枝刈り式構文解析法の効率化と自然言語解析への応用

森 本 真 一[†1]

本論文では，文脈自由文法に対する構文解析アルゴリズムである枝刈り式グラフ構造スタック方式（以下では枝刈り法と呼ぶ）を自然言語解析に適用した場合の課題と対応および枝刈り法の効率化について述べる．枝刈り法は強いあいまい性を持つ文脈自由文法の多くに適用でき，時間計算量は $O(n^2)$ であるがすべての解析木を求めることはできない．文脈自由文法の応用では強いあいまい性に対応するために可能な解析木の一部のみを対象とする場合がある．そのような応用の例として自然言語の解析を取り上げ，枝刈り法の適用を検討する．さらに枝刈り法の時間計算量が $O(n)$ となる文法の十分条件を述べる．本論文ではまず枝刈り法の定義を述べる．次に枝刈り法の自然言語解析への応用について述べる．まず自然言語解析において対象とする解析木を選択する既存の手法を示し，枝刈り法と既存の手法を比較し枝刈り法を適用する場合の課題を述べる．最後にこの課題に対応する枝刈り法の変更を示す．続いて枝刈り法により $O(n)$ で解析できる文法の十分条件を述べる．さらに自然言語解析で用いられる文法がこの条件を満たすことを示す．

Improvement of Prune Parsing and Its Application to Natural Language Processing

Shin-ichi Morimoto[†1]

In this paper we state the application of the Pruned GraphStructured Stack Parsing algorithm (hereafter we abbreviate it as Pruned GSSP) to the natural language processing and its improvement. Pruned GSSP can parse many of context-free languages with high ambiguity in $O(n^2)$ but we can not obtain all parse trees. In some application of context-free grammars, possible parse trees are chosen to decrease ambiguity. As an example of such application, we take natural language processing and discuss application of Pruned GSSP. We also state the improvemnet of time complexity of Pruned GSSP to $O(n)$. In this paper, we first define Pruned GSSP and then discuss its application to the natural language processing. In this discussion, we first state current approaches to choose parse trees, compare Pruned GSSP with these approaches and state modifications of Pruned GSSP to apply natural language processing. Next we state the sufficient conditions of context-free grammars that can be parsed by pruned GSSP in $O(n)$ and show the grammar for natural language processing satisfies these conditions.

1. はじめに

文脈自由文法はプログラミング言語の構造だけでなく自然言語の構造[1),3),5)] の記述にも用いられる．しかし自然言語の構造を記述する文脈自由文法は一般に強いあいまい性を含む．このようなあいまい性を抑える方法としては，構文規則を変更する方法[5)] と確率文脈自由文法を用いる方法[1)] に大別できる．

構文規則を変更する方法は，強いあいまい性の原因となっている特定の構文規則を同じ入力列を受理し，あいまい性が弱い構文規則に置き換えるものである．たとえば文献 4) で述べたように，“S: S S” と “S: a” という構文規則は強いあいまい性を持つ．そこでこれらを同じ入力列を受理し，あいまい性がない “S: a S” と “S: a” という構文規則におきかえる．この方法には，あいまい性の原因となる構文規則を特定する必要があることおよび解析のために本来の構文規則（構造）を変更してしまうという問題点がある．

確率文脈自由文法は各構文規則や LR テーブルのシフト還元の各アクションに対して，確率値を割り当て，それに基づき各解析木の確率値を算出し確率値の高い解析木のみを対象とするものである．この方法では，対象としたい解析木の確率値が高くなるように個々の構文規則やアクションの確率値を定めなければならないという問題点がある．

枝刈り法[4)] は，あいまい性の強い文脈自由文法の多くに適用できる構文解析アルゴリズムであり，時間計

†1 株式会社 NEC 航空宇宙システム
NEC Aerospace Systems, Ltd.

算量は $O(n^2)$ であるがすべての解析木を求めることはできない．しかし文脈自由文法の適用分野では強いあいまい性に対応するために可能な解析木の一部のみを取り出せば十分である場合がある．本論文では，そのような分野として自然言語解析を取り上げ，解析木の一部のみを取り出す手法としての枝刈り法の適用性をこの分野での既存の方式と比較して検討する．

自然言語解析ではあいまい性を減少させるためるに構文規則の変更を行っているが，既存手法では枝刈り法よりも多くの解析木が排除される．そこで既存手法と同様な結果が得られるように枝刈り法の変更を行う．

枝刈り法では，1) 枝刈り以外の処理，2) 枝刈り処理のいずれにも計算量が $O(n^2)$ となる処理が含まれるため計算量は $O(n^2)$ となる．一方，自然言語解析における構文規則の変更では強いあいまい性を持つ構文規則をあいまい性のない右再帰の構文規則に変更するため，変更後の文脈自由文法の構文解析の計算量は $O(n)$ になる．そこで，枝刈り法でも，1) 枝刈り以外の処理，2) 枝刈り処理に含まれる計算量が $O(n^2)$ の処理を $O(n)$ で行うための処理の変更およびそのための文脈自由文法の十分条件を述べる．さらに変更される前の文法がこの条件を満たすことを述べる．

2 章で用語の定義を，3 章で枝刈り法のもとになるグラフ構造スタックを用いたアルゴリズムを述べ，4 章で枝刈り法の定義を述べる．2 章〜4 章は文献 4) の内容と基本的に同じであるが，本論文でも必要なので記述する．5 章では枝刈り法の自然言語解析への応用を述べる．6 章では枝刈り法の計算量が $O(n)$ となる文脈自由文法の十分条件を述べ，5 章の自然言語の文法がその条件を満たすことを示す．7 章では自然言語解析における既存の手法と枝刈り法の比較を述べる．

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（V の要素）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．

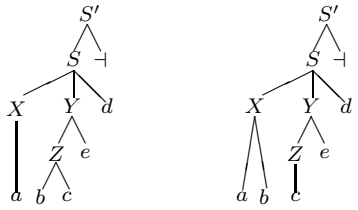


図 1 $G1$ の解析木
Fig. 1 Parse trees for $G1$.

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 $A : B$，$B : A$ 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．

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

$(r^1)\ S : X\ Y\ d$　$(r^2)\ X : a$　$(r^3)\ X : a\ b$

$(r^4)\ Y : Z\ e$　$(r^5)\ Z : c$　$(r^6)\ Z : b\ c$

語 $abced$ に対する $G1$ のすべての解析木を**図 1** 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．

定義 3（拡張項） $t \in I_G$，$\alpha \in {V_T}^*$ に対して，$\langle t, \alpha\rangle$ を拡張項といい，拡張項全体の集合を J_G で表す．$x = \langle t, \alpha\rangle \in J_G$ に対して，t を x の body 部といい $\mathrm{body}(x)$ で表し，α を x の 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．

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．

定義 5（シフト可能集合） $a \in V_T$ に対して，

$\mathrm{S0eitem}(a) = \{\langle\langle r, n\rangle, \alpha\rangle \mid r_{n+1} = a,$

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．

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，

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

と定義する．

定義 7（Redex）

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

と定義する．

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

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

と定義する．

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

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

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

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

と定義する．

定義 5，6，7，8，9 の例を付録に示す．

3. グラフ構造スタック法

アルゴリズム 1（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 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 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
$$\mathrm{Reduce}_{(0)}(E_{\alpha\cdot a}) = \bigcup_{x\in E_\alpha\cap \mathrm{S0eitem}(a)} \mathrm{Shift0}(x,a) \cup \bigcup_{x\in E_\alpha\cap \mathrm{S1eitem}(a)} \mathrm{Shift1}(x,a)$$
$$\mathrm{Reduce}_{(n+1)}(E_{\alpha\cdot a}) = \bigcup_{x\in \mathrm{Reduce}_{(n)}(E_{\alpha\cdot a})\cap \mathrm{Redex}} \Big(\bigcup_{x'\in \mathrm{Parent}(x)\cap \mathrm{R0eitem}(x)} \mathrm{Reduce0}(x',x) \cup \bigcup_{x'\in \mathrm{Parent}(x)\cap \mathrm{R1eitem}(x)} \mathrm{Reduce1}(x',x) \Big)$$
と定義する．

(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 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 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．

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

$w \in \mathrm{Parent}(y)$ であっても解析木上で w が y の親頂点であるとは限らないことに注意せよ．

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

例 2 例 1 の $G1$ において入力列が $abce$ の場合の E_α を**図 2** に示す．図 2 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．

アルゴリズム 1 の場合 (2) で定義される Parent(y) に関して次の補題が成り立つ．証明は文献 4) を参照．

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

定義 11（E^α） $\alpha \in V_T{}^*$ に対して，$E^\alpha \subseteq J_G{}^*$ を次のように定義する．
$$E^\alpha = \{\sigma \mid \mathrm{Parent}(\sigma_1) = \emptyset, \sigma_j \in \mathrm{Parent}(\sigma_{j+1})(1 \le j < \sharp\sigma), \sigma_\sharp \in E_\alpha\}.$$

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

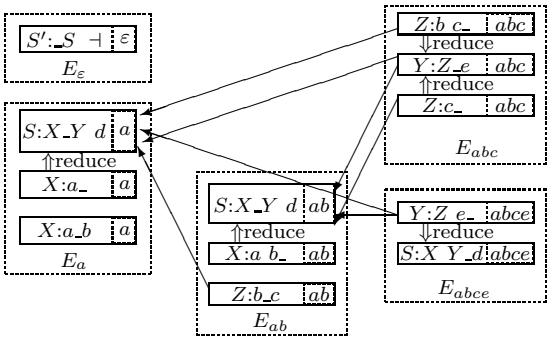


図 2 $G1$ における E_α の例
Fig. 2 Example of E_α in G1.

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

4. 枝刈り式グラフ構造スタック法

有限集合 H の大きさ（要素数）を $|H|$ で表す．

定義 12（Succ） 項 $i = \langle r, n\rangle \in I_G$ に対して
$$\mathrm{Succ}(i) = \langle r, n+1\rangle$$
と定義する．

定義 13（分離的） $D(\subseteq E)$ 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 1 の場合 (4) の $\mathrm{Parent}(x)$ が分離的な場合は次の補題が成り立つ．証明は文献 4) を参照．

補題 2 x，x'，y がアルゴリズム 1 の場合 (4) の条件を満たす場合に，$\mathrm{Parent}(x)$ が分離的であれば，$\mathrm{Parent}(y) = \mathrm{Parent}(x')$ が成り立つ．

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 I_G の内部に 1 対 1 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 1 $D \subseteq {J_G}^*$ に対して，D が分離的であれば $|D| \le |I_G|$．よって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．

E_α はすべての要素の input 部が同一（α）なので分離的である．さらに E_α のすべての要素の Parent 集合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．しかし分離的でない Parent 集合も存在する．

例 4 図 2 の $\mathrm{Parent}(\langle Y : Z\_e, abc\rangle)$ は $\langle S : X\_Y\, d, a\rangle$ と $\langle S : X\_Y\, d, ab\rangle$ を含むので分離的でない．

定義 14（I^α，E^x，I^x） $\alpha \in {V_T}^*$，$x \in E_\alpha$ に対して，$I^\alpha \subseteq {I_G}^*$，$E^x \subseteq E^\alpha$（定義 11 参照），$I^x \subseteq I^\alpha$ を次のように定義する．
$$\begin{aligned}I^\alpha &= \{\rho \mid \exists \sigma \in E^\alpha \text{ ただし } \sharp\rho = \sharp\sigma,\\ &\qquad \rho_j = \mathrm{body}(\sigma_j)\quad (1 \le j \le \sharp\rho)\}.\\ E^x &= \{\sigma \mid \sigma \in E^\alpha, \sigma_\sharp = x\}.\\ I^x &= \{\rho \mid \rho \in I^\alpha, \rho_\sharp = x\}.\end{aligned}$$

例 5 例 1 の $G1$ において，${I^{abc}} = \{\rho^1, \rho^2, \rho^4\}$．ただし，
$$\begin{aligned}\rho^1 &= ({\rho^1}_1, {\rho^1}_2, {\rho^1}_3) = (i_0, S : X\_Y d, Z : bc\_).\\ \rho^2 &= ({\rho^2}_1, {\rho^2}_2, {\rho^2}_3) = (i_0, S : X\_Y d, Y : Z\_e),\\ \rho^4 &= ({\rho^4}_1, {\rho^4}_2, {\rho^4}_3) = (i_0, S : X\_Y d, Z : c\_).\end{aligned}$$
I^x の定義より，次の補題が成り立つ．

補題 3 $x \in E$ に対して，
$$I^x = \textstyle\bigcup_{y \in \mathrm{Parent}(x)} \{\rho \cdot \mathrm{body}(x) \mid \rho \in I^y\}.$$

文献 4) で示したように，x，$y \in \mathrm{Parent}(z)$ に対して，$I^x \subseteq I^y$ ならば y を残して x を枝刈りしてもよい．そこで x，$y \in \mathrm{Parent}(z)$ に対して，$I^x \subseteq I^y$ であることを表す二項関係を以下に定義する．

定義 15（$\prec$） $\mathrm{body}(x) = \mathrm{body}(y)$ を満たす x，$y \in E$ 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．

$\prec$ に関して次の命題が成り立つ．証明は付録に示す．

命題 2 $x \prec y$ ならば $I^x \subseteq I^y$．

命題 2 より $x \prec y$ ならば x を枝刈りしてもよい．

定義 16（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 p_x が存在するとき，D は代表可能という．

定義 17（Prune） $D \subseteq E$ を代表可能な集合とする．各 $x \in D$ に対して x の代表元から 1 つを選んで，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．

定義 17 から次の命題が得られる．

命題 3 $D \subseteq E$ に対して，$\mathrm{Prune}(D)$ が存在すれば $\mathrm{Prune}(D)$ は分離的である．

定義 18（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．

アルゴリズム 2（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 1 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（枝刈り法）と呼ぶ.

枝刈り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 3 とアルゴリズム 4 に示す．アルゴリズム 3 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 4 は枝刈り処理の部分を示す.

ここではループなどのネスト構造をインデントにより表現する．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．配列 Reduce は n 番目の要素（Reduce[n]）がアルゴリズム 1 の Reduce$_{(n)}$($E_{\alpha\cdot a}$) に相当する．さらに，Is_prec(x, y) において PrecTab という共通テーブルを用いることで 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[$i, \sharp$input(x), $\sharp$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.

本アルゴリズムや後述のアルゴリズムでは出力は真偽値であるが，適用規則をポインタでリンクし，認識に成功したときにトレースバックにより構文木を構成するようアルゴリズムを容易に拡張することができる.

アルゴリズム 3 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 1 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 (42) 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 4 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;
```

5. 自然言語処理への適用

5.1 自然言語処理における解析木の削除

本章では文献 5) に基づき既存の自然言語解析においてあいまい性を抑える方法について述べる．自然言語解析において広範囲な対象を解析するためには大規模な文法が必要になる．たとえば，文献 5) で対象としている日本語文法は 1694 個の構文規則，249 個の非終端記号，600 個の終端記号から構成される．このような大規模な文法を人手で作成することは困難であり，文法を大規模な構造付きコーパスから生成されることが行われている．しかし，このようにして生成された文法は強いあいまい性を持つため，そのままで構文解析を行うと膨大な数の解析結果（解析木）が生成される．

このようなあいまい性を抑える方法の 1 つとして確率文脈自由文法を用いる方法がある．これは各構文規則や LR テーブルのシフト還元の各アクションに対して，確率値を割り当て，それに基づき各解析木の確率値を算出し確率値の高い解析木のみを対象とするものである．しかしこの方法では，対象としたい解析木の確率値が高くなるように個々の構文規則やアクションの確率値を定めなければならないという問題点がある．しかし文脈自由文法の構文規則は解析木の親子関係しか規定せず，それ以外の周辺文脈情報を持たないため，それぞれの文脈での適切な解析木を指定するような確率値を定めることは困難である．

また実際の自然言語では特定の構文規則が大部分のあいまい性の原因となっているが，確率文脈自由文法ではそのような場合の対応が困難である．

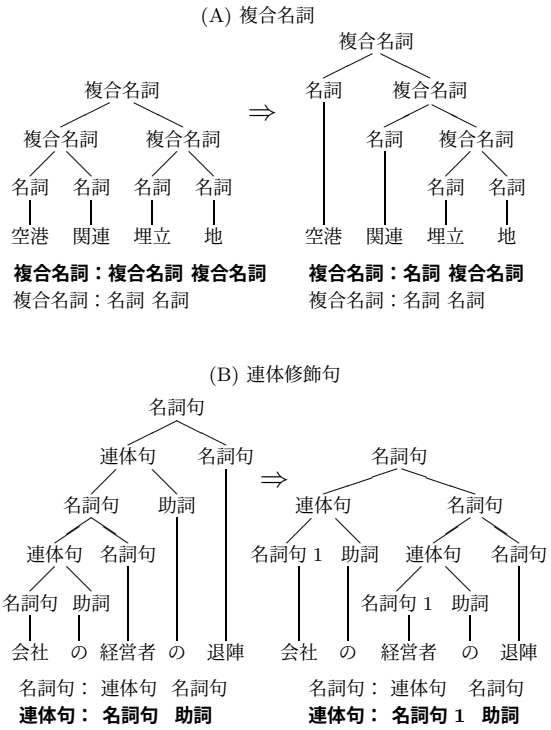


図 3 あいまい性の強い構文規則の変更
Fig. 3 Modification of productions with high ambiguity.

文献 5) では，強いあいまい性の原因となっている構文規則を変更することによりあいまい性を減少させている．日本語文法において強いあいまい性の主な原因となっているのは，(A) 複合名詞 と (B) 連体修飾句 に関する構文規則である．(A) の構文規則は，“複合名詞：複合名詞 複合名詞” および “複合名詞：名詞 名詞” であり，(B) の構文規則は，“名詞句：連体修飾句 名詞句” および “連体修飾句：名詞句 助詞” である．文献 5) では，(A) の “複合名詞：複合名詞 複合名詞” という構文規則を “複合名詞：名詞 複合名詞” に，(B) の “連体修飾句：名詞句 助詞” という構文規則を “連体修飾句：名詞句 1 助詞” に変更している．名詞句 1 は名詞句のうち連体修飾句以外を導出する構文記号である．

(A)，(B) の変更前および変更後の構文規則とそれに基づく解析木を**図 3** に示す．図 3 は，文献 5) の図と基本的に同じである．

5.2 自然言語処理におけるあいまい性の強い構造

文献 5) ではコーパスに基づき作成した文法のあいまい性について分析しているが，そこではあいまい性の原因となる構文構造として

(J1) 複合名詞
(J2) 連体修飾句

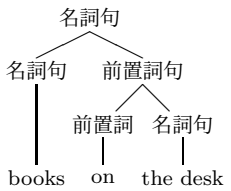


図 4 前置詞句の例
Fig. 4 Rules and parse trees of PP attachment.

(J3) 並列名詞句

の 3 つをあげている．(J1) と (J2) については 5.1 節で述べた．(J3) は “名詞句：名詞句 並列助詞 名詞句” という構文規則を持つ．これらの構文規則はいずれも文献 4) で述べた S_m という文脈自由文法の $m = 2$ の場合（この場合の文法を S_2 と記す）と本質的に同じである．S_m は文献 2) で導入された文法で強いあいまい性を持ち，文献 7)，文献 2)，文献 4) の各アルゴリズムによる計算量が最悪（文献 7) では $O(n^{p+1})$（ただし p は右辺が最も長い構文規則の長さ），文献 2) では $O(n^3)$，文献 4) では $O(n^2)$）になる．

また文献 3) では，被験者が対話システムと行った対話を分析しているが，そこでの強いあいまい性の原因となる構造として

(E1) Noun-Noun Modification

(E2) Conjunction

(E3) Prepositional Phrase Attachment

の 3 つをあげている．(E1) は (J1) に相当し，(E2) は (J3) に相当する．(E3) の構文規則と解析木を**図 4** に示す．これらの構造も基本的には S_2（文献 3) では，grammar AA と呼んでいる）と同じである．

このように S_2 は多くの言語で強いあいまい性の原因となっている．

5.3 自然言語処理にむけた枝刈り法の課題

S_2 は，“S：S S” および “S：a” という構文規則を持つ．S_2 において入力が “aaa” の場合のスタックと対応する解析木を**図 5** に，グラフ構造化スタックを**図 6** に示す．簡単のため図 5 と図 6 では，拡張項の input 部は入力列ではなく入力列の長さを示している．

枝刈り法では，枝刈りにより図 6 の×印の辺を削除する．これは図 5 において×印の解析木とスタックを削除することに相当する．一方，文献 5) における構文規則の変更は S_2 の構文規則を右再帰の構文規則に変更することに相当するので，図 5 において○印の解析木とスタックのみを残すことに相当する．つまり枝刈り法が文献 5) における構文規則の変更と同じ効果

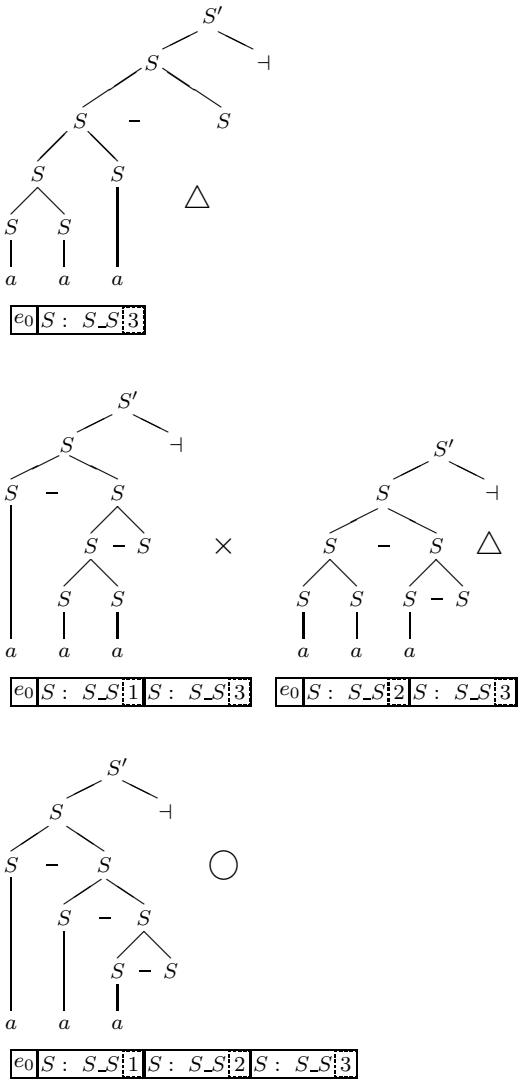


図 5 “aaa” に対する S_2 の解析木とスタック
Fig. 5 Parse trees and stack of S_2 for “aaa”.

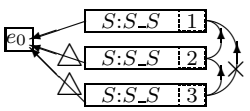


図 6 “aaa” に対する S_2 のグラフ構造スタック
Fig. 6 A Graph Structured Stack of S_2 for “aaa”.

を得るためには，図 5 において△印の解析木とスタックも削除するために図 6 の△印の辺も削除するように枝刈り法を変更する必要がある．

5.4 自然言語処理にむけた枝刈り法の拡張

文献 4) で述べたように，$x \in \mathrm{Parent}(z)$ に対して x を枝刈りできるのは $I^x \subseteq I^y$ を満たす $y \in \mathrm{Parent}(z)$ が存在する場合である．たとえば図 13 で $i = S : S\_S$

とすると，$I^{\langle i,1\rangle} = \{i_0 \cdot i\} \subseteq \{i_0 \cdot i, i_0 \cdot i^2\} = I^{\langle i,2\rangle}$ となるので，図 6 の×印の辺を削除できる．このように枝刈りが可能なのは，図 5 の中段の 2 つのスタックは，トップ要素である $\langle i,3\rangle$ が n 文字目の入力後に還元されると，いずれも同じ内容（$e_0 \cdot \langle \mathrm{Succ}(i), n\rangle$）になるためである．

これに対して $I^{e_0} = \{i_0\}$，$I^{\langle i,2\rangle} = \{i_0 \cdot i, i_0 \cdot i^2\}$ なので，$I^{e_0} \not\subseteq I^{\langle i,2\rangle}$ となり図 6 の $\langle i,3\rangle$ から e_0 への（△印の）辺は削除できない．なぜなら図 5 の各スタックのトップ要素である $\langle i,3\rangle$ の還元後のスタックの要素の body 部は，上段のスタックでは $\mathrm{Succ}(i_0)$，中段の右のスタックでは $i_0 \cdot \mathrm{Succ}(i)$，下段のスタックでは $i_0 \cdot i \cdot \mathrm{Succ}(i)$ となり，すべて異なるためである．

しかし $\mathrm{Succ}(i)$ はさらに還元可能であり，中段右のスタックや下段のスタックも還元を続けると上段のスタックと同じ $\mathrm{Succ}(i_0)$ になる．つまり○印のスタックと△印のスタックは構造は異なるがトップ要素の S によるシフト後に還元を続けると同じスタックになる．ゆえに，この場合は○印のスタックと△印のスタックは（スタックの構造が異なっていても）どちらかを削除してよい．そこでこのような場合は△印のスタックを削除するように枝刈り法を変更する．

以下では簡単のためここでの対象となる場合（還元が続けて発生する場合）をアルゴリズム 1 の場合 (4)（Reduce0 による還元）に限る．還元が続けて発生する場合としてこのほかに，アルゴリズム 1 の場合 (5)（Reduce1 による還元）により単一則が還元される場合も考えられるが仮定より単一則から構成されるループは存在しないので，この場合の還元は固定（構文規則の個数以下）回しか連続して発生しないため，以下の内容が基本的に適用できる．

定義 19（⇒） x，$y \in E$ に対して，
$\mathrm{body}(x) = \langle r^x, n^x\rangle$，$\mathrm{body}(y) = \langle r^y, n^y\rangle$ とするとき
・$y \in \mathrm{Parent}(x)$
・${r^x}_0 = {r^y}_{n^y+1}$
・$n_y = \sharp r^y - 1$
が成り立つとき，$x \Rightarrow y$ と定義する．

$x \Rightarrow y$ の場合は，x が還元されると続いて y も還元される．

例 6 図 6 のグラフ構造スタックにおいて，
$x = \langle S : S\_S, 2\rangle$，$y = \langle S : S\_S, 1\rangle$ とすると，
$r^x = r^y = S : S\ S$，$n_x = n_y = 1$ であり
・$y = \langle S : S\_S, 1\rangle \in \mathrm{Parent}(x = \langle S : S\_S, 2\rangle)$
・${r^x}_0 = {r^y}_{n^y+1} = S$
・$n_y = 1 = \sharp r^y - 1$
が成り立つので，定義 19 の条件は満たされ $x \Rightarrow y$ が成り立つ．

定義 20（Prune_N） $x \in E$，
$P \subseteq \mathrm{Parent}(x)$ に対して
$\mathrm{P1} = \{y | y \in P, x \Rightarrow y\}$，
$\mathrm{P2} = \{z | z \in P, z \notin \mathrm{P1}(x)\}$ とすると
P1，P2 が
$\forall y \in \mathrm{P1}, \forall z \in \mathrm{P2}$ に対して $z \in \mathrm{Parent}(y)$
を満たすとき，$\mathrm{Prune}_N(P, x) = \mathrm{P1}$ とする．

例 7 図 6 において例 6 と同じく
$x = \langle S : S\_S, 2\rangle$，$y = \langle S : S\_S, 1\rangle$ とすると，
$P = \mathrm{Parent}(x)$ に対し $\mathrm{P1} = \{y\}$，$\mathrm{P2} = \{e_0\}$ となり
P1，P2 は定義 20 の条件を満たすので，
$\mathrm{Prune}_N(\mathrm{Parent}(x), x) = \mathrm{P1} = \{y\}$ となる．これは Prune_N により図 6 のグラフ構造スタックにおける x から e_0 への（△印の）辺が削除されたことに相当する．

この変更に対応するために，アルゴリズム 2 の「$\mathrm{Parent}(y) = \mathrm{Prune}(\mathrm{P}'(y))$」を「$\mathrm{Parent}(y) = \mathrm{Prune}_N(\mathrm{Prune}(\mathrm{P}'(y)), y)$」に置き換える．

アルゴリズム 5 自然言語解析対応の枝刈り法
枝刈り可能文法に対して，アルゴリズム 1 の場合 (5) の定義中の 「$\mathrm{Parent}(y) = \mathrm{P}'(y)$」を
「$\mathrm{Parent}(y) = \mathrm{Prune}_N(\mathrm{Prune}(\mathrm{P}'(y)), y)$」に
置き換えたアルゴリズムを自然言語解析対応の枝刈り法と呼ぶ．

アルゴリズム 2 からアルゴリズム 5 へ変更すると，Parent 集合に対して Prune_L によりさらに枝刈りが行われるため Parent 集合が縮小する．このため変更前の Parent 集合に対しては定義 20 の条件が成り立っていた x，P が変更後では成り立たなくなる場合が存在する．このような場合に対応するために，Prune_L により削除された Parent 集合の要素（P2 の要素）を表すテーブルを別途設けて，変更後のアルゴリズムでも，ある要素が変更前の Parent 集合に含まれていたかを判定できるようにする必要がある．

例 8 例 6 と同じ x, y に対して，
$w = \langle S : S\_S, 3\rangle$ とすると，
$\mathrm{Parent}(w) = \{e_0, x\}$，$\mathrm{Parent}(x) = \{e_0, y\}$ なので，$\mathrm{Parent}(w)$，w に関して定義 20 の条件が成り立つ．しかし $\mathrm{Prune}_N(\mathrm{Parent}(x), x) = \{y\}$ なので，変更後のアルゴリズムにより $\mathrm{Prune}_N(\mathrm{Parent}(x), x)$ を $\mathrm{Parent}(x)$ とすると $\mathrm{Parent}(w)$，w に関して定義 20 の条件が成り立たなくなる．

5.5 自然言語処理にむけた枝刈り法の定義

5.4 節で述べた変更を行った枝刈り法は，アルゴリズム 3 の

(39) Parent[y]:=Prune(Parent[y]);
を
(39) Parent[y]:=Prune_N((Prune(Parent[y]),y);
に置き換え，アルゴリズム 4 にアルゴリズム 6 で示す Prune_N を追加したものである．

アルゴリズム 6 における is_fold(x, y) は，x，y が $x \Rightarrow y$ ならば true，そうでなければ false を返す関数である．この is_fold(x, y) は，x と y の body 部のみに依存するので入力列には依存しない．また was_Parent_elem(z, x) は例 8 のような場合に対応する関数であり，z が x の Parent 集合の（P2 の）要素であるかを示す．

アルゴリズム 6 自然言語解析対応の枝刈り法

```
Prune_N(P, x)
(N01) P1 := ∅; P2 := ∅;
(N02) for y ∈ P
(N03)    if is_fold(x, y)
(N04)      then P1 := P1 ∪ {y};
(N05)      else P2 := P2 ∪ {y};
(N06) is_pruneN := true; N_result := P1;
(N07) for y ∈ P1 ; /* 定義 20 の条件が成り立つか確認 */
(N08)    for z ∈ P2
(N09)      if not (z ∈ Parent(y) or was_Parent_elem(z, y))
(N10)        then is_pruneN :=false;/*定義 20 の条件不成立*/
(N11) if is_pruneN
(N12)    then P1 := P1 ∪ {y};
(N13)      for z ∈ P2
(N14)        was_Parent_elem(z, x) := true;
(N15)    else N_result := P;
(N16) return N_result;
```

6. 枝刈り法の効率化

5.2 節で述べたように図 3 における (A)，(B) の構造の構文規則は S_2 の構文規則と本質的に同じである．このため (A) や (B) の構造を枝刈り法により構文解析を行う場合の時間計算量は $O(n^2)$ である．しかし文献 5) では (A)，(B) の構造の構文規則を右再帰の構文規則に変更している．右再帰の構文規則はあいまい性がないため変更後の構造に対する構文解析の時間計算量は $O(n)$ である．そこで本章では S_2 に対して $O(n)$ で構文解析が行えるように枝刈り法の効率化を行う．

枝刈り方式は，(1) 枝刈り以外の処理，(2) 枝刈り処理 の 2 つの処理から構成され，文献 4) で述べたように両方の計算量はともに $O(n^2)$ である．本章ではこれらの計算量を $O(n)$ とする方式およびそれが可能な文脈自由文法の条件を述べる．なお以下では input 部の a^n を n と表す．

6.1 枝刈り以外の処理の効率化

枝刈り法が対象としているあいまい性のある文法では n 文字目の読み込み処理において長さ $O(n)$ の拡張項の列に対して $O(n)$ 回の還元処理と $O(n)$ 回の新たな拡張項の生成を行う場合がある．このような場合は読み込み処理ごとに $O(n)$ 回の処理を行う必要があるので，長さ n の入力列の処理全体で $O(n^2)$ の処理が必要になる．

定義 21（$\Longrightarrow$） x，$y \in E$ に対して，
$\mathrm{body}(x) = \langle r^x, n^x \rangle$，$\mathrm{body}(y) = \langle r^y, n^y \rangle$ とするとき
- $x \Rightarrow y$
- ${r^y}_{n^y+1} = {r^x}_1$

が成り立つとき，$x \Longrightarrow y$ と定義する．

$\Longrightarrow$ の最初の条件は x が還元されると y も還元されることを示し，次の条件は x が還元されるとアルゴリズム 1 の場合 (5) により body 部が $\langle r^x, 1 \rangle$ であり y を Parent 集合に含む拡張項が生成されることを示す．

例 9 次の構文規則を持つ文脈自由文法 S_3 を考える．

(r^0) $S' : S \dashv$
(r^1) $S : S\ S\ S$ (r^2) $S : S\ a$ (r^3) $S : a$

$x = \langle S : SS\_S, n \rangle$，$y = \langle S : SS\_S, n-1 \rangle$ とすると $r^x = r^y = S : SSS$，$n^x = n^y = 2$ より $x \Longrightarrow y$ が成り立つ．

S_3 において n 文字目の読み込み処理において $O(n)$ 回の還元処理と $O(n)$ 回の拡張項の生成を行う状況を**図 7** に示す．図 7 の上段は n 文字目の読み込み処理において $\langle S : a\_, n \rangle$ という拡張項が生成された状況を示す．

$\langle S : a\_, n \rangle$ が還元されるとアルゴリズム 1 の場合 (4) より，$\langle S : S\ S\_S, n-1 \rangle$ は $\langle S : S\ S\ S\_, n \rangle$ に置き換えられる．またアルゴリズム 1 の場合 (5) より，

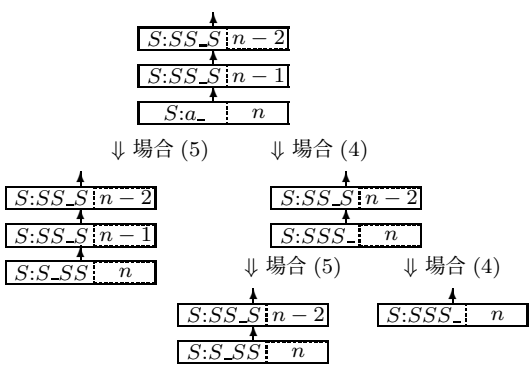


図 7 $O(n)$ の処理が必要な還元列
Fig. 7 Reduction sequence that needs $O(n)$ process.

$\langle S : S\_S\ S, n\rangle$ がプッシュされる．前者ではさらに，$\langle S : S\ S\ S\_, n\rangle$ の還元により $\langle S : S\ S\ S\_, n-1\rangle$ がポップされ $\langle S : S\ S\_S, n-2\rangle$ が $\langle S : S\ S\ S\_, n\rangle$ に置き換えられる場合と $\langle S : S\ S\ S\_, n-1\rangle$ が $\langle S : S\_S\ S, n\rangle$ に置き換えられる場合が存在する．
これらの繰返しにより，この場合は $O(n)$ 回の還元処理と $O(n)$ 回の拡張項の生成が行われる．

定義 21 において生成される拡大項の body 部が x の body 部と同じ場合は，生成される拡大項の Parent 集合に関する処理は x の Parent 集合に関する処理と同じになる．そこで生成される拡大項の Parent 集合に関する処理にすでに行った処理結果である x の Parent 集合を利用することにより，重複した処理を省き枝刈り法による構文解析の計算量を削減することを考える．

命題 4 x，x'，$x'' \in E$ が
・$\mathrm{body}(x) = \mathrm{body}(x') = \langle r, 1\rangle$
・$x \Longrightarrow x', x' \Longrightarrow x''$
を満たせば，
$\mathrm{Parent}(x) \supseteq \{x'\} \cup \mathrm{Parent}(x')$ が成り立つ．
証明：仮定より，$x = \langle\langle r, 1\rangle, l\rangle$，$x' = \langle\langle r, 1\rangle, l'\rangle$，$x'' = \langle i, l''\rangle$ とおける．

l 文字目の処理で x' を Parent 集合に含み r_0 を左辺に持つ構文規則に対応する拡張項が還元されると，アルゴリズム 1 の場合 (5) により x が生成されるから $x' \in \mathrm{Parent}(x)$ となる．また場合 (4) により x' が還元されると x' はポップされ x'' を Parent 集合に含む x が生成される．$\mathrm{body}(x) = \mathrm{body}(x')$ だから，これ以降の処理は Parent 集合の生成と枝刈りも含めて l' 文字目の処理で x' が生成された処理と同じである．ゆえに $\mathrm{Parent}(x) \supseteq \mathrm{Parent}(x')$ が成り立つ． □

命題 4 の条件を満たす場合は x に対する処理のうち x' に対する処理と重複する部分は x' の処理結果を利用できるので $O(n)$ 回の還元処理や拡張項の生成（Parent 集合の生成や枝刈りなどの処理を含む）を行わなくてよい．

例 10 S_2 において $r = S : SS$ とし，$x = \langle\langle r, 1\rangle, l\rangle$，$x' = \langle\langle r, 1\rangle, l-1\rangle$，$x'' = \langle\langle r, 1\rangle, l-2\rangle$ とすると x，x'，x'' は命題 4 の条件を満たす．この状況を**図 8** に示す．図 8 では l' は $l-1$ を，l'' は $l-2$ を表す．図 8 の右側の下段の 2 つのグラフ構造スタックは input 部を除けば左側の下段の 2 つのグラフ構造スタックと同じである．ゆえに右側のグラフ構造スタックのこれ以降の処理も inpit 部以外は左側のグラフ構造スタックのこれ以降の処理と同じになる．

命題 4 により Rclosure の処理を効率化した枝刈り法はアルゴリズム 3 の Rclosure をアルゴリズム 7 に示す Rclosure で置き換えたものである．$\mathrm{IsProp4}(x, x')$ は，x，x' が命題 4 の x，x' に関する条件を満たす場合にのみ True を返す関数である．

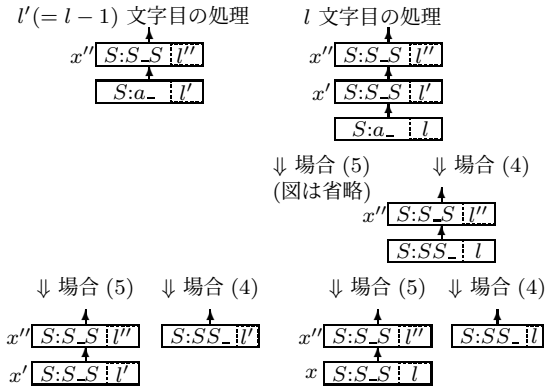


図 8 S_2 における還元時の重複処理
Fig. 8 Duplicate process in reduceaction in S_2.

アルゴリズム 7 効率化された枝刈り法

```
Rclosure(D) /* アルゴリズム 1 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 (42) のみ */
(L0)   L_flag=True;
(25)   for x ∈ Reduce[n]∩Redex
(26)     for x' ∈ Parent[x]
(L1)       L_flag=IsProp4(x,x') and L_flag;
(L2)       for x'' ∈ Parent[x']∩Redex
(L3)         L_flag=IsLngArrw(x',x'') and L_flag;
(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'};
(L4)                 if L_flag then
(L5)                   Parent[y]:=Parent[y]∪Parent[z'];
(39)           Parent[y]:=Prune(Parent[y]);
(L40)    if Reduce[n + 1]≠ ∅ and Not L_flag
(41)     then E_return:=E_return∪Reduce[n + 1];
```

n:=$n+1$; Reduce[$n+1$]:=$\emptyset$;

(42) else return E_return; /*ループ ((24)-(42)) の出口*/

6.2 枝刈り処理の効率化

4 章で述べたように現状の枝刈り法（アルゴリズム 4）では枝刈り可能かの判定を PrecTab というテーブルを用いて行っている．$\mathrm{body}(x) = \mathrm{body}(y) = i$ を満たす $x,\ y \in E$ に対して，$\mathrm{PrecTab}[i, \sharp\mathrm{input}(x), \sharp\mathrm{input}(y)]$ 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．n 文字目の処理時の PrecTab の要素数は n^2 であり，アルゴリズム 4 の $\mathrm{Is\_prec}(x, y)$ の計算量は PrecTab の必要な要素の値が判明している（0 でない）場合は $O(1)$ である．ゆえに長さ n の入力列の処理における枝刈り可能性の判定（$\prec$ 関係の判定）の計算量は $O(n^2)$ である．

命題 5 枝刈り可能文法に対して

(1) 各拡張項の枝刈りの対象となる拡張項（枝刈り前の Parent 集合の要素）の数は $O(1)$,

(2) 拡張項間の枝刈り可能かの判定（$\prec$ 関係の判定）の計算量は $O(1)$,

が成り立つ場合は，アルゴリズム 4（枝刈り処理）の計算量は $O(n)$ である．

証明：(1) より各拡張項の枝刈り前の Parent 集合の要素数は $O(1)$ であり，(2) より Parent 集合の要素間の $\prec$ 関係の判定の計算量は $O(1)$ である．ゆえに Parent 集合の各要素の代表元は $O(1)$ で決定できる．つまり各入力文字の処理時の枝刈り処理の計算量は $O(1)$ なのでアルゴリズム 4 の計算量は $O(n)$ である． □

例 11 S_2 で枝刈りを実施した場合と実施しない場合の $\langle S : S\_S, k\rangle$ $(1 \le k \le 4)$ とその Parent 集合の関係を**図 9** に示す．図 9 では x，y が $y \in \mathrm{Parent}(x)$ を満たすとき x から y への矢印で表す．以下では $S : S\_S$ を i と表す．

図 9 から分かるように枝刈り法を適用する場合は

(1) $\langle i, n\rangle$ の枝刈りの対象は $\langle i, n-1\rangle$ と $\langle i, n-2\rangle$,

(2) $\langle i, n-1\rangle$ と $\langle i, n-2\rangle$ の $\prec$ 関係は $\langle i, n-2\rangle$ と $\langle i, n-3\rangle$ の $\prec$ 関係と同じであり，n 文字目の読み込み処理の時点で $\mathrm{PrecTab}[i, n-3, n-2] = 1$ なので，$\langle i, n-1\rangle$，$\langle i, n-2\rangle$ の $\prec$ 関係判定の計算量は $O(1)$,

が成り立つ．ゆえに命題 5 の 2 つの条件が成り立つので，この場合の枝刈り処理の計算量は $O(n)$ である．

6.3 S_2 に対する構文解析

例 10 より S_2 は命題 4 の条件を満たすのでアルゴリズム 7 により枝刈り処理以外の処理を $O(n)$ で行うことができる．また例 11 より S_2 は命題 5 の条件を満たすのでアルゴリズム 4 により枝刈り処理を $O(n)$ で行うことができる．

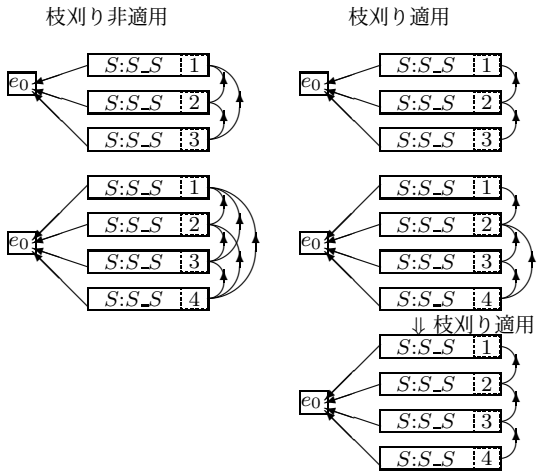


図 9 S_2 における Parent 集合の関係
Fig. 9 Relation among parent sets in S_2.

つまりアルゴリズム 7 とアルゴリズム 4 により，S_2 は構文規則を変更することなく構文解析を $O(n)$ で行うことができる．

7. 既存手法との比較

既存の自然言語解析では強いあいまい性に対応するために，確率文脈自由文法を利用する場合がある．確率文脈自由文法は各構文規則や LR テーブルのシフト還元の各アクションに対して確率値を割り当て，その確率値に基づきあいまい性を減少させる．これに対して枝刈り法は拡張項間に $\prec$ 関係を定義し，この関係に基づきあいまい性を減少させる．つまりあいまい性を減少させるために確率文脈自由文法では確率値という全順序を用いるが，枝刈り法では $\prec$ 関係という前順序（反射則と推移則は成り立つが，反対称則が成り立たない関係）を用いる．また確率文脈自由文法における全順序は文法の構造ではなく構文解析の対象の内容により決定される．このため確率文脈自由文法ではあらかじめ構造の判明している文を解析（パラメータ学習）し，その構造が選択されるように確率値を与えている．これに対して枝刈り法では文法の構造に基づき前順序が決定される．このため枝刈り法ではパラメータ学習を行う必要がない．

また既存の自然言語解析では強いあいまい性に対応するためにあいまい性の原因となる構文規則を変更してしまう場合がある．たとえばあいまい性をなくすために S_2 の構造を持つ構文規則を右再帰の構文規則に変更する．これに対して枝刈り法では構文規則ではなく構文解析の方法を変更することによりあいまい性を

減少させる．たとえば S_2 の構造を変更せずに右再帰の構文規則に変更した場合と同等の時間計算量で構文解析を行うことができる．

8. ま と め

本論文では枝刈り法の自然言語解析への適用と効率化について述べた．

枝刈り法は多くの言語で強いあいまい性の原因である構造 (S_2) を $O(n)$ で構文解析できる．

7 章で述べたように枝刈り法は二項関係に基づきあいまい性を減少させるという点では確率文脈自由文法と同じであるが，枝刈り法では文法の構造に基づき二項関係を定義し確率文脈自由文法では解析対象の内容に基づき二項関係を定義する．

このように枝刈り法は既存の手法とは異なる考え方による手法なので既存の手法と組み合わせることによりさらにあいまい性を減少させることができると思われる．

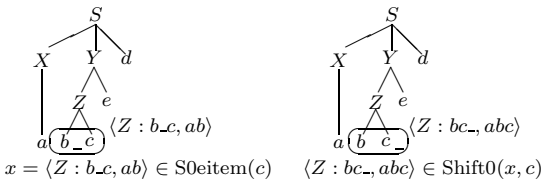
参 考 文 献

1) Inui, K., Sornlertlamvanich, V., Tanaka, H. and Tokunaga, T.: Probabilistic GLR Parsing: A New Formalization and Its Impact on Parsing Performance, *Journal of Natural Language Processing*, No.5, pp.33–52 (1998).
2) Kipps, J.R.: GLR parsing in time $O(n^3)$, *Generalized LR Parsing*, pp.43–59, Kluwer Academic Publishers (1991).
3) Martin, W.A., Church, W.K. and Patil, R.S.: Preliminary Analysis of a Breadth-First Parsing Algorithm: Theoretical and Experimental Results, *Natural Language Parsing Systems*, Bolc, L. (ed.), pp.267–328, Springer-Verlag (1987).
4) 森本真一，石畑 清，疋田輝雄：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，情報処理学会論文誌：プログラミング，Vol.47, No.SIG 16 (PRO 31), pp.66–85 (2006).
5) 野呂智哉，橋本泰一，徳永健伸，田中穂積：大規模日本語文法の開発，自然言語処理，Vol.12, No.1, pp.3–31 (2005).
6) Tomita, M. (Ed.): *Generalized LR Parsing*, Kluwer Academic Publishers (1991).
7) Tomita, M. and Ng, S.K.: The generalized LR parsing algorithm, *Generalized LR Parsing*, pp.1–16, Kluwer Academic Publishers (1991).

付 録

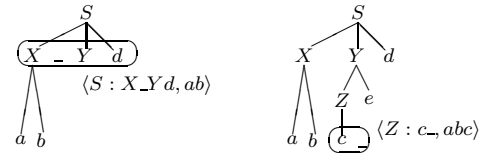
A.1 定義の例

2 章の定義 5，6 の例を**図 10**，**図 11** に，定義 7，8，9 の例を**図 12**，**図 13** に示す．



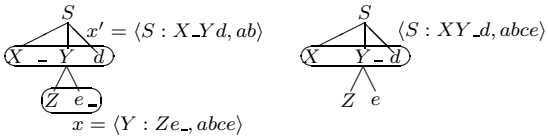
$x = \langle Z : b\_c, ab\rangle \in \mathrm{S0eitem}(c)$　$\langle Z : bc\_, abc\rangle \in \mathrm{Shift0}(x, c)$

図 10 S0eitem(c) と Shift0(x, c) の例
Fig. 10 Example of S0eitem(c) and Shift0(x, c).



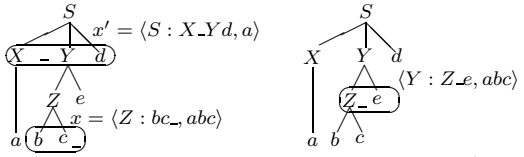
$x = \langle S : X\_Yd, ab\rangle \in \mathrm{S1eitem}(c)$　$\langle Z : c\_, abc\rangle \in \mathrm{Shift1}(x, c)$

図 11 S1eitem(c) と Shift1(x, c) の例
Fig. 11 Example of S1eitem(c) and Shift1(x, c).



$x' \in \mathrm{R0eitem}(x)$　$\langle S : XY\_d, abce\rangle \in \mathrm{Reduce0}(x', x)$

図 12 R0eitem(x) と Reduce0(x', x) の例
Fig. 12 Example of R0eitem(x) and Reduce0(x', x).



$x' \in \mathrm{R1eitem}(x)$　$\langle Y : Z\_e, abc\rangle \in \mathrm{Reduce1}(x', x)$

図 13 R1eitem(x) と Reduce1(x', x) の例
Fig. 13 Example of R1eitem(x) and Reduce1(x', x).

A.2 枝刈り法の考え方

例 1 の $G1$ の入力 “abce” に対する構文解析の状況を**図 14** に示す．$x = \langle Y : Z\_e, abc\rangle$，$x_1 = \langle S : X\_Yd,\ a\rangle$，$x_2 = \langle S : X\_Yd,\ ab\rangle$，$x' = \langle S : XY\_d,\ abce\rangle$ とする．図 14 では，左の列の 3 つが図 1 の左の解析木に対応する解析過程を，右の列の 3 つが図 1 の右の解析木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 2 で Parent(x) が x_1，x_2 を含むことに対応する．

図 14 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

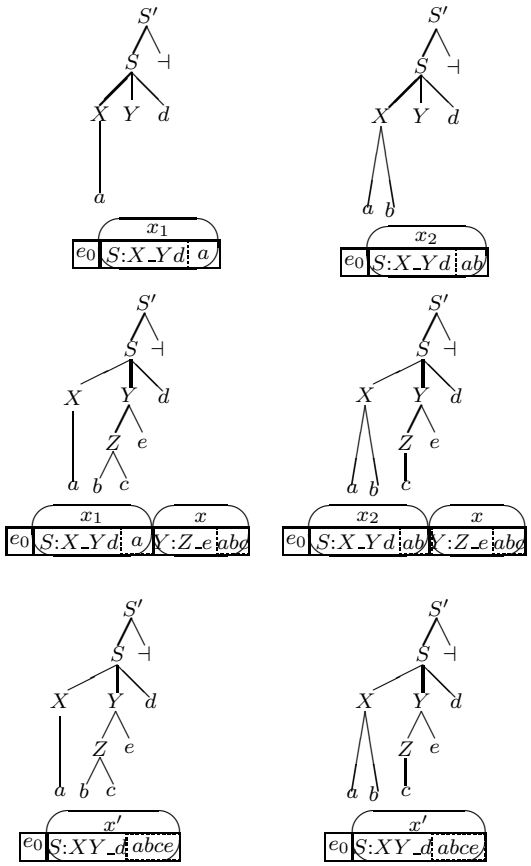


図 14 $G1$ における入力 $abce$ の解析
Fig. 14 Parse process for $abce$.

元すると x_1 や x_2 はいずれも x' にシフトするから，2 つの解析木のスタックは同じになる．

図 14 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．これら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，トップ要素の還元後はそれらは同じ要素（x'）にシフトするので 2 つのスタックは同じスタック（下段のスタック）になる．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場合は，それらのうち 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できる．

A.3 命題 2 の証明

命題 2 $x \prec y$ ならば $I^x \subseteq I^y$.

証明：$\mathrm{L}(x,y)=\max(\sharp\mathrm{input}(x),\sharp\mathrm{input}(y))$ とするとき，$\mathrm{L}(x,y)$ に関する帰納法により証明する．本アルゴリズムの対象となる文脈自由文法は ε 規則を持たないからアルゴリズム 1 より次の (A)，(B) が成り立つ．

(A) $x \in \mathrm{Parent}(y)$ ならば $\sharp\mathrm{input}(x) < \sharp\mathrm{input}(y)$.

(B) $\sharp\mathrm{input}(x)=0$ ならば $x = e_0$.

定義 15 より，$x \prec y, \sharp\mathrm{input}(y) = 0$ ならば $\sharp\mathrm{input}(x) = 0$ が成り立つ．また $\sharp\mathrm{input}(x) = 0$ の場合はアルゴリズム 1 の場合 (1) より $\mathrm{Parent}(x) = \emptyset$ となるので題意は成り立つ．ゆえに $\sharp\mathrm{input}(x) \geq 1, \sharp\mathrm{input}(y) \geq 1$ の場合を考えればよい．

(1) $\mathrm{L}(x,y) = 1$ の場合．この場合は $\sharp\mathrm{input}(x)=\sharp\mathrm{input}(y)=1$ となるので，(A)，(B) より $\mathrm{Parent}(x) = \mathrm{Parent}(y) = \{e_0\}$ となり，題意は成り立つ．

(2) $\mathrm{L}(x,y) \leq k$ の場合に題意が成立するとして，$\mathrm{L}(x,y) = k+1$ 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，$I^{x'} = I^{y'}$ が成り立つ．$x' \prec y'$ の場合は，$x' \in \mathrm{Parent}(x)$，$y' \in \mathrm{Parent}(y)$ だから，(A) より $\sharp\mathrm{input}(x') \leq k$，$\sharp\mathrm{input}(y') \leq k$ となり，$\mathrm{L}(x',y') \leq k$ が成り立つので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 3 より，すべての $\rho \in I^x$ に対して，$\rho = \rho' \cdot \mathrm{body}(x)$ を満たす $x' \in \mathrm{Parent}(x)$，$\rho' \in I^{x'}$ 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． □

(平成 19 年 5 月 7 日受付)
(平成 19 年 8 月 20 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．