x_2 \in \mathrm{Parent}(x)$ より，$x_1 \mapsto^* x$，$x_2 \mapsto^* x$ なので，$x' = \langle \mathrm{Succ}(i), \alpha\rangle$ とすると，$x \mapsto^* x'$ が成り立つから

$x_1 \mapsto^* x \mapsto^* x'$,
$x_2 \mapsto^* x \mapsto^* x'$

となり，題意は成り立つ． □

例 11 例 1 の $G1$ において $x = \langle Y : Z\_e, abc\rangle$，$x_1 = \langle S : X\_Yd,\ a\rangle$，$x_2 = \langle S : X\_Yd,\ ab\rangle$，$x' = \langle S : XY\_d,\ abce\rangle$ とすると，$x_1, x_2 \in \mathrm{Parent}(x)$，$x \mapsto^* x'$ となり，x_1，x_2，x，x' は定理 3 の条件を満たすので $x_1 \mapsto^* x'$，$x_2 \mapsto^* x'$ が成り立つ．

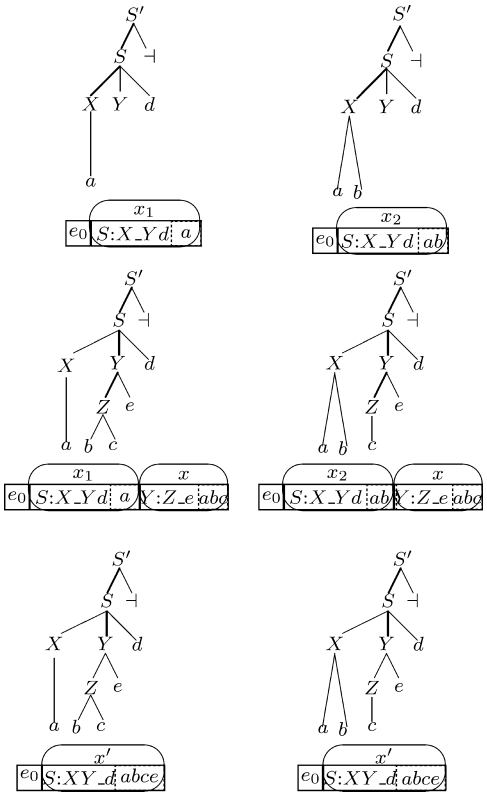

図 15 定理 3 の例：$G1$ における入力 $abce$ の解析
Fig. 15 Example of Theorem 3 : Parse process for $abce$.

定理 3 の状況を**図 15** に示す．図 15 では，左の列の 3 つが $x_1 \mapsto^* x \mapsto^* x'$ に対応する解析過程を表し，右の列の 3 つが $x_2 \mapsto^* x \mapsto^* x'$ に対応する解析過程を表す．中段は abc を読み込み，Z に関する構文規則で還元した時点での解析木と解析スタックを示す．$G1$ のあいまい性のため，この時点で 2 つの解析木と解析スタックが存在する．2 つのスタックとも先頭要素は同じ（x）でその下の要素が異なる（x_1，x_2）．これは図 12 で $\mathrm{Parent}(x)$ が x_1，x_2 を含むことに対応する．

図 15 の下段は，この後で e を読み込み $Y : Z\ e$ で還元した後での解析木と解析スタックを示す．x を還元すると x_1 や x_2 はいずれも x' に shift するから，2 つの解析木のスタックは同じになる．

図 15 に基づき，枝刈りの基本的な考え方を以下に述べる．図 15 の中段の 2 つのスタックは異なるがトップ要素（x）は同じなので，トップ要素が還元されるまで同じ入力に対する 2 つのスタックの動作は同じである．中段の 2 つのスタックはトップ要素の下の要素が異なる（x_1，x_2）が，トップ要素が還元されると，それらは同じ要素（x'）に shift するので 2 つのスタッ

クは同じになる（下段のスタックは同じである）．このためトップ要素の還元後も（2 つのスタックは同じになるので）同じ入力に対する 2 つのスタックの動作は同じである．つまり図 15 の中段の状態以後は，異なる解析木に対応するスタックは同じ入力に対して同じ動作を行う．このため中段の時点で 2 つのスタックのうちどちらか一方を削除しても入力列が語かどうかの判定には影響しない．このように異なる解析木に対応するスタックの動作が同じであれば，それらのスタックのうち 1 つを残し他を削除することにより，削除したスタックに対応する解析木の情報は得られなくなるが，入力列が語かどうかの判定を効率化することができる．これが枝刈りの基本的な考え方である．

5.2 枝刈り式グラフ構造スタック法の定義

本節では分離的でない Parent 集合から枝刈りにより分離的な Parent 集合を生成する方法，およびそれによりグラフ構造スタック法を改良した方法を述べる．

body 部が同じ 2 つの拡張項に対して，それらが同じ拡張項の Parent 集合に含まれるとき一方を残して他方を削除してもよい（枝刈りしてもよい）ことを表す二項関係を以下に定義する．

定義 15（$\prec$） $x \neq e_0$，$y \neq e_0$，$\mathrm{body}(x) = \mathrm{body}(y)$，$\mathrm{input}(x) \neq \mathrm{input}(y)$ を満たす $\alpha_x, \alpha_y \in {V_T}^*$，$x \in E_{\alpha_x}$，$y \in E_{\alpha_y}$ に対して，

$$\forall x' \in \mathrm{Parent}(x)\,.\,\exists y' \in \mathrm{Parent}(y)\ .\ x' = y' \text{ または } x' \prec y'$$

が成り立つ場合に，$x \prec y$ と定義する．

$x \prec y$ は，$\mathrm{Parent}(x) \subseteq \mathrm{Parent}(y)$ を再帰的に表現したものである．定義 15 より，$x \prec y$ が成り立つのは，$\forall x' \in \mathrm{Parent}(x)\,.\,\exists y' \in \mathrm{Parent}(y)\ x' = y'$ が成り立つ場合と $\forall x' \in \mathrm{Parent}(x)\,.\,\exists y' \in \mathrm{Parent}(y)\ x' \prec y'$ が成り立つ場合がある．これらの 2 つの場合の例を**図 16** に示す．

反射則 $x \prec x$ が成立することに注意．

後出の 5.3 節の命題 10 と命題 11 が示すように，異なる x，$y \in \mathrm{Parent}(z)$ に対して $x \prec y$ ならば，$\mathrm{Parent}(z)$ から x を削除しても構文解析は可能である．

例 12 例 11 の x，x_1，x_2 に対して，$\mathrm{Parent}(x_1) = \mathrm{Parent}(x_2) = \{e_0\}$ から，定義 15 の条件を満たすので，$x_1 \prec x_2$ および $x_2 \prec x_1$ が成立する．これは図 15 が示すように，$G1$ では入力 abc の解析時に $\mathrm{Parent}(x)$ から x_1 または x_2 のいずれかを削除（$\mathrm{Parent}(x)$ が x_1 であるスタックまたは x_2 であるスタックのいずれかを削除）できることに相当する．

以下では $\prec$ が推移則を満たすことを示す．以降は，構文解析において $S' : S \dashv$ において S を還元する直前までを考察の対象とする．つまり，body 部が $S' : S\_ \dashv$ や $S' : S \dashv \_$ である拡張項は E に含まれないとする．

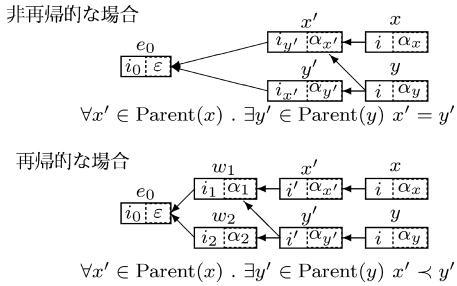

図 16 $x \prec y$ となる場合の例
Fig. 16 Examples of $x \prec y$.

アルゴリズム 2 が示すように，E の要素はすべて e_0 から構成的に生成され，新たに生成された要素の Parent 集合の要素はすべてすでに生成されている要素である．E の任意の要素 x に対して Parent 集合の要素をとり，その要素に対してさらに Parent 集合の要素をとる，という処理を繰り返すとつねに有限回で e_0 になる．アルゴリズム 2 の Parent 集合の定義により，input 部の長さは Parent 集合の要素の方がつねに小さいので，Parent 集合をたどる操作がループに陥ることはない．E の要素 x に対して，x から e_0 まで Parent 集合をとる回数の最大値を $\mathrm{Level}(x)$ と表すと，$\mathrm{Level}(x)$ は次のように定義できる．

定義 16（Level） $x \in E$ に対して，

$x = e_0$ の場合，

$$\mathrm{Level}(x) = 0,$$

$x \neq e_0$ の場合，

$$\mathrm{Level}(x) = 1 + \max\{\mathrm{Level}(x') \mid x' \in \mathrm{Parent}(x)\}.$$

例 11 の x，x_1，x_2 に対して，図 12 より，$\mathrm{Level}(x_1) = \mathrm{Level}(x_2) = 1$，$\mathrm{Level}(x) = 2$ が成り立つ．

以下では $\mathrm{Level}(x)$ に関する帰納法により $\prec$ の推移則を示す．定義 15 より，$e_0 \prec x$ または $x \prec e_0$ を満たす E の要素 $x \neq e_0$ は存在しないから，次の補題が成り立つ．

補題 3 $x \prec y$ を満たす x，$y \in E$ に対して，$e_0 \in \mathrm{Parent}(x)$ ならば $e_0 \in \mathrm{Parent}(y)$．

定義 16 より，次の 2 つの補題が成り立つ．

補題 4 $x \in E$ に対して，$x \neq e_0$ ならば $\mathrm{Level}(x) \geq 1$．

補題 5 x，$x' \in E$ に対して，$x' \in \mathrm{Parent}(x)$ ならば $\mathrm{Level}(x') < \mathrm{Level}(x)$．

$\prec$ と $\mathrm{Level}(x)$ に関して次の補題が成り立つ．

補題 6 $x, y \in E$ に対して，$x \prec y$ ならば $\mathrm{Level}(x) \leq \mathrm{Level}(y)$.

証明：$\mathrm{Level}(x)$ に関する帰納法により証明する．

（1） $\mathrm{Level}(x) = 1$ の場合．定義 16 より，$\mathrm{Parent}(x) = \{e_0\}$ となるので，$x \prec y$ と補題 3 より，$e_0 \in \mathrm{Parent}(y)$ が成り立つ．ゆえに，

$$\begin{aligned}\mathrm{Level}(y) &= 1 + \max\{\mathrm{Level}(y') \mid y' \in \mathrm{Parent}(y)\} \\ &\geq 1 + \mathrm{Level}(e_0) \\ &= 1 = \mathrm{Level}(x)\end{aligned}$$

となり，題意は成り立つ．

（2） $\mathrm{Level}(x) \leq k$ の場合に題意が成立しているとして，$\mathrm{Level}(x) = k+1$ の場合を証明する．$x \prec y$ より，$\mathrm{Parent}(x)$ の任意の要素 x' に対して，$x' = y'$ または $x' \prec y'$ を満たす $\mathrm{Parent}(y)$ の要素 y' が存在する．$x' = y'$ の場合は，$\mathrm{Level}(x') = \mathrm{Level}(y')$ が成り立つ．$x' \prec y'$ の場合は，$x' \in \mathrm{Parent}(x)$ だから，補題 5 より $\mathrm{Level}(x') < \mathrm{Level}(x) = k+1$ となり，帰納法の仮定により，$\mathrm{Level}(x') \leq \mathrm{Level}(y')$ が成り立つ．ゆえに，

$$\begin{aligned}\mathrm{Level}(x) &= 1 + \max\{\mathrm{Level}(x') \mid x' \in \mathrm{Parent}(x)\} \\ &\leq 1 + \max\{\mathrm{Level}(y') \mid y' \in \mathrm{Parent}(y)\} \\ &= \mathrm{Level}(y)\end{aligned}$$

となり，題意は成り立つ． □

命題 3（$\prec$ の推移則） $x, y \in E$ に対して，$x \prec y$, $y \prec z$ ならば $x \prec z$.

証明：$\mathrm{Level}(z)$ の値に関する帰納法により証明する．

（1） $\mathrm{Level}(z) = 1$ の場合．$x \prec y$, $y \prec z$ だから，補題 6 より $\mathrm{Level}(x) \leq \mathrm{Level}(y)$ および $\mathrm{Level}(y) \leq \mathrm{Level}(z)$ が成り立つ．ゆえに $\mathrm{Level}(z) = 1$ と補題 4 より $\mathrm{Level}(x) = \mathrm{Level}(y) = \mathrm{Level}(z) = 1$ となるので，定義 16 より $\mathrm{Parent}(x) = \mathrm{Parent}(y) = \mathrm{Parent}(z) = \{e_0\}$ が成り立つ．ゆえに，この場合は題意は成り立つ．

（2） $\mathrm{Level}(z) \leq k$ のとき推移則が成立するとして，$\mathrm{Level}(z) = k+1$ の場合を証明する．$x \prec y$ より，$\mathrm{Parent}(x)$ の任意の要素 x' に対して，$x' = y'$ または $x' \prec y'$ を満たす $\mathrm{Parent}(y)$ の要素 y' が存在する．$x' = y'$ の場合は，$x' \in \mathrm{Parent}(y)$ となるので，$y \prec z$ より，この x' に対して $x' = z'$ または $x' \prec z'$ を満たす $\mathrm{Parent}(z)$ の要素 z' が存在する．ゆえにこの場合は $x \prec z$ が成り立つ．$x' \prec y'$ の場合は，$y \prec z$ より，この y' に対して $y' = z'$ または $y' \prec z'$ を満たす $\mathrm{Parent}(z)$ の要素 z' が存在する．$y' = z'$ の場合は，$x' = y'$ の場合と同様に $x \prec z$ が成り立つ．$y' \prec z'$ の場合は $x' \prec y', y' \prec z'$ が成り立つが，$z' \in \mathrm{Parent}(z)$ と補題 6 より $\mathrm{Level}(z') < \mathrm{Level}(z) = k+1$ となるので，帰納法の仮定により $x' \prec z'$ が成り立ち，$x \prec z$ が成り立つ．ゆえに $\mathrm{Level}(z) = k+1$ の場合も題意は成り立つ． □

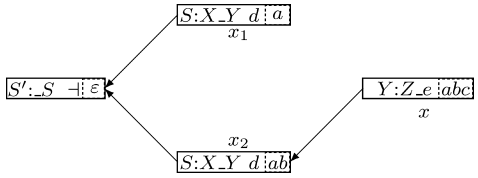

図 17 $G1$ での $\mathrm{Prune}(\mathrm{Parent}(\langle Y : Z\_e, abc\rangle))$
Fig. 17 $\mathrm{Prune}(\mathrm{Parent}(\langle Y : Z\_e, abc\rangle))$ in $G1$.

定義 17（代表元，代表可能） $D \subseteq E$, $x \in D$ に対して，$D_x = \{y \mid y \in D,\ \mathrm{body}(y) = \mathrm{body}(x)\}$ とするとき，D_x の任意の要素 z に対して，$z \prec p_x$ を満たす D_x の要素 p_x を x の代表元という．D の任意の要素 x に対して x の代表元が存在するとき，D は代表可能という．

D_x は，D の要素のうち body 部が x の body 部と同じである要素の集合であり，x の代表元は D_x の要素に枝刈りを行う際に残す要素を表す．D が代表可能であるとは，D の要素のうち body 部が同じ要素は代表元（の 1 つ）だけを残して枝刈り（削除）できることを表す．

一般に，次の例 13 で示すように，代表元は複数個ありえて，そのどれでも今後の議論には差し支えない．

定義 18（Prune） 代表可能な $D \subseteq E$ に対して，$x \in D$, $P_x = \{p_x \mid p_x \text{ は } x \text{ の代表元 }\}$ とする．各 x に対して P_x の中から 1 つ代表元を選んで，それらをすべての x に対して集めたものを $\mathrm{Prune}(D)$ と定義する．

$\mathrm{Prune}(D)$ は，D の要素のうち body 部が同じ要素は代表元（の 1 つ）だけを残した集合である．

例 13 例 11 の x, x_1, x_2 において，$\mathrm{Parent}(x) = \{x_1, x_2\}$, $x_1 \prec x_2$, $x_2 \prec x_1$ だから x_1, x_2 のどちらも代表元であり，$\mathrm{Parent}(x)$ は代表可能である．x_2 を $\mathrm{Prune}(\mathrm{Parent}(x))$ の要素とした場合の関係を**図 17** に示す．

定義 18 から次の命題が得られる．

命題 4 $D \subseteq E$ に対して，$\mathrm{Prune}(D)$ が存在すれば，$\mathrm{Prune}(D)$ は分離的である．

本論文のアルゴリズムが対象とする文脈自由文法を次に定義する．

定義 19（枝刈り可能文法） 文脈自由文法 G において，E の任意の要素 x に対して $\mathrm{Parent}(x)$ が代表可能である場合に，G を枝刈り可能文法という．

例 14 例 13 で述べたように，例 1 の $G1$ では $\mathrm{Parent}(x)$ は代表可能だから，$G1$ は枝刈り可能文法

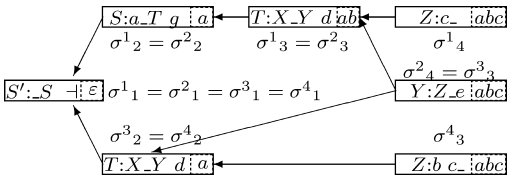

図 18 文法 $G2$ における枝刈り可能でない場合
Fig. 18 Unprunable case in $G2$.

である．

例 15 $G2$ を次の構文規則を持つ文脈自由文法とする．

$(r^0)\ S' : S\ \dashv$　$(r^1)\ S : T\ f$　$(r^2)\ S : a\ T\ g$
$(r^3)\ T : X\ Y\ d$　$(r^4)\ X : a$　$(r^5)\ X : b$
$(r^6)\ Y : Z\ e$　$(r^7)\ Z : c$　$(r^8)\ Z : b\ c$

$G2$ における $E^{abc} = \{\sigma^1, \sigma^2, \sigma^3, \sigma^4\}$ の要素の関係を**図 18** に示す．図 18 より，$\langle T : X\_Y\ d, a\rangle \prec \langle T : X\_Y\ d, ab\rangle$ と $\langle T : X\_Y\ d, ab\rangle \prec \langle T : X\_Y\ d, a\rangle$ のいずれも成り立たないから，Parent($\langle Y : Z\_e, abc\rangle$) は代表可能でないので $G2$ は枝刈り可能文法ではない．

アルゴリズム 3（枝刈り式グラフ構造スタック法）
枝刈り可能文法に対して，アルゴリズム 2（グラフ構造スタック法）の場合 (5) の定義中の「Parent(y) = P′(y)」を，「Parent(y) = Prune(P′(y))」に置き換えたアルゴリズムを，枝刈り式グラフ構造スタック法とよぶ．

命題 5 アルゴリズム 3 の定義中（アルゴリズム 2 参照）の x と y を考える．Parent(x) が分離的であれば，場合 (2)，(3)，(4) で定義される y に対する Parent(y) も分離的である．
証明：場合 (2) では，補題 1 より Parent(y) = Parent(x) であり，Parent(x) は分離的だから Parent(y) は分離的である．場合 (3) では，Parent(y) $\subseteq E_\alpha$ であり，E_α は分離的だから Parent(y) は分離的である．場合 (4) では，Parent(x) は分離的だから補題 2 より Parent(y) は分離的である． □

命題 5 より，グラフ構造スタック法では，Parent(x) が分離的であれば，アルゴリズム 2 の場合 (5) のみ Parent(y) を Prune(Parent(y)) で置き換えれば，すべての Parent(y) は分離的になる．

アルゴリズム 2 では枝刈りを行わずに E_α を生成するのに対して，アルゴリズム 3 では枝刈りを行いつつ E_α を生成する．このため，アルゴリズム 2 に基づいて E_α を生成した場合とアルゴリズム 3 に基づいて E_α を生成した場合では，$x \in E_\alpha$ に対する Parent(x) が異なる場合が存在する．しかしこの場合でも次の命題が成り立つ．

命題 6 $D \subseteq E$ が代表可能であれば，Prune(D) は代表可能である．
証明： $y \in D$ を枝刈りにより削除される要素とすると，Prune の定義により $y \prec z$ を満たす D の要素 z が Prune(D) の中に存在する．ゆえに $x \prec y$ を満たす D の要素 x に対しては，命題 3 により $x \prec z$ が成り立つから，y が削除されても Prune(D) には x の代表元は存在する． □

命題 6 より，$x \in E_\alpha$ に対して，E_α をアルゴリズム 2 に基づいて枝刈りを行わずに生成した場合に Parent(x) が代表可能であれば，E_α をアルゴリズム 3 に基づいて枝刈りを行って生成した場合にも Parent(x) は代表可能である．ゆえに，枝刈り可能文法に対してアルゴリズム 3 を適用できる．

5.3 枝刈り式グラフ構造スタック法の正当性

本節ではグラフ構造スタック法の Parent(x) を Prune(Parent(x)) に置き換えても，結果（入力列が文法 G の語であるか）が変わらないことを示す．これにより枝刈り可能文法に対する枝刈り式グラフ構造スタック法の正当性が示される．まず，定義 13 の拡張項間の関係である $\mapsto^*$ と構文解析との間の関係を示す．

命題 7 $\alpha' \in {V_T}^*$ が G の語であることは，

$$\langle i_0, \varepsilon\rangle = \langle S' : \_S\ \dashv, \varepsilon\rangle \mapsto^* \langle S' : S\_\ \dashv, \alpha'\rangle = \langle \mathrm{Succ}(i_0), \alpha'\rangle$$

と同値である．

命題 8 $\alpha \in {V_T}^*$ に対して，$\alpha \cdot \beta$ が G の語である $\beta \in {V_T}^*$ が存在する場合は，

$$\langle i_0, \varepsilon\rangle \mapsto^* \sigma_\sharp \mapsto^* \langle \mathrm{Succ}(i_0), \alpha \cdot \beta\rangle$$

を満たす $\sigma \in E^\alpha$ が存在する．

命題 8 の条件を満たす σ に対して，定理 2 より $\sigma \in L^\alpha$ だから σ は解析スタックであり，各 σ_j ($1 \le j \le \sharp\sigma$) は解析スタックの要素である．$\sigma_\sharp \mapsto^* \langle S' : S\_\ \dashv, \alpha\cdot\beta\rangle$ が成り立つから，今後 β を読み込むと最終的に σ のすべての要素 σ_j ($1 \le j \le \sharp\sigma$) は還元されスタックからポップされる．σ_j が還元された後に，その Parent 集合の要素 σ_{j-1} の body 部 (body(σ_{j-1})) は Succ(body(σ_{j-1})) に変化する．Succ(body(σ_{j-1})) に変化した後に，さらにその Parent 集合の要素 σ_{j-2} の body 部は Succ(body(σ_{j-2})) に変化する．以下同様に成立するので，次の命題が成り立つ．

命題 9 $\alpha \in {V_T}^*$ に対して，$\alpha \cdot \beta$ が G の語である $\beta \in {V_T}^*$ が存在することは，次の条件を満たす

$\sigma \in E^\alpha$，$\alpha'_j \in {V_T}^*$（$1 \le j < \sharp\sigma$）が存在することと同値である．

(1) $\sigma_1 = \langle i_0, \varepsilon\rangle$，$\alpha'_1 = \alpha \cdot \beta$，

(2) $\langle \mathrm{Succ}(\mathrm{body}(\sigma_j)), \alpha'_j\rangle \mapsto^* \langle \mathrm{Succ}(\mathrm{body}(\sigma_{j-1})), \alpha'_{j-1}\rangle$ $(1 < j < \sharp\sigma)$，

(3) $\sigma_\sharp \mapsto^* \langle \mathrm{Succ}(\mathrm{body}(\sigma_{\sharp\sigma-1})), \alpha'_{\sharp\sigma-1}\rangle$．

例 16 例 1 の $G1$ において，$\alpha = abc \in {V_T}^*$ に対して，$\alpha \cdot \beta$ が $G1$ の語である $\beta = ed \in {V_T}^*$ が存在し，次の条件を満たす $\sigma = \{\sigma_1 = \langle i_0, \varepsilon\rangle, \sigma_2 = \langle S : X\_Yd, a\rangle, \sigma_3 = \langle Y : Z\_e, abc\rangle\} \in E^{abc}$，$\alpha'_1 = abced$，$\alpha'_2 = abce \in {V_T}^*$ が存在する．

$\sigma_3 = \langle Y : Z\_e, abc\rangle \mapsto^* \langle S : XY\_d, abce\rangle = \langle \mathrm{Succ}(\mathrm{body}(\sigma_2)), \alpha'_2\rangle$，

$\langle \mathrm{Succ}(\mathrm{body}(\sigma_2)), \alpha'_2\rangle = \langle S : XY\_d, abce\rangle \mapsto^* \langle \mathrm{Succ}(i_0), abced\rangle = \langle \mathrm{Succ}(\mathrm{body}(\sigma_1)), \alpha'_1\rangle$

定義 20（I^α） $\alpha \in {V_T}^*$ に対して，$I^\alpha \subseteq {I_G}^*$ を次のように定義する．

$$I^\alpha = \{\rho \mid \exists \sigma \in E^\alpha \text{ ただし } \sharp\rho = \sharp\sigma, \ \rho_j = \mathrm{body}(\sigma_j)\ (1 \le j \le \sharp\rho)\}.$$

I^α の要素は，E^α の要素の body 部を取り出したものである．

例 17 例 1 の $G1$ において，$I^{abc} = \{\rho^1, \rho^2, \rho^3, \rho^4\}$ である．ただし

$\rho^1 = (\rho^1{}_1, \rho^1{}_2, \rho^1{}_3) = (i_0, S : X\_Yd, Z : bc\_)$.

$\rho^2 = (\rho^2{}_1, \rho^2{}_2, \rho^2{}_3) = (i_0, S : X\_Yd, Y : Z\_e)$,

$\rho^3 = (\rho^3{}_1, \rho^3{}_2, \rho^3{}_3) = (i_0, S : X\_Yd, Y : Z\_e)$,

$\rho^4 = (\rho^4{}_1, \rho^4{}_2, \rho^4{}_3) = (i_0, S : X\_Yd, Z : c\_)$.

I^{abc} の要素 ρ^1，ρ^2，ρ^3，ρ^4 は，例 8 の E^{abc} の要素 σ^1，σ^2，σ^3，σ^4 の body 部を取り出したものである．たとえば $\sigma^1 = (e_0, \langle S : X\_Y\ d, a\rangle, \langle Z : b\ c\_, abc\rangle)$ の body 部を取り出したものが $\rho^1 = (i_0, S : X\_Yd, Z : bc\_)$ である．

定義 21（E^x，I^x） 拡張項 $x \in E_\alpha$ に対して，

$$E^x = \{\sigma \mid \sigma \in E^\alpha, \sigma_\sharp = x\} \subseteq E^\alpha,$$

$$I^x = \{\rho \mid \exists \sigma \in E^x, \sharp\rho = \sharp\sigma, \ \rho_j = \mathrm{body}(\sigma_j)\ (1 \le j \le \sharp\rho)\} \subseteq I^\alpha.$$

E^x は E^α の要素 σ のうち $\sigma_\sharp = x$ を満たす要素の集合であり，I^x は E^x の body 部を取り出したものに対応する．

例 18 例 1 の $G1$ において $x = \langle Y : Z\_e, abc\rangle$ とすると，$x \in E_{abc}$ だから E^x は E^{abc} の要素 σ のうち $\sigma_\sharp = x$ を満たす要素の集合である．例 8 より E^{abc} の要素 σ^1，σ^2，σ^3，σ^4 に対して，$\sigma^1_\sharp \ne x$，$\sigma^2_\sharp = x$，$\sigma^3_\sharp = x$，$\sigma^4_\sharp \ne x$ だから，$E^x = \{\sigma^2, \sigma^3\}$ となる．

σ^2，σ^3 の各要素の関係を**図 19** に示す．また，例 12 より E^x の $\mathrm{Parent}(x)$ を $\mathrm{Prune}(\mathrm{Parent}(x))$ で置き換え

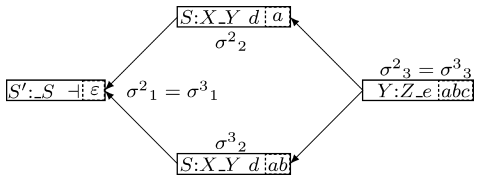

図 19 $G1$ における $E^{\langle Y:Z\_e,abc\rangle} = \{\sigma^2, \sigma^3\}$ の要素
Fig. 19 Elements in $E^{\langle Y:Z\_e,abc\rangle} = \{\sigma^2, \sigma^3\}$.

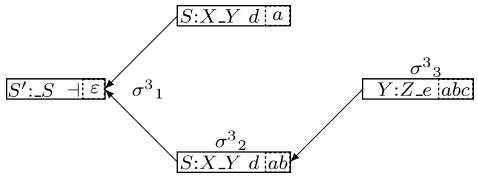

図 20 $G1$ における $E^{\langle Y:Z\_e,abc\rangle}$（Prune 版）の要素
Fig. 20 Elements in $E^{\langle Y:Z\_e,abc\rangle}$ (Prune version).

た関係を**図 20** に示す．図 19，図 20 から分かるように，この場合に E^x の $\mathrm{Parent}(x)$ を $\mathrm{Prune}(\mathrm{Parent}(x))$ で置き換えると σ^2 が失われる．しかし $I^x = \{\rho_2, \rho_3\}$ とすると，$\rho_2 = \rho_3 = \{i_0,\ S : X\_Yd,\ Y : Z\_e\}$ だから，$\mathrm{Parent}(x)$ を $\mathrm{Prune}(\mathrm{Parent}(x))$ で置き換えても I^x は変わらない．

命題 9 は I^α により次のように述べることができる．

命題 10 $\alpha \in {V_T}^*$ に対して，$\alpha \cdot \beta$ が G の語である $\beta \in {V_T}^*$ が存在することは，次の条件を満たす $\rho \in I^\alpha$，$x \in E_\alpha$，$\alpha'_j \in {V_T}^*$（$1 \le j < \sharp\rho$）が存在することと同値である．

(1) $\rho_1 = i_0, \alpha'_1 = \alpha \cdot \beta$，

(2) $\langle \mathrm{Succ}(\rho_j), \alpha'_j\rangle \mapsto^* \langle \mathrm{Succ}(\rho_{j-1}), \alpha'_{j-1}\rangle$ $(1 < j < \sharp\rho)$，

(3) $x = \langle \rho_\sharp, \alpha\rangle \mapsto^* \langle \mathrm{Succ}(\rho_{\sharp\rho-1}), \alpha'_{\sharp\rho-1}\rangle$．

命題 10 より，すべての $x \in E_\alpha$ に対して I^x を保つような変更をグラフ構造スタック法に行っても，変更後のアルゴリズムの結果は変更前と同じであることが分かる．E^α，I^α，E^x，I^x の定義より，次の補題が成り立つ．

補題 7 $\alpha \in {V_T}^*$ に対して

$E^\alpha = \bigcup_{x \in E_\alpha} E^x$，　$I^\alpha = \bigcup_{x \in E_\alpha} I^x$.

補題 8 $x \in E$ に対して，

$$E^x = \bigcup_{y \in \mathrm{Parent}(x)} \{\sigma \cdot x \mid \sigma \in E^y\},$$

$$I^x = \bigcup_{y \in \mathrm{Parent}(x)} \{\rho \cdot \mathrm{body}(x) \mid \rho \in I^y\}.$$

これらの補題から，次の命題が成り立つ．

命題 11 $x \prec y$ ならば $I^x \subseteq I^y$.

証明：$\mathrm{Level}(x)$ に関する帰納法により証明する．

(1) $\mathrm{Level}(x) = 1$ の場合．定義 16 より，

Parent$(x)=\{e_0\}$ となるので，$x \prec y$ と補題 3 より，$e_0 \in$ Parent(y) が成り立つ．ゆえに，$I^x=\{i_0\} \subseteq I^y$ となり，題意は成り立つ．

（2） Level$(x) \leq k$ の場合に題意が成立するとして，Level$(x)=k+1$ の場合を証明する．$x \prec y$ より，Parent(x) の任意の要素 x' に対して，$x'=y'$ または $x' \prec y'$ を満たす Parent(y) の要素 y' が存在する．$x'=y'$ の場合は，$I^{x'}=I^{y'}$ が成り立つ．$x' \prec y'$ の場合は，$x' \in$ Parent(x) だから，補題 5 より Level$(x')<$ Level$(x)=k+1$ となり，帰納法の仮定により，$I^{x'} \subseteq I^{y'}$ が成り立つ．ゆえにいずれの場合も，$I^{x'} \subseteq I^{y'}$ が成り立つ．補題 8 より，すべての $\rho \in I^x$ に対して，$\rho=\rho' \cdot \mathrm{body}(x)$ を満たす $x' \in$ Parent(x)，$\rho' \in I^{x'}$ が存在し，$I^{x'} \subseteq I^{y'}$ より，$\rho' \in I^{y'}$ が成り立つ．このため，$\rho=\rho' \cdot \mathrm{body}(x)=\rho' \cdot \mathrm{body}(y) \in I^y$ となり，題意は成り立つ． □

補題 8，命題 11，定義 17 より次の定理が成り立つ．

定理 4 $x \in E$ に対して，

$$\bigcup_{y \in \mathrm{Parent}(x)} I^y = \bigcup_{y \in \mathrm{Prune}(\mathrm{Parent}(x))} I^y.$$

この定理から，$x \in E_\alpha$ に対して Parent(x) を Prune(Parent(x)) に置き換えても，

$$\bigcup_{y \in \mathrm{Parent}(x)} I^y$$

が保存されることが分かる．つまりグラフ構造スタック法の Parent(x) を Prune(Parent(x)) に変更した枝刈り式グラフ構造スタック法も正しいことが分かる．ただし枝刈り式グラフ構造スタック法では枝刈りを行うため，すべての解析木の情報を得ることはできなくなる．

注意：枝刈り可能文法の場合は，命題 11 の逆

$I^x \subseteq I^y$ ならば $x \prec y$

が成り立つ．

証明：$x' \in$ Parent(x)，$\rho' \in I^{x'}$ に対して，$\rho' \cdot \mathrm{body}(x) \in I^x \subseteq I^y$ だから，補題 8 より $\rho' \in I^{y'_{\rho'}}$ を満たす $y'_{\rho'} \in$ Parent(y) が存在する．$x' \neq y'_{\rho'}$ の場合は，この文法は枝刈り可能なので Parent(y) は代表可能だから，$y'_{\rho'}$ の代表元 $p_{y'_{\rho'}}$ が存在する．また補題 8 より，$\rho'=\rho'' \cdot \mathrm{body}(x)=\rho'' \cdot \mathrm{body}(y'_{\rho'})$ となるので，$I^{x'}$ のすべての要素 ρ' に対して body$(y'_{\rho'})$ の値は同じである．

ゆえに $I^{x'}$ のすべての要素 ρ' に対して，$y'_{\rho'}$ の代表元として同じ拡張項をとることができるので，これを $p_{y'}$ とする．このとき $I^{x'}$ のすべての要素 ρ' に対して，$y'_{\rho'} \prec p_{y'}$ だから，命題 11 より $I^{y'_{\rho'}} \subseteq I^{p_{y'}}$ が成り立つので，$I^{x'} \subseteq I^{p_{y'}}$ が成り立つ．ゆえに帰納法の仮定により $x' \prec p_{y'}$ が成り立つので，定義 15 より $x \prec y$ が成立する． □

6. 枝刈り式グラフ構造スタック法の計算量

ここでは，枝刈り式グラフ構造スタック法の時間計算量について述べる．枝刈り式グラフ構造スタック法を擬似プログラムの形式で記述したものをアルゴリズム 4 とアルゴリズム 5 に示す．アルゴリズム 4 はグラフ構造スタック法と共通の処理の部分を示し，アルゴリズム 5 は枝刈りに関する処理の部分を示す．

ここではループなどのネスト構造をインデントによって表現する．Parent 集合は，拡張項を添字とし値が拡張項の集合である配列 Parent として表現される．また配列 Reduce は n 番目の要素（Reduce$[n]$）がアルゴリズム 2 の Reduce$_{(n)}(E_{\alpha \cdot a})$ に相当する．さらに，ここでは Is_prec(x,y) において PrecTab という共通テーブルを用いることにより Is_prec(x,y) における重複した計算を抑止している．PrecTab の各要素の初期値は 0 であり，body$(x)=$ body$(y)=i$ を満たす x，$y \in E$ に対して，PrecTab$[i, \sharp\mathrm{input}(x), \sharp\mathrm{input}(y)]$ の値は，$x \prec y$ ならば 1，$x \prec y$ でなければ -1，不明であれば 0 である（文字列 α は固定と考えられるから，α の 1 文字目から j 文字目までの部分列を j で表すことができる）．

アルゴリズム 4 枝刈り式グラフ構造スタック法 a

```
Parse(α) /*α が語ならば True，そうでなければ False を返す*/
( 1) E_top:={e_0}; Parent[e_0]:=∅;
( 2) for i ∈ I_G
( 3)     for j from 0 to ♯α for k from 0 to ♯α
( 4)         PrecTab[i,j,k] := 0;
( 5) for j from 1 to ♯α
( 6)     E_top:=Goto(E_top,α_j);
( 7) E_top:=Goto(E_top,⊣);
( 8) return (E_top≠ ∅);
Goto(E_α, a) /* E_α と a から E_{α·a} を生成する */
( 9) Reduce_base:=∅;
(10) for x ∈ E_α
(11)     if x ∈ S0eitem(a) then /* 場合 (2) の処理 */
(12)         E_shift0:=Shift0(x,a);
             Reduce_base:=Reduce_base∪E_shift0;
(13)         for y ∈ E_shift0 /* 場合 (2) の Parent[y] */
```

```
(14)        Parent[y]:=Parent[x]; /* 補題 1 より */
(15)    if x ∈ S1eitem(a) then /* 場合 (3) の処理 */
(16)      E_shift1:=Shift1(x, a);
          Reduce_base:=Reduce_base∪E_shift1;
(17)      for y ∈ E_shift1 /* 場合 (3) の Parent[y] */
(18)        Parent[y]:=∅ ;
(19)        for z ∈ E_α
(20)          if z ∈ S1eitem(a)∧y ∈ Shift1(z, a) then
(21)            Parent[y]:=Parent[y]∪{z};
(22) return Rclosure(Reduce_base);
Rclosure(D) /* アルゴリズム 2 の場合 (4),(5) に対応 */
(23) n:=0; Reduce[0]:=D; Reduce[1]:=∅;
     E_return:=Reduce[0];
(24) loop /* このループ ((24)—(42)) の出口は (39) のみ */
(25)    for x ∈ Reduce[n]∩Redex
(26)      for x′ ∈ Parent[x]
(27)        if x′ ∈ R0eitem(x) then /* 場合 (4) の処理 */
(28)          E_reduce0:=Reduce0(x′, x);
              Reduce[n + 1]:=Reduce[n + 1]∪E_reduce0;
(29)          for y ∈ E_reduce0 /* 場合 (4) の Parent[y] */
(30)            Parent[y]:=Parent[x′]; /* 補題 2 より */
(31)        if x′ ∈ R1eitem(x) then/* 場合 (5) の処理 */
(32)          E_reduce1:=Reduce1(x′, x);
              Reduce[n + 1]:=Reduce[n + 1]∪E_reduce1;
(33)          for y ∈ E_reduce1 /* 場合 (5) の Parent[y]*/
(34)            Parent[y]:= ∅ ;
(35)            for z ∈ Reduce[n]∩Redex
(36)              for z′ ∈ Parent[z]∩R1eitem(z)
(37)                if y ∈ Reduce1(z′, z) then
(38)                  Parent[y]:=Parent[y]∪{z′};
(39)            Parent[y]:=Prune(Parent[y]);
(40)    if Reduce[n + 1]≠ ∅
(41)      then E_return:=E_return∪Reduce[n + 1];
              n:=n + 1; Reduce[n + 1]:=∅;
(42)      else return E_return; /*ループ ((24)-(42)) の出口*/
```

アルゴリズム 5 枝刈り式グラフ構造スタック法 b

```
Prune(D)
(43) D_result := ∅;
(44) for i ∈ I_G D_x[i]:=∅;
(45) for x ∈ D
(46)    D_x[body(x)]:=D_x[body(x)]∪{x};
(47) for i ∈ I_G
(48)    if | D_x[i] | = 1 then
(49)      D_result:=D_result∪ D_x[i] ;
(50)    else if | D_x[i] |> 1 then
(51)      p_x:=D_x[i] の任意の要素;
(52)      for x ∈ D_x[i] /* 代表元候補の選定 */
(53)        if Is_prec(p_x, x) then
(54)          p_x := x;
(55)      for x ∈ D_x[i] /* p_x が代表元でなければエラー */
(56)        if not Is_prec(x, p_x)
(57)          then エラー /* 枝刈り可能文法でない */
(58)      D_result:=D_result∪ {p_x};
(59) return D_result;
Is_prec(x, y) /* x ≺ y ならば True, そうでなければ False */
(60) i=body(x); l_x := ♯input(x); l_y := ♯input(y);
(61) if PrecTab[i, l_x, l_y]=1 then /* x ≺ y が成立 */
(62)    return True;
(63) if PrecTab[i, l_x, l_y]= −1 then /* x ≺ y が不成立 */
(64)    return False;
(65) flag:=True;
(66) for x′ ∈Parent[x] /* 定義 15 の条件が成り立つか確認 */
(67)    x_flag:=False;
(68)    for y′ ∈Parent[y]
(69)      x_flag:=x_flag∨(x′ = y′)
                ∨((body(x′)=body(y′))∧Is_prec(x′, y′))
(70)    flag:=flag∧x_flag;
(71) if flag then PrecTab[i, l_x, l_y]:=1;
(72)      else PrecTab[i, l_x, l_y]:= −1;
(73) return flag;
```

定義 22（縮退的） $D \subseteq J_G$ が，D の任意の要素 x, y に対して，$\mathrm{input}(x) = \mathrm{input}(y)$ が成り立つとき，D は縮退的（degenerated）であるという．

定義 14 と定義 22 より次の補題が成り立つ．

補題 9 $D \subseteq J_G$ に対して，D が縮退的であれば D は分離的である．

補題 10 $x, x' \in J_G$，$a \in V_T$ に対して，$\mathrm{Shift0}(x, a)$，$\mathrm{Shift0}(x, a)$，$\mathrm{Reduce0}(x', x)$，$\mathrm{Reduce1}(x', x)$ は縮退的であり，n に関しては定数時間（正確には $O(|I_G|)$）で求めることができる．

証明：y をこれら 4 つの集合の要素とする．定義 6 と定義 9 より $\mathrm{body}(y)$ は $\mathrm{body}(x)$ と $\mathrm{body}(x')$ のみに依存する．拡張項の body 部は有限集合 I_G の要素だから，$\mathrm{body}(x)$ と $\mathrm{body}(x')$ の値に対する 4 つの集合の要素の body 部は事前に求めることができる．つまりこれらの集合の body 部は解析時には n に関しては定数時間で求めることができる．

また定義 6 と定義 9 からこれらの 4 つの集合の要素の input 部はすべて同じなので，これらの集合は縮退的である．そこで補題 5 よりこれらの集合の要素

数は $|I_G|$ 以下である．ゆえにこれらの集合の要素を $O(|I_G|)$ で求めることができる． □

定義 14 から次の補題が成り立つ．

補題 11 $x \in J_G$ および $D, D' \subseteq J_G$ に対して，D と D' が分離的であれば所属判定（$x \in D$），和集合（$D \cup D'$），共通集合（$D \cap D'$）および代入（$D' := D$）は，定数時間（$O(|I_G|)$）で求めることができる．

補題 12，13，14 はアルゴリズム 4 と 5 で用いられる主な集合が分離的であることを示す．補題 11 から次の補題が成り立つ．

補題 12 D が縮退的ならば，アルゴリズム 4 の Rclosure(D) と Goto(D, a) の結果は縮退的である．

集合 $\{e_0\}$ は縮退的なので，補題 12 より次の補題が成り立つ．

補題 13 アルゴリズム 4 の $\mathrm{E_{top}}$ や E_α はつねに分離的である．

命題 5 より次の補題が成り立つ．

補題 14 $y \in \mathrm{E_{top}}$ に対して，Parent(y) は分離的である．

以下では関数 f の時間計算量を $||f||$ で表す．

命題 12 アルゴリズム 4 の時間計算量は，Prune の処理を除くと $O(n^2)$ である．

証明：アルゴリズム 4 の時間計算量は ||Parse|| と等しく，これは (3) のループにより $n \times$||Goto|| と表すことができる．ここで n は入力 α の長さを表す．補題 10，13，14 より，(10)，(13)，(17)，(19) のループの実行回数は定数 $|I_G|$ 以下であり，補題 10 と 11 より，このループ内の演算の時間計算量も定数である．このため，||Goto|| は ||Rclosure|| と等しい．同様に (24) の内部のループ (25)，(26)，(29)，(33)，(35)，(36) の実行回数およびそれらのループ内の Prune 以外の演算の時間計算量も定数である．また (24) の実行回数は各終端記号の解析時に還元動作が行われる回数に等しく，仮定より単一規則のみから構成されるループは存在しないので，(24) の実行回数は $O(n)$ である．ゆえに ||Rclosure|| は $n \times$||Prune|| に等しい．つまり枝刈り式グラフ構造スタック法の時間計算量は，Prune の処理を除くと $O(n^2)$ である． □

命題 13 長さ n の入力列を構文解析する場合の Prune の時間計算量（の総和）は $O(n^2)$ である．

証明：(35) の Reduce[n] と (36) の Parent[z] は分離的だから，Prune の引数である Parent[y] の要素数は $|I_G|^2$ 以下である．また D_x[i] $\subseteq$ Parent[y]（$i \in I_G$）より $|\mathrm{D\_x}[i]| \leq |I_G|^2$ となり，(52) と (55) のループの回数は n に依存しないので，||Prune|| = ||Is_prec|| が成り立つ．

(66) の Parent[x] と (68) の Parent[y] は分離的だから，PrecTab の値が判明している（0 でない）要素から (71)，(72) の PrecTab[i, l_x, l_y] を求める計算量は $O(1)$ である．ゆえに PrecTab(i, j, k)（$i \in I_G, 0 \leq j, k \leq n$）の値をすべて求める時間計算量は $O(n^2)$ なので，長さ n の入力列を構文解析する場合の Is_Prec の時間計算量の総和は $O(n^2)$ となり，題意は成り立つ． □

これらの命題から次の定理が成り立つ．

定理 5 枝刈り可能文法に対する枝刈り式グラフ構造スタック法の時間計算量は $O(n^2)$ である．

7. 強いあいまい性を持つ文法への適用例

本章では枝刈り式グラフ構造スタック法の適用例として，構文解析の時間計算量が Tomita ら[9] のアルゴリズムでは $O(n^{m+1})$，Kipps[2] のアルゴリズムでは $O(n^3)$ である文法 S_m（$m \geq 3$）に対して，枝刈り式グラフ構造スタック法では $O(n^2)$ で構文解析が行えることを示す．

例 19 次の構文規則を持つ文脈自由文法 $S3$ を考える．

(r^0) $S' : S \dashv$

(r^1) $S : S\ S\ S$　　(r^2) $S : S\ a$　　(r^3) $S : a$

$S3$ は文献 2) で $S3$ として定義されている文法と同じである．$S3$ の r^1 の右辺の S の個数が m 個（$m \geq 3$）である文脈自由文法を文献 2) と同じく S_m と呼ぶ．

例 20 例 19 の文法 $S3$ において，入力列が aa およびその部分列の場合の L^α を**図 21** に示す．図 21 では ⇑reduce と ⇓reduce の表示を省略している．

$S3$ において L^{a^n} から $L^{a^{n+1}}$ を生成する際の L^{a^n}

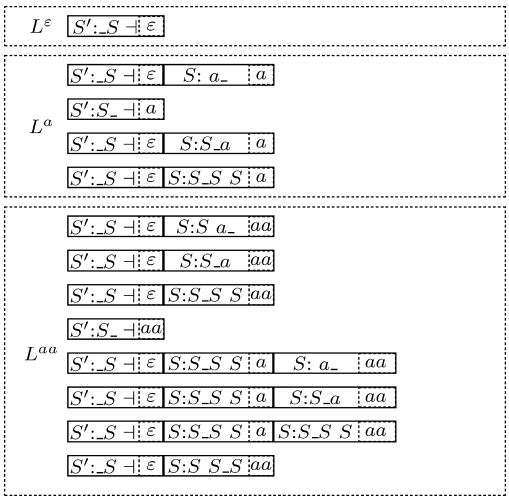

図 21 $S3$ における L^α の例
Fig. 21 Example of L^α in $S3$.

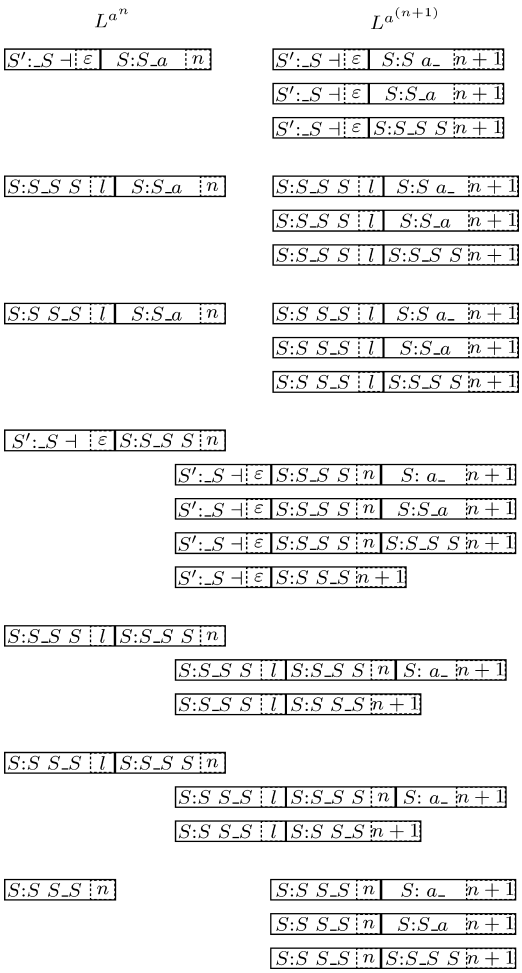

図 22 $S3$ における解析スタックのトップの変化例
Fig. 22 Stack top in $S3$.

に含まれる主な解析スタックのトップ要素の変化例を**図 22** に示す．図 22 では input 部の a^n を n で表す．図 22 から次の命題が成り立つ．

命題 14

$$\begin{aligned}
&\mathrm{Parent}(\langle S : S_a, n+1\rangle)\\
&= \{e_0, \langle S : S_S\ S, k\rangle, \langle S : S\ S_S, l\rangle\}\\
&\qquad\qquad (1 \le k \le n, 2 \le l \le n),\\
&\mathrm{Parent}(\langle S : S_S\ S, n+1\rangle)\\
&= \{e_0, \langle S : S_S\ S, k\rangle, \langle S : S\ S_S, l\rangle\}\\
&\qquad\qquad (1 \le k \le n, 2 \le l \le n),\\
&\mathrm{Parent}(\langle S : S\ S_S, n+1\rangle)\\
&= \{e_0, \langle S : S_S\ S, k\rangle, \langle S : S\ S_S, l\rangle\}\\
&\qquad\qquad (1 \le k \le n-1, 2 \le l \le n-1).
\end{aligned}$$

関係 $\prec$ の定義と上の命題から次の命題が成り立つ．

命題 15 $k < l$ として，

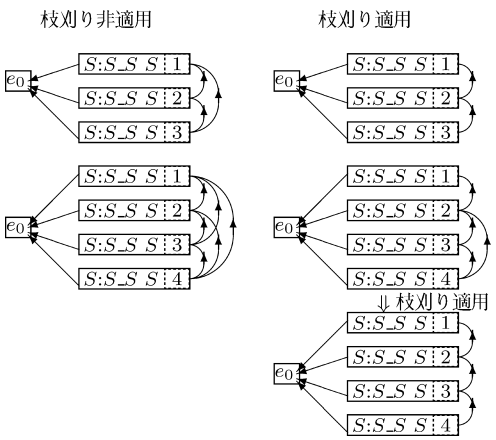

図 23 $S3$ における Parent 集合の関係
Fig. 23 Relation among Parent sets in $S3$.

$$\begin{aligned}
&\langle S : S_a, k\rangle \prec \langle S : S_a, l\rangle,\\
&\langle S : S_S\ S, k\rangle \prec \langle S : S_S\ S, l\rangle,\\
&\langle S : S\ S_S, k\rangle \prec \langle S : S\ S_S, l\rangle.
\end{aligned}$$

Prune の定義と上の命題から次の命題が成り立つ．

命題 16 $n \ge 2$ の場合は，

$$\begin{aligned}
&\mathrm{Prune}(\mathrm{Parent}(\langle S : S_a, n+1\rangle))\\
&= \{e_0, \langle S : S_S\ S, n\rangle, \langle S : S\ S_S, n\rangle\},\\
&\mathrm{Prune}(\mathrm{Parent}(\langle S : S_S\ S, n+1\rangle))\\
&= \{e_0, \langle S : S_S\ S, n\rangle, \langle S : S\ S_S, n\rangle\},\\
&\mathrm{Prune}(\mathrm{Parent}(\langle S : S\ S_S, n+1\rangle)\\
&= \{e_0, \langle S : S_S\ S, n-1\rangle, \langle S : S\ S_S, n-1\rangle\}.
\end{aligned}$$

命題 16 より S_3 は枝刈り可能文法なので，次の命題が成り立つ．

命題 17 $S3$ に対する，枝刈り式グラフ構造スタック法による構文解析の時間計算量は，$O(n^2)$ である．

命題 16 と同様に，S_m（$m > 3$）も枝刈り可能文法であるので，文法 S_m に対する枝刈り式グラフ構造スタック法による構文解析の時間計算量も $O(n^2)$ である．

例 21 $S3$ において，枝刈りを実施した場合と実施しない場合の $\langle S : S\_S\ S, k\rangle$（$1 \le k \le 4$）とその Parent 集合の関係を**図 23** に示す．図 23 では，拡張項 x, y が $y \in \mathrm{Parent}(x)$ を満たすとき x から y への矢印で表し，input 部の a^n を n と表す．以下では $S : S\_S\ S$ を i と表す．

上段は，aaa に関する処理が終わった時点での，$\langle i, k\rangle$（$1 \le k \le 3$）とその Parent 集合の関係を示す．枝刈りを実施しない場合は命題 14 より $\langle i, j\rangle \in \mathrm{Parent}(\langle i, k\rangle)$（$1 \le j < k \le 3$），$e_0 \in \mathrm{Parent}(\langle i, k\rangle)$ が成り立つ．枝刈りを実施する場合は命題 16 より $\langle i, k-1\rangle \in \mathrm{Parent}(\langle i, k\rangle)$（$1 < k \le 4$），$e_0 \in \mathrm{Parent}(\langle i, k\rangle)$ が

成り立つ．

中段は，$aaaa$ を読み込んだ時点での $\langle i,k\rangle$（$1 \leq k \leq 4$）とその Parent 集合の関係を示す．枝刈りを実施しない場合は $\langle i,j\rangle \in \mathrm{Parent}(\langle i,4\rangle)$（$1 \leq j < 4$），$e_0 \in \mathrm{Parent}(\langle i,4\rangle)$ が成り立つ．枝刈りを実施する場合は，$\langle i,j\rangle \in \mathrm{Parent}(\langle i,4\rangle)$（$j = 2, j = 3$），$e_0 \in \mathrm{Parent}(\langle i,4\rangle)$ が成り立つ．

下段は，$aaaa$ を読み込んだ後で枝刈りを実施した時点での $\langle i,k\rangle$（$1 \leq k \leq 4$）とその Parent 集合の関係を示す．このとき $\langle i,2\rangle \prec \langle i,3\rangle$ より，$\langle i,2\rangle$ は枝刈りにより $\mathrm{Parent}(\langle i,4\rangle)$ から削除されるので，$\langle i,3\rangle \in \mathrm{Parent}(\langle i,4\rangle)$，$e_0 \in \mathrm{Parent}$ が成り立つ．しかしこの場合も，$I^{\langle i,k\rangle}$（$1 \leq k \leq 4$）は，枝刈り前と枝刈り後で同じである．

8. 枝刈り式グラフ構造スタック法の適用範囲

本章では，グラフ構造スタック法や枝刈り式グラフ構造スタック法により構文解析が $O(n^2)$ で行える文法の例と行えない例を示す．命題 12 より，文法 G において，E の任意の要素 x に対して $\mathrm{Parent}(x)$ が分離的であれば，G の構文解析はグラフ構造スタック法により $O(n^2)$ で行える．また命題 12，命題 13 より G が枝刈り可能文法であれば，G の構文解析は枝刈り式グラフ構造スタック法により $O(n^2)$ で行える．さらに文法 G が E の任意の要素 x に対して $|\mathrm{Parent}(x)|$ が n に依存しない値以下であれば，命題 12 と同様に G の構文解析は $O(n^2)$ で行えることが証明できる．

逆に文法 G に対して，G の構文解析がグラフ構造スタック法や枝刈り式グラフ構造スタック法では $O(n^2)$ で行うことができない条件は，次の条件を満たす $y_1, \cdots, y_l$ を $\mathrm{Parent}(x)$ に含むような E の要素 x が存在することである．

(1) $l = O(n)$,

(2) y_j と y_k の間に $\prec$ 関係は存在しない（$1 \leq j, k \leq l$）．

例 22 G_{RL} を次の構文規則を持つ文脈自由文法とする．

$(r^0)S' : S \dashv \quad (r^1)S : a\ S \quad (r^2)S : X$

$(r^3)X : X\ a \quad (r^4)X : a$

以下では $S : a\_S$ を i，$X : X\_a$ を i' と表す．

$$\mathrm{Parent}(\langle i',n\rangle) = \{\langle i,k\rangle\}(2 \leq k \leq n)$$

が成り立つが，$\mathrm{Parent}(\langle i,1\rangle) = \{e_0\}$，$\mathrm{Parent}(\langle i,n\rangle) = \{\langle i,n-1\rangle\}$（$2 \leq n$）だから $\langle i,k\rangle$（$1 \leq k \leq n$）の間に $\prec$ 関係は存在しない．さらに $|\mathrm{Parent}(\langle i',n\rangle)| = O(n)$ となるので，G_{RL} の構文解析はグラフ構造スタック法や枝刈り式グラフ構造スタック法では $O(n^2)$ で行うことはできない．

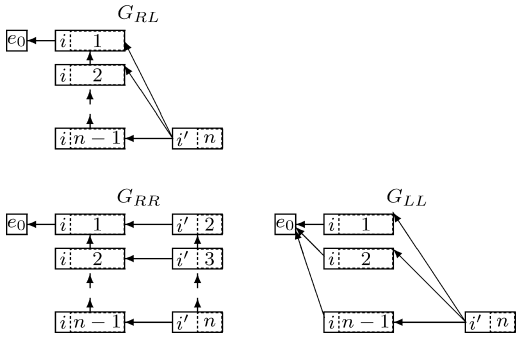

図 24 Parent 集合の関係
Fig. 24 Relation among Parent sets.

例 23 G_{RR} を次の構文規則を持つ文脈自由文法とする．

$(r^0)S' : S \dashv \quad (r^1)S : a\ S \quad (r^2)S : X$

$(r^3)X : a\ X \quad (r^4)X : a$

以下では $S : a\_S$ を i，$X : a\_X$ を i' と表す．$\mathrm{Parent}(\langle i,1\rangle) = \{e_0\}$，$\mathrm{Parent}(\langle i,n\rangle) = \{\langle i,n-1\rangle\}$，$\mathrm{Parent}(\langle i',n\rangle) = \{\langle i,n-1\rangle, \langle i',n-1\rangle\}$（$2 \leq n$）だから，$\mathrm{Parent}(\langle i,n\rangle)$ と $\mathrm{Parent}(\langle i',n\rangle)$ は分離的である．ゆえに G_{RR} の構文解析はグラフ構造スタック法により $O(n^2)$ で行うことができる．

例 24 G_{LL} を次の構文規則を持つ文脈自由文法とする．

$(r^0)S' : S \dashv \quad (r^1)S : S\ X \quad (r^2)S : X$

$(r^3)X : X\ a \quad (r^4)X : a$

以下では $S : S\_X$ を i，$X : X\_a$ を i' と表す．$\mathrm{Parent}(\langle i,n\rangle) = \{e_0\}$，$\mathrm{Parent}(\langle i',n\rangle) = \{\langle i,k\rangle\}$（$1 \leq k < n$）だから $\langle i,j\rangle \prec \langle i,k\rangle$（$1 \leq j < k < n$）が成り立つので，$G_{LL}$ は枝刈り可能文法である．ゆえに G_{LL} の構文解析はグラフ構造スタック法では $O(n^2)$ で行えないが，枝刈り式グラフ構造スタック法により $O(n^2)$ で行うことができる．

G_{RL}，G_{RR}，G_{LL} における $\langle i,j\rangle$，$\langle i',k\rangle$（$1 \leq j,k \leq n$）間の Parent 関係を**図 24** に示す．

例 22 から分かるように左再帰の構文規則と（左再帰でない）右再帰の構文規則をともに含む文脈自由文法は枝刈りが適用できない可能性がある．また例 24 から分かるように，左再帰の構文規則のみを含む文脈自由文法は枝刈りが適用できる．

9. 結　　語

本論文ではグラフ構造スタックに対して枝刈りを行うことにより構文解析を効率化するアルゴリズムを述べた．構文解析アルゴリズムを効率化する手法とし

て従来は表やメモ関数の利用が知られているが，それらはすでに行った処理を重複して行わないことにより効率化するのに対し，枝刈りは今後行う処理を重複して行わないことにより効率化するものである．このことが可能となるのは，本アルゴリズムが用いるスタックにおいて，入力列を受理する場合は，プッシュした要素を最終的にはすべてポップするため，過去にプッシュした要素は，過去の処理を表すだけでなく，ポップされるまでの今後の処理も表す，と見なせるためである．

本アルゴリズムは効率的であるが，すべての解析木を求めることはできない．ただし 8 章で述べたように，計算量は $O(n^2)$ のままで，任意の定数個の解析木を求めるように本アルゴリズムを改良することができる．

今後に残された課題としては，

(1) 枝刈り解析法の実装と評価，

(2) 枝刈り可能文法の詳細な特徴付け，

などがある．

参 考 文 献

1) Earley, J.: An efficient context free parsing algorithm, *Comm. ACM*, Vol.13, No.2, pp.94–102 (1970).
2) Kipps, J.R.: GLR parsing in time $O(n^3)$, in 8), Chap. 4, pp.43–59 (1991).
3) Leemakers, R.: A recursive ascent Earley parser, *Info. Process. Lett.*, Vol.41, pp.87–91 (1992).
4) Nederhof, M.J.: Generalized left-corner parsing, *Proc. 6th Conf. on European Chapter of the ACL*, pp.305–314 (1993).
5) Schabes, Y.: Polynomial time and space shift-reduce parsing of arbitrary context-free grammars, *Proc. 29th ACL*, pp.106–113 (1991).
6) Shann, P.: Experiments with GLR and chart parsing, in 8), Chap. 2, pp.17–34 (1991).
7) Sippu, S. and Soisalon-Soininen, E.: *Parsing Theory*, Vol.I, II, Springer, Berlin (1990).
8) Tomita, M. (Ed.): *Generalized LR Parsing*, Kluwer Academic Publishers (1991).
9) Tomita, M. and Ng, S.K.: The generalized LR parsing algorithm, in 8), Chap. 1, pp.1–16 (1991).

(平成 18 年 5 月 1 日受付)
(平成 18 年 8 月 10 日採録)

森本 真一（正会員）

1956 年生．1981 年東京大学大学院理学系研究科情報科学専攻修士課程修了．同年日本電気株式会社入社．1999 年（株）NEC 航空宇宙システム出向．言語処理系，ソフトウェア開発環境，品質管理システムの研究開発に従事．日本ソフトウェア科学会会員．

石畑 清（正会員）

1952 年生．理学博士．現在明治大学理工学部情報科学科教授．プログラミング言語，プログラミング方法論等に興味を持つ．著書に『アルゴリズムとデータ構造』（岩波書店）ほか．

疋田 輝雄（正会員）

1947 年生．1978 年理学博士（東京大学）．1989 年より明治大学理工学部情報科学科教授．計算理論，ネットワークコンピューティング等に興味を持つ．著書に『コンパイラの理論と実現』（共著，共立出版）ほか．