Research Reports on Mathematical and Computing Sciences

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

Shin-ichi Morimoto

Jan 2007, C–240

Department of
Mathematical and
Computing Sciences
Tokyo Institute of Technology

SERIES C: Computer Science

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

Shin-ichi Morimoto
NEC Aerospace Systems, Ltd.
s-morimoto@ah.jp.nec.com

In this report, I introduce an efficient parsing algorithm even for highly ambiguous context-free grammars, which is intended to be applied to natural language processing. 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 sets of 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. The new technique here is to prune the 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 report, I first define the algorithm and show its correctness, and then I analyze its time complexity. Then I show that by this 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 and application for natural language processing.

1 Introduction

Many parsing algorithms for general context-free grammars have been presented, in which there are mainly two approaches: the tabular approach [1], [4], and the stack approach [3], [5], [11]. The tabular approach uses an $n \times n$ matrix for an input $a_1 \ldots a_n$ where the (i, j) element of this matrix corresponds to a set of items that represents recognition of the part of the input $a_{i+1} \ldots a_j$. On the other hand, the stack approach uses stacks whose elements are states [3], [11] or items [5]. In order to parse a grammar with ambiguity, these algorithms use a set of stacks, or a graph structured stack that is a directed graph whose nodes correspond to stack elements and edges correspond to pointers to previous elements, which are called parent pointers. An example of stacks whose top elements are D and a corresponding graph structured stack is shown in the left part of Fig. 1.

As for efficiency, the tabular approaches are generally better than stack approaches, since time complexity of the former is $O(n^3)$, and the latter is $O(n^{p+1})$ in the original implementation [11], where p is the longest length of the right-hand side of productions. There are modified versions of the stack approaches as in [3] and [5], whose theoretical time complexities are $O(n^3)$. However, the implementation in [5] is presented only in its outline, and according to [3], its implementation is not practical. For practical situations the stack approaches seem to be better than the tabular approaches [8].

The algorithm presented here, named Pruned Graph-Structured Stack Parsing, is a modification of the algorithm by Nederhof [5]. In this algorithm when a new node is generated during parsing, we prune some of parent pointers of the new generated node if possible. The graph structured stack after pruning and its corresponding stack set is shown in the right part of Fig. 1. The graph structured stack in the right part is same as the one in the left part except that the

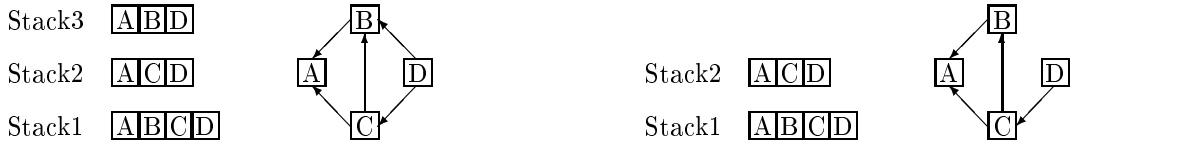


Figure 1: Stack sets and corresponding graph structured stacks

edge from node D to node B is pruned. Pruning this edge corresponds to delete stack 3 from the stack set in the left part.

In this algorithm stack elements are extended items that are pairs of an item and an input string, whereas elements in stack are items and tokens in the algorithm by Nederhof [5]. By adding more information of input strings to items, we can distinguish items generated from different input strings. Thus, we can prune parent pointers in a reduce action, so that the parent pointer set for every stack element does not have more than one element whose items are same and inputs are different. By this pruning we can reduce the time complexity to $O(n^2)$, since the size of a set of parent pointers becomes less than constant (= the size of the set of items), although we cannot obtain all possible parse trees.

In Sect. 2, I introduce definitions and notations used in this report. I state an algorithm based on a set of parse stacks in Sect. 3, and modify it to a graph-structured stack in Sect. 4. The methods in Sect. 3 and 4 seem rather known, and the chief contribution here is to formulate them in formal descriptions. In Sect. 5, by introducing the idea of pruning into the algorithm in Sect. 4, I state the improved algorithm, and its correctness. In Sect. 6 I analyze the time complexity of this algorithm. In Sect. 7 as an example of this algorithm, I show that grammars called S_m ($m \geq 3$) whose time complexity of parsing is $O(n^{m+1})$ in [11] and $O(n^3)$ in [3], can be parsed in $O(n^2)$ by this algorithm. In Sect. 8 some study is made on the range of grammars that can be applicable for this algorithm. In Sect. 9, application of this algorithm for natural language processing is stated. Finally in conclusion, I state characteristics of pruning approach compared with the current approaches to improve the complexity. In Appendix I prove that the parsing algorithms in Sect. 3 and Sect. 4 corresponds exactly to each other which are rather routine.

2 Definitions and Notations

Definition 1 (Context-free grammar). *A context-free grammar (CFG) G is a 4-tuple (V_N, V_T, S, P) where V_N is a finite set of nonterminal symbols, V_T a finite set of terminal symbols, $S \in V_N$ a start symbol, and P is a set of productions. For $r \in P$, $\sharp r$ denotes the length of the right part of a production r, r_j the j-th element in the right part of r. Thus, a production r is expressed as $r_0 : r_1 r_2 \ldots r_{\sharp r}$.*

As usual we augment a CFG with the new start symbol S' and a production $S' : S \dashv$. Throughout this report we assume that all CFGs are ε-free.

Example 1. Let $G1$ be a context-free grammar with the following productions.

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

Parse trees of $G1$ are shown in Fig. 2. For $G1$, ambiguity exists for an input "abced".

This grammar will be used as a primary example throughout this report.

Definition 2 (item). *For $r \in P$ and a nonnegative integer n ($0 \leq n \leq \sharp r$), an item is a pair $\langle r, n \rangle$, and the set of all items is denoted by I_G. For a production r of the form $r_0 : r_1 r_2 \ldots r_{\sharp r}$,*

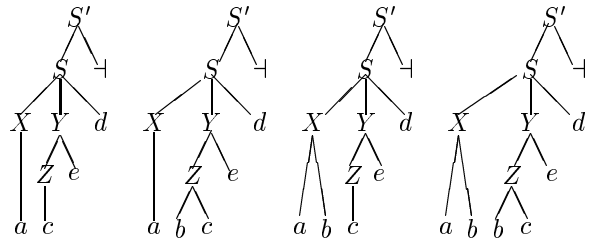


Figure 2: Parse trees for $G1$

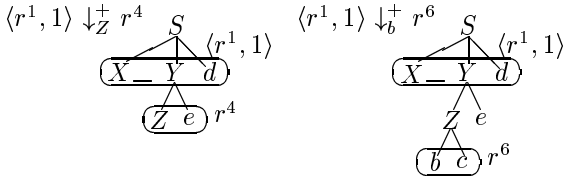


Figure 3: Examples of $\downarrow_w^+$

an item $\langle r, n\rangle$ can be expressed as $r_0 : r_1 r_2 \ldots r_n \_ r_{n+1} \ldots r_{\sharp r}$. In particular, the item $\langle S' : \_\ S\ \dashv\rangle$ is denoted by i_0.

Definition 3 (extended item or e-item). *For $t \in I_G$ and $\alpha \in {V_T}^*$, an extended item (or an e-item) is a pair $\langle t, \alpha\rangle$, and the set of all e-items are denoted by J_G. For $x = \langle t, \alpha\rangle \in J_G$, we define* $\mathrm{body}(x)$ *and* $\mathrm{input}(x)$ *by* $\mathrm{body}(x) = t$ *and* $\mathrm{input}(x) = \alpha$*, respectively. In particular, the e-item $\langle i_0, \varepsilon\rangle$ is denoted by e_0.*

For a sequence η, $\sharp\eta$ denotes the length of η, η_j the j-th element of η, and $\eta_\sharp$ the last element of η.

Definition 4 ($\downarrow$, $\downarrow_w^+$). *For $\langle r, n\rangle \in I_G$, $r' \in P$ and $w \in V_N \cup V_T$, we define $\langle r, n\rangle \downarrow r'$ if $r_{n+1} = {r'}_0$. We define $\langle r, n\rangle \downarrow_w^+ r'$ if there exist $r^{m_1}, \ldots, r^{m_k} = r' \in P$ $(k \geq 1)$ that satisfy the following conditions:*

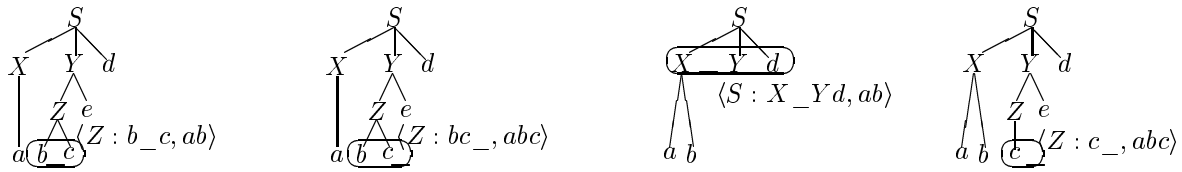
1. $\langle r, n\rangle \downarrow r^{m_1}$,
2. $\langle r^{m_j}, 0\rangle \downarrow r^{m_{j+1}}$, ${r^{m_j}}_1 \neq w$ $(1 \leq j < k)$,
3. ${r^{m_k}}_1 = w$.

Example 2. Let $G1$ be a CFG in example 1. We have $\langle r^1, 1\rangle \downarrow r^4$, $\langle r^1, 1\rangle \downarrow_Z^+ r^4$, and $\langle r^1, 1\rangle \downarrow_b^+ r^6$. We recall that $r^1 = S : XYd$, $r^4 = Y : Ze$, and $r^6 = Z : bc$. The situation for $\langle r^1, 1\rangle \downarrow_Z^+ r^4$ and $\langle r^1, 1\rangle \downarrow_b^+ r^6$ are shown in Fig. 3.

Definition 5 ($\mathrm{S0eitem}_G, \mathrm{S1eitem}_G$). *For a CFG G, $a \in V_T$, we define*
$\mathrm{S0eitem}_G(a) = \{\langle\langle r, n\rangle, \alpha\rangle \mid r_{n+1} = a$ for some $r \in P, \alpha \in V_T^*, 0 \leq n \leq \sharp r\}$,
$\mathrm{S1eitem}_G(a) = \{\langle\langle r, n\rangle, \alpha\rangle \mid \langle r, n\rangle \downarrow_a^+ r'$ for some $r, r' \in P, 0 \leq n \leq \sharp r\}$.

Hereafter we fix a CFG G and we often omit G in notations like S0eitem.

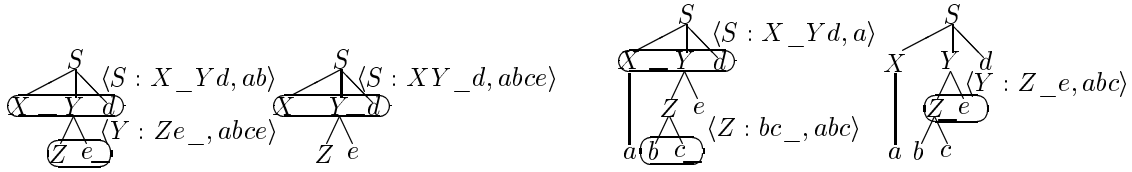
Definition 6 (Shift0, Shift1). *For $a \in V_T$, $x_0 = \langle\langle r^0, n_0\rangle, \alpha_0\rangle \in \mathrm{S0eitem}(a)$, $x_1 = \langle\langle r^1, n_1\rangle, \alpha_1\rangle \in \mathrm{S1eitem}(a)$, we define* $\mathrm{Shift0}(x_0, a)$ *and* $\mathrm{Shift1}(x_1, a) \subseteq J_G$ *by*
$\mathrm{Shift0}(x_0, a) = \{\langle\langle r^0, n_0 + 1\rangle, \alpha_0 \cdot a\rangle\}$,
$\mathrm{Shift1}(x_1, a) = \{\langle\langle r', 1\rangle, \alpha_1 \cdot a\rangle \mid \langle r^1, n_1\rangle \downarrow_a^+ r'\}$.



$x_0 = \langle Z : b\_c, ab\rangle \in \mathrm{S0eitem}(c)$
$\langle Z : bc\_, abc\rangle \in \mathrm{Shift0}(x_0, c)$

$x_1 = \langle S : X\_Yd, ab\rangle \in \mathrm{S1eitem}(c)$
$\langle Z : c\_, abc\rangle \in \mathrm{Shift1}(x_1, c)$

Figure 4: Examples of S0eitem(c) , Shift0(x, c) and S1eitem(c), Shift1(x, c)



$x_0 = \langle Y : Ze\_, abce\rangle \in \mathrm{Redex}$
$x'_0 = \langle S : X\_Yd, ab\rangle \in \mathrm{R0eitem}(x_0)$
$\langle S : XY\_d, abce\rangle \in \mathrm{Reduce0}(x'_0, x_0)$

$x_1 = \langle Z : bc\_, abc\rangle \in \mathrm{Redex}$
$x'_1 = \langle S : X\_Yd, a\rangle \in \mathrm{R1eitem}(x_1)$
$\langle Y : Z\_e, abc\rangle \in \mathrm{Reduce1}(x'_1, x_1)$

Figure 5: Example of R0eitem(x),Reduce0(x', x) and R1eitem(x),Reduce1(x', x)

Example 3. Let $G1$ be a CFG in example 1,
$\langle X : a\_b, a\rangle \in \mathrm{S0eitem}(b)$, $\mathrm{Shift0}(\langle X : a\_b, a\rangle, b) = \{\langle X : a\ b\_, ab\rangle\}$
$\langle S : X\_Y\ d, a\rangle \in \mathrm{S1eitem}(b)$, $\mathrm{Shift1}(\langle S : X\_Y\ d, a\rangle, b) = \{\langle Z : b\_c, ab\rangle\}$
The situation of this example is shown in Fig. 4.

Definition 7 (Redex). *We define* $\mathrm{Redex}_G \subseteq J_G$ *by*
$\mathrm{Redex}_G = \{x \in J_G \mid \mathrm{body}(x) = \langle r, \sharp r\rangle \text{ for some } r \in P\}$.

Definition 8 (R0eitem, R1eitem). *For* $x \in \mathrm{Redex}$ *we define* $\mathrm{R0eitem}(x)$ *and* $\mathrm{R1eitem}(x) \subseteq J_G$ *by*
$\mathrm{R0eitem}(x) = \{\langle\langle r', n'\rangle, \alpha\rangle \mid {r'}_{n'+1} = r_0 \text{ for } r' \text{ and } n' \text{ where } x = \langle\langle r, \sharp r\rangle, \alpha\rangle\}$,
$\mathrm{R1eitem}(x) = \{\langle\langle r', n'\rangle, \alpha\rangle \mid \langle r', n'\rangle \downarrow^+_{r_0} r'' \text{ for some } r'' \in P\}$.

Definition 9 (Reduce0, Reduce1). *For* $x = \langle\langle r, \sharp r\rangle, \alpha\rangle$ *and* $x'_0 = \langle\langle {r'}^0, {n'}_0\rangle, {\alpha'}_0\rangle \in \mathrm{R0eitem}(x)$, *we define* $\mathrm{Reduce0}(x'_0, x) \subseteq J_G$ *by*
$\mathrm{Reduce0}(x'_0, x) = \{\langle\langle {r'}^0, {n'}_0 + 1\rangle, \alpha\rangle\}$.
For $x = \langle\langle r, \sharp r\rangle, \alpha\rangle$ *and* $x'_1 = \langle\langle {r'}^1, {n'}_1\rangle, {\alpha'}_1\rangle \in \mathrm{R1eitem}(x)$, *we define* $\mathrm{Reduce1}(x'_1, x) \subseteq J_G$ *by*
$\mathrm{Reduce1}(x'_1, x) = \{\langle\langle r'', 1\rangle, \alpha\rangle \mid \langle {r'}^1, {n'}_1\rangle \downarrow^+_{r_0} r''\}$.

Example 4. Let $G1$ be a CFG in example 1, $x = \langle Y : Z\ e\_, abce\rangle$, $x'_0 = \langle S : X\_Y\ d, ab\rangle$
$x \in \mathrm{Redex}$, $x'_0 \in \mathrm{R0eitem}(x)$, $\mathrm{Reduce0}(x'_0, x) = \{\langle S : X\ Y\_d, abce\rangle\}$.
For $x = \langle Z : b\ c\_, abc\rangle$, $x'_1 = \langle S : X\_Y\ d, a\rangle$
$x \in \mathrm{Redex}$, $x'_1 \in \mathrm{R1eitem}(x)$, $\mathrm{Reduce1}(x'_1, x) = \{\langle Y : Z\_e, abc\rangle\}$.
The situation of this example is shown in Fig. 5.

3 Parsing Algorithm for Parse Stacks

First we formulate a parsing method similar to [4], but using e-items instead of items. For an input string $\alpha \in {V_T}^*$ we define a set of sequences of e-items $L^\alpha \subseteq {J_G}^*$ inductively on substrings of α and an integer n, which can be seen as a parsing for α.

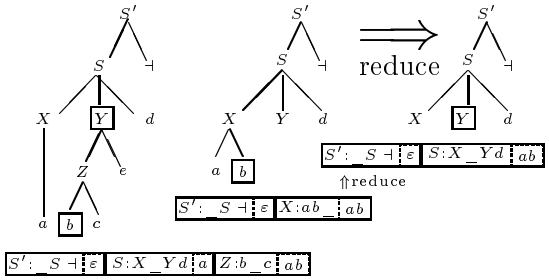


Figure 6: parse stacks in L^{ab} and corresponding parse trees

Example 5. Let $G1$ be a CFG in example 1, parse stacks in L^{ab} and their corresponding parse trees are shown in Fig. 6. In these parse trees paths from e_0 (the root) to b (a leaf) are expressed in thick lines and nodes corresponding to the elements of parse stacks are expressed in boxes.

Algorithm 1 (Parse Stack Parsing).

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

In the following case 2 and case 3 we define elements of $L^{\alpha \cdot a}$ generated by a shift action from elements of L^{α}.

2. *For $\mu \cdot x \in L^{\alpha}$, $\mu \cdot y \in L^{\alpha \cdot a}$ where $x \in \mathrm{S0eitem}(a)$ and $y \in \mathrm{Shift0}(x, a)$.*

3. *For $\mu \cdot x \in L^{\alpha}$, $\mu \cdot x \cdot y \in L^{\alpha \cdot a}$ where $x \in \mathrm{S1eitem}(a)$ and $y \in \mathrm{Shift1}(x, a)$.*

In the following case 4 and case 5 we define elements of $L^{\alpha \cdot a}$ generated by a reduce action from elements of $L^{\alpha \cdot a}$ by induction on n. The induction base is defined by
$\mathrm{Reduce}^{(0)}(L^{\alpha}, a) = \mathrm{Shift0}^{(0)}(L^{\alpha}, a) \cup \mathrm{Shift1}^{(0)}(L^{\alpha}, a)$
where $\mathrm{Shift0}^{(0)}(L^{\alpha}, a)$ is a set of sequences of e-items $(\mu \cdot y)$ defined in case 2 and $\mathrm{Shift1}^{(0)}(L^{\alpha}, a)$ is a set of sequences of e-items $(\mu \cdot x \cdot y)$ defined in case 3.

4. *For $\mu \cdot x' \cdot x \in \mathrm{Reduce}^{(n)}(L^{\alpha}, a)$, $\mu \cdot y \in L^{\alpha \cdot a}$ where $x \in \mathrm{Redex}$, $x' \in \mathrm{R0eitem}(x)$ and $y \in \mathrm{Reduce0}(x', x)$.*

5. *For $\mu \cdot x' \cdot x \in \mathrm{Reduce}^{(n)}(L^{\alpha}, a)$, $\mu \cdot x' \cdot y \in L^{\alpha \cdot a}$ where $x \in \mathrm{Redex}$, $x' \in \mathrm{R1eitem}(x)$ and $y \in \mathrm{Reduce1}(x', x)$.*

Case 4 and case 5 generate $\mathrm{Reduce}^{(n+1)}(L^{\alpha}, a)$ from $\mathrm{Reduce}^{(n)}(L^{\alpha}, a)$ where n corresponds to the times of reduce action. $\mathrm{Reduce}^{(n+1)}(L^{\alpha}, a)$ is defined by
$\mathrm{Reduce}^{(n+1)}(L^{\alpha}, a) = \mathrm{Reduce0}^{(n+1)}(L^{\alpha}, a) \cup \mathrm{Reduce1}^{(n+1)}(L^{\alpha}, a)$
where $\mathrm{Reduce0}^{(n+1)}(L^{\alpha}, a)$ is a set of sequences of the e-items $(\mu \cdot y)$ defined in case 4 and $\mathrm{Reduce1}^{(n+1)}(L^{\alpha}, a)$ is a set of sequences of the e-items $(\mu \cdot x' \cdot y)$ defined in case 5.

Example 6. Let $G1$ be a CFG in example 1, L^{α} of $G1$ for an input string "*abced*" and its substrings are shown in Fig. 7. Each element of L^{α} is a sequence of e-items and corresponds to a parse stack in [5].

Let $\mu^2 = (e_0)$, $x^2 = \langle X : a\_b, a \rangle$, and $y^2 = \langle X : ab\_, ab \rangle$.
As $\mu^2 \cdot x^2 \in L^a$, $x^2 \in \mathrm{S0eitem}(b)$ and $y^2 \in \mathrm{Shift0}_E(x^2, b)$,
$\mu^2 \cdot y^2 \in L^{ab}$, by case 2 of the stack set parse algorithm.

Let $\mu^3 = (e_0)$, $x^3 = \langle S : X\_Y\ d, a \rangle$, $y^3 = \langle Z : b\_c, ab \rangle$.
As μ^3, x^3, y^3 satisfy $\mu^3 \cdot x^3 \in L^a$, $x^3 \in \mathrm{S1eitem}(b)$, $y^3 \in \mathrm{Shift1}_E(x^3, b)$,
$\mu^3 \cdot x^3 \cdot y^3 \in L^{ab}$ by case 3 of the stack set parse algorithm.

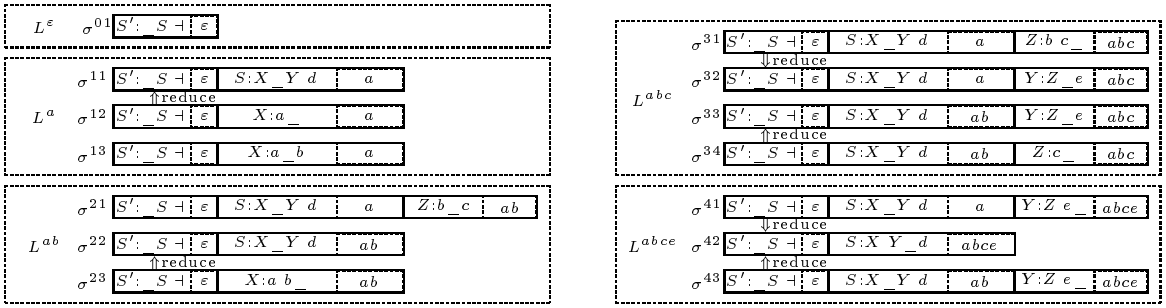


Figure 7: examples of L^{α} for an input "$abce$"

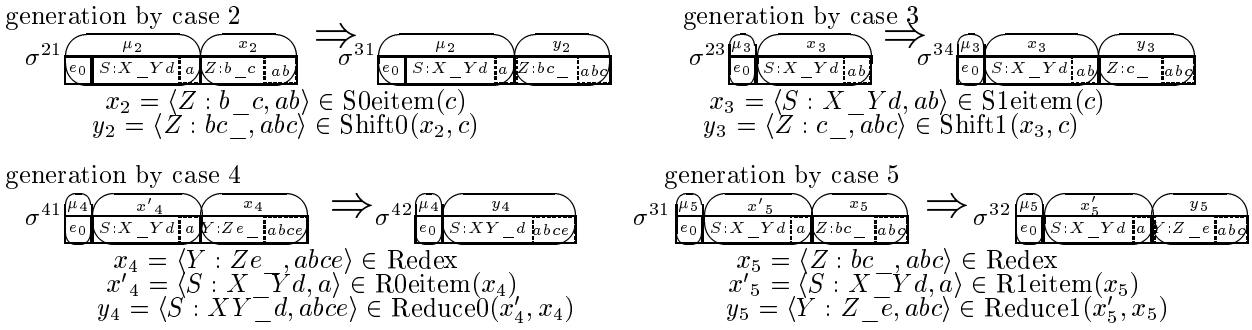


Figure 8: Example of parse stack generation

Let $\mu^4 = (e_0)$, $x'^4 = \langle S : X\_Y\ d, a\rangle$, $x^4 = \langle Y : Z\ e\_, abce\rangle$, $y^4 = \langle S : X\ Y\_d, abce\rangle$.
As μ^4, x^4, x'^4, y^4 satisfy $\mu^4 \cdot x'^4 \cdot x^4 \in \mathrm{Reduce}^{(0)}{}_{E^*}(L^{abc}, e), x^4 \in \mathrm{Redex}, x'^4 \in \mathrm{R0eitem}(x^4), y^4 \in \mathrm{Reduce0}_E(x'^4, x^4)$,
$\mu^4 \cdot y^4 \in L^{abce}$ by case 4 of the stack set parse algorithm.

Let $\mu^5 = (e_0)$, $x'^5 = \langle S : X\_Y\ d, a\rangle$, $x^5 = \langle Z : b\ c\_, abc\rangle$, $y^5 = \langle Y : Z\_e, abc\rangle$.
As μ^5, x^5, x'^5, y^5 satisfy $\mu^5 \cdot x'^5 \cdot x^5 \in \mathrm{Reduce}^{(0)}{}_{E^*}(L^{ab}, c), x^5 \in \mathrm{Redex}, x'^5 \in \mathrm{R1eitem}(x^5), y^5 \in \mathrm{Reduce1}_E(x'^5, x^5)$,
$\mu^5 \cdot x'^5 \cdot y^5 \in L^{abc}$ by case 5 of the stack set parse algorithm.

Examples of parse stack generation are shown in Fig. 8.

As shown in the example above, we can get all parse stacks for an input string by starting from L^{ε} as the initial state and construct $L^{\alpha \cdot a}$ from L^{α} if the current input is α and the next symbol is a.

4 Graph-Structured Stack Parsing

In this section I modify Algorithm 1 utilizing the idea of graph-structured stacks [3], [5], [11]. Each element of a stack has parents, i. e. , those pointed by the element, which are its previous elements in the parse stack. In this algorithm, the parents of a stack element, an e-item x, are a set of e-items, denoted by $\mathrm{Parent}(x)$.

Algorithm 2 (Graph-structured stack parsing).
For $\alpha \in {V_T}^*$ we define $E_\alpha \subseteq J_G$, and $\mathrm{Parent}(x)$ *for $x \in E_\alpha$, inductively on elements of a string α and a positive integer n, as follows:*

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

In the following case 2 and case 3 we define elements of $E_{\alpha \cdot a}$ generated by a shift action from elements of E_α.

2. *For* $x \in E_\alpha,\ y \in E_{\alpha\cdot a}$ *where* $x \in \mathrm{S0eitem}(a), y \in \mathrm{Shift0}(x,a)$.
For this $y, \mathrm{Parent}(y)$ *is defined by*
$\mathrm{Parent}(y) = \{w \mid w \in \mathrm{Parent}(z)$ for z where $z \in E_\alpha, z \in \mathrm{S0eitem}(a),$
$y \in \mathrm{Shift0}(z,a)\}$

3. *For* $x \in E_\alpha,\ y \in E_{\alpha\cdot a}$ *where* $x \in \mathrm{S1eitem}(a), y \in \mathrm{Shift1}(x,a)$.
For this $y,\ \mathrm{Parent}(y)$ *is defined by*
$\mathrm{Parent}(y) = \{z \mid z \in E_\alpha, z \in \mathrm{S1eitem}(a), y \in \mathrm{Shift1}(z,a)\}$.

In the following case 4 and case 5 we define elements of $E_{\alpha\cdot a}$ *generated by a reduce action from elements of* $E_{\alpha\cdot a}$ *by induction on* n. *The induction base is defined by*
$\mathrm{Reduce}_{(0)}(E_\alpha, a) = \mathrm{Shift0}_{(0)}(E_\alpha, a) \cup \mathrm{Shift1}_{(0)}(E_\alpha, a)$
where $\mathrm{Shift0}_{(0)}(E_\alpha, a)$ *is a set of e-items* (y) *defined in case 2 and* $\mathrm{Shift1}_{(0)}(E_\alpha, a)$ *is a set of e-items* (y) *defined in case 3.*

4. *For* $x \in \mathrm{Reduce}_{(n)}(E_\alpha, a),\ y \in E_{\alpha\cdot a}$ *where* $x' \in \mathrm{Parent}(x), x \in \mathrm{Redex}, x' \in \mathrm{R0eitem}(x), y \in \mathrm{Reduce0}(x', x)$.
For this $y, \mathrm{Parent}(y)$ *is defined by*
$\mathrm{Parent}(y) = \{w \mid w \in \mathrm{Parent}(z')$ for z, z' where $z \in \mathrm{Reduce}_{(n)}(E_\alpha, a),$
$z' \in \mathrm{Parent}(z), z \in \mathrm{Redex}, z' \in \mathrm{R0eitem}(z), y \in \mathrm{Reduce0}(z', z)\}$

5. *For* $x \in \mathrm{Reduce}_{(n)}(E_\alpha, a),\ y \in E_{\alpha\cdot a}$ *where* $x' \in \mathrm{Parent}(x), x \in \mathrm{Redex}, x' \in \mathrm{R1eitem}(x), y \in \mathrm{Reduce1}(x', x)$.
For this $y, \mathrm{Parent}(y)$ *is defined by*
$\mathrm{Parent}(y) = \{z' \mid$ for z' where $z \in \mathrm{Reduce}_{(n)}(E_\alpha, a),$
$z' \in \mathrm{Parent}(z), z \in \mathrm{Redex}, z' \in \mathrm{R1eitem}(z), y \in \mathrm{Reduce1}(z', z)\}$.

Case 4 and case 5 generate $\mathrm{Reduce}_{(n+1)}(E_\alpha, a)$ *from* $\mathrm{Reduce}_{(n)}(E_\alpha, a)$ *where* n *corresponds to the times of reduce action.* $\mathrm{Reduce}_{(n+1)}(E_\alpha, a)$ *is defined by*
$\mathrm{Reduce}_{(n+1)}(E_\alpha, a) = \mathrm{Reduce0}_{(n+1)}(E_\alpha, a) \cup \mathrm{Reduce1}_{(n+1)}(E_\alpha, a)$
where $\mathrm{Reduce0}_{(n+1)}(E_\alpha, a)$ *is a set of e-items* (y) *defined in case 4 and* $\mathrm{Reduce1}_{(n+1)}(E_\alpha, a)$ *is a set of e-items* (y) *defined in case 5.*

Example 7. Let $G1$ be a CFG in example 1. The graph structured stack of $G1$ for an input string "*abced*" is shown in Fig. 9, where elements of $\mathrm{Parent}(x)$ are denoted by an arrow from x to each elements when $\mathrm{Parent}(x) \neq \{e_0\}$. For example, there is an arrow from $x = \langle Y : Z\_e,\ abc\rangle$ to $\langle S : X\_Yd,\ ab\rangle$ since $\mathrm{Parent}(x) = \{\langle S : X\_Yd,\ a\rangle,\ \langle S : X\_Yd,\ ab\rangle\}$ and there is no arrow from $x' = \langle S : X\_Yd,\ ab\rangle$ to e_0 since $\mathrm{Parent}(x') = \{e_0\}$.

Let $x^2 = \langle X : a\_b, a\rangle,\ y^2 = \langle X : a\ b\_, ab\rangle$.
As $x^2,\ y^2$ satisfy $x^2 \in E_a,\ x^2 \in \mathrm{S0eitem}(b),\ y^2 \in \mathrm{Shift0}_E(x^2, b)$,
$y^2 \in E_{ab}$ and $\mathrm{Parent}(y^2) = \mathrm{Parent}(x^2) = \{e_0\}$ by case 2 of the graph structured stack parse algorithm.

Let $x^3 = \langle S : X\_Y\ d, a\rangle,\ y^3 = \langle Z : b\_c, ab\rangle$.
As $x^3,\ y^3$ satisfy $x^3 \in E_a,\ x^3 \in \mathrm{S1eitem}(b),\ y^3 \in \mathrm{Shift1}_E(x^3, b)$,
$y^3 \in E_{ab}$ and $\mathrm{Parent}(y^3) = \{x_3\}$ by case 3 of the graph structured stack parse algorithm.

Let $x'^4 = \langle S : X\_Y\ d, a\rangle,\ x^4 = \langle Y : Z\ e\_, abce\rangle,\ y^4 = \langle S : X\ Y\_d, abce\rangle$.
As $x^4, x'^4,\ y^4$ satisfy $x^4 \in \mathrm{Reduce}^{(0)}{}_E(E_{abc}, e), x^4 \in \mathrm{Redex},\ x'^4 \in \mathrm{R0eitem}(x^4),\ y^4 \in \mathrm{Reduce0}_E(x'^4, x^4)$,
$y^4 \in E_{abce}$ and $\mathrm{Parent}(y^4) = \mathrm{Parent}(x'^4) = \{e_0\}$ by case 4 of the graph structured stack parse algorithm.

Let $x'^5 = \langle S : X\_Y\ d, a\rangle,\ x^5 = \langle Z : b\ c\_, abc\rangle,\ y^5 = \langle Y : Z\_e, abc\rangle$.
As $x^5, x'^5,\ y^5$ satisfy $x^5 \in \mathrm{Reduce}^{(0)}{}_E(E_{ab}, c), x^5 \in \mathrm{Redex},\ x'^5 \in \mathrm{R1eitem}(x^5),\ y^5 \in \mathrm{Reduce1}_E(x'^5, x^5)$,

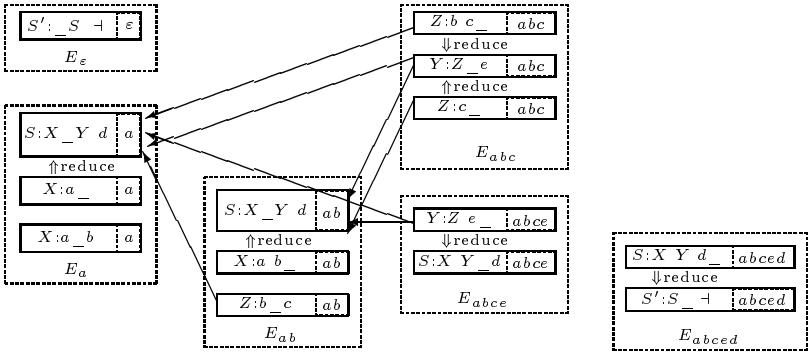
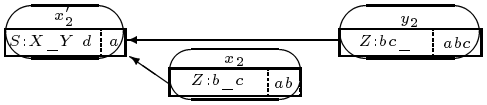


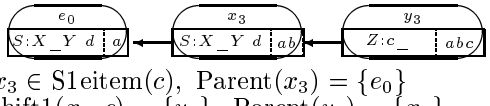
Figure 9: Graph structured stack for the input "*abced*"

Generation by case 2



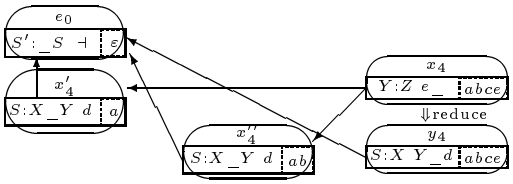
$x_2 \in \text{S0eitem}(c),\ \text{Parent}(x_2) = \{x_2'\}$
$\text{Shift0}(x_2, c) = \{y_2\}$
$\text{Parent}(y_2) = \text{Parent}(x_2) = \{x_2'\}$

Generation by case 3



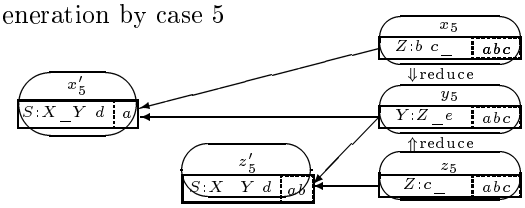
$x_3 \in \text{S1eitem}(c),\ \text{Parent}(x_3) = \{e_0\}$
$\text{Shift1}(x_3, c) = \{y_3\},\ \text{Parent}(y_3) = \{x_3\}$

Generation by case 4



$x_4', x_4'' \in \text{R0eitem}(x_4),\ \text{Parent}(x_4) = \{x_4', x_4''\}$
$\text{Reduce0}(x_4', x_4) = \text{Reduce0}(x_4'', x_4) = \{y_4\}$
$\text{Parent}(x_4') = \text{Parent}(x_4'') = \{e_0\}$
$\text{Parent}(y_4) = \text{Parent}(x_4') \cup \text{Parent}(x_4'') = \{e_0\}$

Generation by case 5



$x_5' \in \text{R1eitem}(x_5),\ \text{Parent}(x_5) = \{x_5'\}$
$z_5' \in \text{R1eitem}(z_5),\ \text{Parent}(z_5) = \{z_5'\}$
$\text{Reduce1}(x_5', x_5) = \text{Reduce1}(z_5', z_5) = \{y_5\}$
$\text{Parent}(y_5) = \text{Parent}(x_5) \cup \text{Parent}(z_5) = \{x_5', z_5'\}$

Figure 10: Example of Graph Structured Stack Generation

$y^5 \in E_{abc}$ and $\text{Parent}(y^5) = \{\langle S : X\_Y\ d, a\rangle, \langle S : X\_Y\ d, ab\rangle, \}$ by case 5 of the graph structured stack parse algorithm.

Examples of graph structured stack generation are shown in Fig. 10.

The following Lemma 1 holds from definitions of $\text{Shift0}(x, a)$ and $\text{Reduce0}(x', x)$.

Lemma 1. *For $a \in V_T$ and x, $y \in J_G$, if $x \in \text{S0eitem}(a)$ and $y \in \text{Shift0}(x, a)$, then* $\text{Parent}(y) = \text{Parent}(x)$.

Proof. If y is expressed as $\langle\langle r, n\rangle, \alpha \cdot a\rangle$, x is expressed as $\langle\langle r, n-1\rangle, \alpha\rangle$ and is unique. Therefore, x is the only e-item that satisfies the conditions for z in the definition of $\text{Parent}(y)$ in Case 2 of Algorithm 2. □

Definition 10 (E^α). *For $\alpha \in {V_T}^*$, $E^\alpha \subseteq {J_G}^*$ is defined by*
$E^\alpha = \{\sigma \mid \text{Parent}(\sigma_1) = \emptyset, \sigma_j \in \text{Parent}(\sigma_{j+1})(1 \le j < \sharp\sigma), \sigma_{\sharp\sigma} \in E_\alpha\}$.

Example 8. For $G1$ of example 1, $E^{ab} = \{\sigma^1, \sigma^2, \sigma^3\}$ where
$\sigma^1 = ({\sigma^1}_1, {\sigma^1}_2) = (e_0, \langle X : a\ b\_, ab\rangle)$,
$\sigma^2 = ({\sigma^2}_1, {\sigma^2}_2) = (e_0, \langle S : X\_Y\ d, ab\rangle)$,
$\sigma^3 = ({\sigma^3}_1, {\sigma^3}_2, {\sigma^3}_3) = (e_0, \langle S : X\_Y\ d, a\rangle, \langle Z : b\_c, ab\rangle)$.

Relations between elements of $\sigma^1, \sigma^2, \sigma^3$ are shown in Fig. 11, where every element of $\text{Parent}(x)$ is denoted by an arrow from x to each element. For example, for ${\sigma^1}_2 = \langle X : ab\_,\ ab\rangle$, $\text{Parent}({\sigma^1}_2) = \{e_0\}$ and there is an arrow from x to e_0.

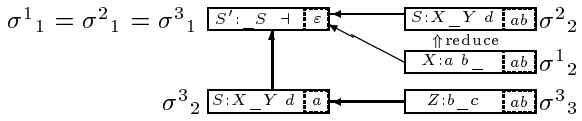


Figure 11: Elements of $\sigma^1, \sigma^2, \sigma^3$

We have the basic theorems about L^α and E^α. They are proved inductively based on definitions of case (1) to case 5 in Algorithm1 and Algorithm2.

Theorem 1. *For $\alpha \in {V_T}^*$, if $\lambda \in L^\alpha$ then $\lambda \in E^\alpha$.*

Theorem 2. *For $\alpha \in {V_T}^*$, if $\lambda \in E^\alpha$ then $\lambda \in L^\alpha$.*

5 Pruned Graph Structured Stack Parsing

5.1 Outline of Pruned Graph Structured Stack Parsing

We denote the size of a finite set H by $|H|$.

Definition 11 ($\mapsto$). *For e-items $x, y \in J_G$, we define $x \mapsto y$ if one of the following is satisfied:*

1. $x \in \mathrm{S0eitem}(a)$ *and* $y \in \mathrm{Shift0}(x, a)$,

2. $x \in \mathrm{S1eitem}(a)$ *and* $y \in \mathrm{Shift1}(x, a)$,

3. $x \in \mathrm{Redex}$ *and* $y \in \mathrm{Reduce0}(x', x)$ for some $x' \in \mathrm{R0eitem}(x)$,

4. $x \in \mathrm{Redex}$ *and* $y \in \mathrm{Reduce1}(x', x)$ for some $x' \in \mathrm{R1eitem}(x)$.

The transitive closure of $\mapsto$ is denoted by $\mapsto^*$.

Example 9. For $G1$ in example 1,
$\langle Z : b\_c, ab\rangle \mapsto \langle Z : bc\_, abc\rangle$ holds because $\langle Z : bc\_, abc\rangle$ is generated from $\langle Z : b\_c, ab\rangle$ from case 2 of Algorithm 2, as stated in example 6.
Similarly, $\langle S : X\_Yd, a\rangle \mapsto^* \langle S : XY\_d, abce\rangle$ holds because
$\langle S' : \_S\ \dashv, \varepsilon\rangle \mapsto \langle X : a\_, a\rangle \mapsto \langle S : X\_Yd, a\rangle \mapsto \langle Z : b\_c, ab\rangle \mapsto \langle Z : bc\_, abc\rangle \mapsto \langle Y : Z\_e, abc\rangle \mapsto$
$\langle Y : Ze\_, abce\rangle \mapsto \langle S : XY\_d, abce\rangle \mapsto \langle S' : S\_\ \dashv, abced\rangle$ hold

By definitions of $\mapsto^*$ and $\mathrm{Parent}(x)$, we easily have:

Proposition 1. *If $y \in \mathrm{Parent}(x)$ then $y \mapsto^* x$.*

Definition 12 (separable). *For a set of e-items $D \subseteq J_G$, D is called separable if* $\mathrm{body}(x) = \mathrm{body}(y)$ *implies $x = y$ for $x, y \in D$.*

For a separable set D we can naturally embed D into I_G by ignoring the input parts of the elements. By this embedding we obtain the following easy proposition.

Proposition 2. *For a set of e-items $D \subseteq J_G$, if D is separable then $|D| \leq |I_G|$. In particular, a separable set is a finite set.*

Following lemma holds for separable parent sets in case 4of Algorithm 2.

Lemma 2. *For x, x' and y that satisfy $x \in \mathrm{Redex}$, $x' \in \mathrm{R0eitem}(x) \cap \mathrm{Parent}(x)$, $y \in \mathrm{Reduce0}(x', x)$, if* $\mathrm{Parent}(x)$ *is separable then* $\mathrm{Parent}(y) = \mathrm{Parent}(x')$.

Proof. If there exists $z' \in \mathrm{R0eitem}(x) \cap \mathrm{Parent}(x)$ satisfying $y \in \mathrm{Reduce0}(x', x)$, $\mathrm{body}(x') = \mathrm{body}(z')$ holds from $y \in \mathrm{Reduce0}(x', x)$ and $y \in \mathrm{Reduce0}(z', x)$. Therefore $x' = z'$ holds if $\mathrm{Parent}(x)$ is separable. It means that x' that satisfy $y \in \mathrm{Reduce0}(x', x)$ is unique and $\mathrm{Parent}(y) = \mathrm{Parent}(x')$ holds from definition of $\mathrm{Parent}(y)$ in case 4of Algorithm 2. □

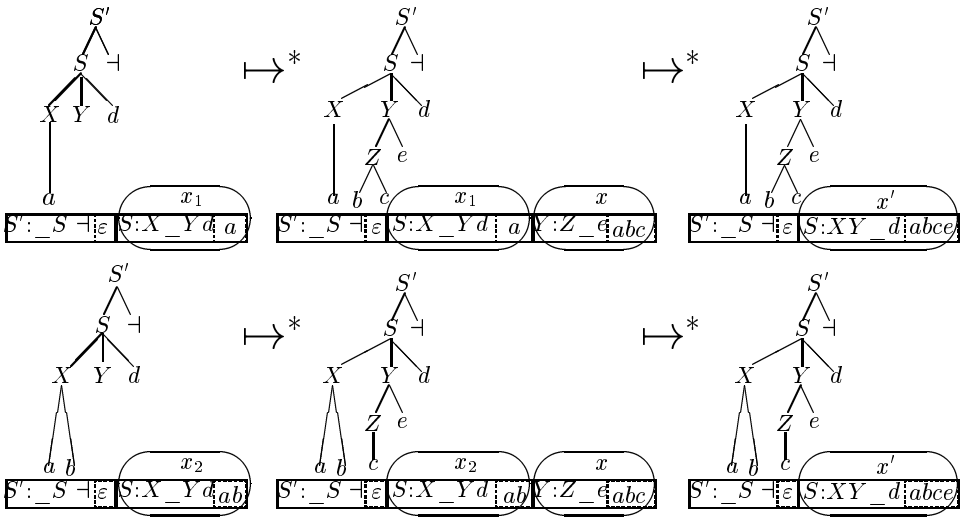


Figure 12: example of Theorem 3: Parse process for "abce"

Unfortunately, all parent sets are not separable.

Example 10. As is seen from Fig. 2, Parent($\langle Y : Z\_e,\ abc\rangle$) is not separable, because it contains $\langle S : X\_Yd,\ a\rangle$ and $\langle S : X\_Yd,\ ab\rangle$.

For parent sets which are not separable, we have the following important theorem.

Theorem 3. *Let* $x_1,\ x_2 \in \mathrm{Parent}(x)$ *for an e-item* x*, and assume that* $\mathrm{body}(x_1) = \mathrm{body}(x_2) = \langle r, n\rangle$ *and* $\mathrm{input}(x_1) \neq \mathrm{input}(x_2)$*. If* $x \mapsto^* \langle\langle r, n+1\rangle, \alpha\rangle$ *for some* $\alpha \in {V_T}^*$*, then* $x_1 \mapsto^* \langle\langle r, n+1\rangle, \alpha\rangle$ *and* $x_2 \mapsto^* \langle\langle r, n+1\rangle, \alpha\rangle$.

Proof. $x_1 \mapsto^* x$ and $x_2 \mapsto^* x$ hold because $x_1,\ x_2 \in \mathrm{Parent}(x)$. Therefore, $x_1 \mapsto^* x \mapsto^* \langle\langle r, n+1\rangle, \alpha\rangle$ and $x_2 \mapsto^* x \mapsto^* \langle\langle r, n+1\rangle, \alpha\rangle$ are satisfied. □

Example 11. Let x_1, x_2, x *and* x' be $x_1 = \langle S : X\_Yd,\ a\rangle, x_2 = \langle S : X\_Yd,\ ab\rangle, x = \langle Y : Z\_e,\ abc\rangle$ *and* $x' = \langle S : XY\_d,\ ab\rangle$. In this case $x_1 \mapsto^* x'$ and $x_2 \mapsto^* x'$ holds since $x_1, x_2 \in \mathrm{Parent}(x)$ and $x \mapsto^* x'$.

This situation is shown in Fig. 12, in which the upper part indicates a parse process corresponding to $x_1 \mapsto^* x \mapsto^* x'$ and the lower part indicates a parse process corresponding to $x_2 \mapsto^* x \mapsto^* x'$. As Parent($x$) is not separable, two different parse trees exist for an input "*abc*" that are indicated in the center of Fig. 12. The right part of Fig. 12 shows that after the reduction of "$Y : Ze$", there exists 2 different parse trees but their parse stack is same because the contents of parse stack do not depend on the substructures of X and Y after the reduction of productions for X and Y.

As top elements of stacks in the middle part are same (x), moves of both stacks are same until their top elements are reduced. After top elements are reduced, elements under top elements (x_1, x_2) are shifted to a same element (x') and both stacks become same. Therefore moves of both stacks are same after their top elements is reduced that means moves of both stacks in the middle part are same for any input string. It means that we can prune one of stacks in the middle part.

5.2 Definition of Pruned Graph Structured Stack Parsing

Definition 13 (E)**.** *We define* E *as* $E = \cup_{\alpha \in {V_T}^*} E_\alpha$.

Definition 14 (I^α, E^x, I^x). *For $\alpha \in {V_T}^*, x \in E_\alpha$, we define*
$I^\alpha = \{\rho \mid \exists \sigma \in E^\alpha$ *where* $\sharp\rho = \sharp\sigma,\ \rho_j = \mathrm{body}(\sigma_j)\ \ (1 \leq j \leq \sharp\rho)\}$.
$E^x = \{\sigma \mid \sigma \in E^\alpha, \sigma_\sharp = x\}$.
$I^x = \{\rho \mid \rho \in I^\alpha, \rho_\sharp = x\}$.

Elements of I^α are body parts of elements of E^α and E^x is a subset of E^α whose element σ satisfies $\sigma_\sharp = x$.

Example 12. For $G1$ in example 1, $I^{abc} = \{\rho^1, \rho^2, \rho^4\}$ where
$\rho^1 = ({\rho^1}_1, {\rho^1}_2, {\rho^1}_3) = (i_0, S : X\_Yd, Z : bc\_)$.
$\rho^2 = ({\rho^2}_1, {\rho^2}_2, {\rho^2}_3) = (i_0, S : X\_Yd, Y : Z\_e)$,
$\rho^4 = ({\rho^4}_1, {\rho^4}_2, {\rho^4}_3) = (i_0, S : X\_Yd, Z : c\_)$.
For $x = \langle Y : Z\_e, abc\rangle$, $I^x = \{\rho^2\}$.

From the definition of I^x, we have

Lemma 3. *For $x \in E$,*
$I^x = \bigcup_{y \in \mathrm{Parent}(x)} \{\rho \cdot \mathrm{body}(x) \mid \rho \in I^y\}$.

From proposition 11 in Sec 5.3, we can prune x if $I^x \subseteq I^y$ holds for $x, y \in \mathrm{Parent}(z)$. We define a binary relation that shows $I^x \subseteq I^y$ over $x, y \in \mathrm{Parent}(z)$ that satisfy $\mathrm{body}(x) = \mathrm{body}(y)$.

Definition 15 ($\prec$). *For $x, y \in E$ that satisfy* $\mathrm{body}(x) = \mathrm{body}(y)$,
we define $x \prec y$ if the following condition is satisfied,
$\forall x' \in \mathrm{Parent}(x),\ \ \exists y' \in \mathrm{Parent}(y)\ .\quad x' = y'$ *or* $x' \prec y'$

Proposition 3. *If $x \prec y$ then $I^x \subseteq I^y$.*

Proof. As context free grammars in this report are ε-free, we have (A),(B) from Algorithm 2.
(A) For $x, y \in E$, if $x \in \mathrm{Parent}(y)$ then $\sharp\mathrm{input}(x) < \sharp\mathrm{input}(y)$.
(B) For $x \in E$ if $\sharp\mathrm{input}(x)$=0 then $x = e_0$. Let $\mathrm{L}(x, y)$ be $\mathrm{L}(x, y)$=max($\sharp\mathrm{input}(x)$, $\sharp\mathrm{input}(y)$) and we prove this proposition by induction on $\mathrm{L}(x, y)$.

1. $\mathrm{L}(x, y) = 1$ As $\sharp\mathrm{input}(x)$=$\sharp\mathrm{input}(y)$=1 holds in this case, $\mathrm{Parent}(x) = \mathrm{Parent}(y) = \{e_0\}$ holds from (A), (B) and this proposition is satisfied.

2. We assume that this proposition is satisfied for $\mathrm{L}(x, y) \leq k$ and prove this proposition for $\mathrm{L}(x, y) = k + 1$. From $x \prec y$, for any $x' \in \mathrm{Parent}(x)$ there exists $y' \in \mathrm{Parent}(y)$ that satisfies $x' = y'$ or $x' \prec y'$.
   For case of $x' = y'$, $I^{x'} \subseteq I^{y'}$ holds.
   For case of $x' \prec y'$, $\sharp\mathrm{input}(x') \leq k$, $\sharp\mathrm{input}(y') \leq k$ holds from $x' \in \mathrm{Parent}(x), y' \in \mathrm{Parent}(y)$ and (A). Therefore $\mathrm{L}(x', y') \leq k$ holds and $I^{x'} \subseteq I^{y'}$ holds from the induction hypothesis.
   In both cases $I^{x'} \subseteq I^{y'}$ holds. From lemma3, for any $\rho \in I^x$, there exist $x' \in \mathrm{Parent}(x)$, $\rho' \in I^{x'}$ that satisfy $\rho = \rho' \cdot \mathrm{body}(x)$. From $I^{x'} \subseteq I^{y'}$, $\rho' \in I^{y'}$ holds. This proposition holds from $\rho = \rho' \cdot \mathrm{body}(x) = \rho' \cdot \mathrm{body}(y) \in I^y$.

□

Definition 16 (representative, representable). *For $D \subseteq E$, $x \in D$ and $D_x = \{y \mid y \in D, \mathrm{body}(y) = \mathrm{body}(x)\}$, $p_x \in D_x$ is called the representative of x if and only if $z \prec p_x$ holds for any $z \in D_x$. $D$is called representable if and only if the representative of x exists for any $x \in D$.*

D_x is a set of elements whose body parts are same as the body part of x and the representative of x is the unpruned element of D_x. If D is representable, we can prune elements of D to their representative.

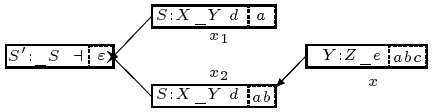


Figure 13: Prune(Parent($\langle Y : Z\_e, abc\rangle$)) in $G1$

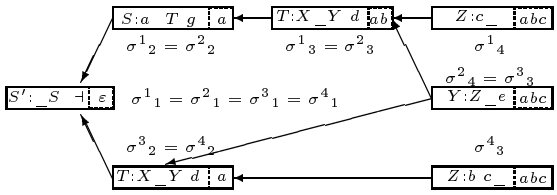


Figure 14: Unprunable case in $G2$

Example 13. For x, x_1, x_2 in example 11, x_1 and x_2 are both representative of x because $\mathrm{Parent}(x) = \{x_1, x_2\}$, $x_1 \prec x_2$, $x_2 \prec x_1$ holds. The case for Prune(Parent(x))=$\{x_2\}$ is shown in Fig. 13.

From proposition 3 and Definition 16, we have

Proposition 4. *If p_x and p'_x are representatives of x then $I^{p_x} = I^{p'_x}$.*

Definition 17 (Prune). *For a representable set $D \subseteq E$,* Prune(D) *is a set whose elements are one of representative for each element of D.*

As shown in example 13, there may be more than one representatives for one element of D. However we can choose any one of them as the element of Prune(D) from proposition 4.

From definition 17, we have

Proposition 5. *For $D \subseteq E$, if D representable then* Prune(D) *is separable.*

We define context free grammars for this parsing algorithm.

Definition 18 (prunable). *A context free grammar G is called prunable if and only if Parent(x) is representable for any $x \in E$.*

Example 14. $G1$ is prunable because Parent(x) is representable from example 13.

Example 15. Let $G2$ be a context free grammar with following productions.

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

Relations of elements of $E^{abc} = \{\sigma^1, \sigma^2, \sigma^3, \sigma^4\}$ are shown in Fig. 14. From Fig. 14, neither $\langle T : X\_Y\ d, a\rangle \prec \langle T : X\_Y\ d, ab\rangle$ nor $\langle T : X\_Y\ d, ab\rangle \prec \langle T : X\_Y\ d, a\rangle$ hold. Therefore $G2$ is not prunable because Parent($\langle Y : Z\_e, abc\rangle$) is not representable.

Algorithm 3 (Pruned Graph-Structured Stack Parsing). *An algorithm in which "*Parent(y) = P$'$(y)*" in case 5of Algorithm 2 is modified to "*Parent(y) = Prune(P$'$(y))*" is called a Pruned Graph-Structured Stack Parsing algorithm.*

Proposition 6. *For x, y in Algorithm2,*
if Parent(x) *is separable then* Parent(y) *defined in case 2, 3, 4 is separable.*

Proof. In Case 2, Parent(x) = Parent(y) from lemma 1 and Parent(y) is separable because Parent(x) is separable. In Case 3, Parent(y) $\subseteq E_\alpha$ and Parent(y) is separable because E_α is separable. In Case 4, Parent(x) is separable and Parent(y) is separable from lemma. □

From this proposition, Parent(y) defined in case 2, 3, 4, 5 of Pruned Graph-Structured Stack Parsing are separable.

From proposition 3, definition 16, definition 17, we have

Theorem 4. *For $x \in E$ satisfying D is representable,*
$\bigcup_{y \in \mathrm{Parent}(x)} I^y = \bigcup_{y \in \mathrm{Prune}(\mathrm{Parent}(x))} I^y$.

From this theorem, $I^x = \bigcup_{y \in \mathrm{Parent}(x)} I^y$ is conserved if Parent(x) is modified to Prune(Parent(x)). It means that the result of Pruned Graph-Structured Stack Parsing in which Parent(x) of Graph-Structured Stack Parsing is modified to Prune(Parent(x)) is same as the result of Graph-Structured Stack Parsing.

5.3 Correctness of the Pruned Graph-Structured Stack Parsing

In this section, I show the correctness of Pruned Graph-Structured Stack Parsing by showing that the result (whether the input is a word of the grammar) of Graph-Structured Stack Parsing is not changed if Parent(x) is modified to Prune(Parent(x)).

From definition 13, we have

Proposition 7. *For $\alpha' \in {V_T}^*$*
α is a word of G
if and only if $\langle i_0, \varepsilon \rangle = \langle S' : \_\, S \dashv, \varepsilon \rangle \mapsto^* \langle S' : S\_ \dashv, \alpha' \rangle = \langle \mathrm{Succ}(i_0), \alpha' \rangle$ holds

Proposition 8. *For $\alpha \in {V_T}^*$,*
there exists $\beta \in {V_T}^*$ that satisfies $\alpha \cdot \beta$ is a word of G
if and only if $\sigma \in E^\alpha$ that satisfy $\langle i_0, \varepsilon \rangle \mapsto^* \sigma_\sharp \mapsto^* \langle \mathrm{Succ}(i_0), \alpha \cdot \beta \rangle$ exists.

Example 16. For $\alpha = ab, \beta = ced$,
$\alpha \cdot \beta = abced$ is a word of $G1$ and $\sigma^2 \in E^{ab} = E^\alpha$ in example 8 satisfies
$\langle i_0, \varepsilon \rangle = \langle S' : \_S \dashv, \varepsilon \rangle \mapsto^* \sigma^2_\sharp = \langle S : X\_Yd, ab \rangle \mapsto^* \langle S' : S\_ \dashv, abced \rangle = \langle \mathrm{Succ}(i_0), \alpha \cdot \beta \rangle$ from example 9.

For σ that satisfies condition of proposition 8, every elements(σ_j) of σ will be popped while reading and parsing β. After σ_j will be popped, body(σ_{j-1}), which is the body part of an element of the parent set of σ_j will shift to Succ(body(σ_{j-1})). After this shifted element will be popped, the body part of σ_{j-2} will shift to Succ(body(σ_{j-2})) and so on. From this, we have

Proposition 9. *For $\alpha \in {V_T}^*$,*
there exists $\beta \in {V_T}^*$ that satisfy $\alpha \cdot \beta$ is a word of G if and only if there exist $\sigma \in E^\alpha, \beta_j \in {V_T}^*, \sigma^j \in E^{\alpha \cdot \beta_j}$ $(1 \leq j \leq \sharp\sigma - 1)$ that satisfy following conditions.

1. $\sharp\sigma^j = \sharp\sigma - j$
2. ${\sigma^j}_k = \sigma_k$ $(1 \leq k \leq \sharp\sigma^j - 1 = \sharp\sigma - j - 1)$
3. ${\sigma^j}_\sharp = \langle \mathrm{Succ}(\mathrm{body}(\sigma_{\sharp\sigma - j})), \alpha \cdot \beta_j \rangle$
4. $\beta_{\sharp\sigma - 1} = \beta$

Example 17. For $\alpha = abc \in {V_T}^*$,
there exists $\beta = ed \in {V_T}^*$ that satisfy $\alpha \cdot \beta$ is a word of $G1$.
For σ^{21}, σ^{42} in Fig. 7
$\sigma = \sigma^{21}$, $\beta_1 = e$, $\beta_2 = \beta = ed$,
$\sigma^1 = \sigma^{42} \in E^{abce} = E^{\alpha \cdot \beta_1}$, $\sigma^2 = (\langle S' : S\_ \dashv, abced \rangle) \in E^{abced} = E^{\alpha \cdot \beta_2}$
satisfy conditions of proposition 9. Relations between elements of $\sigma, \sigma^1, \sigma^2$ are shown in Fig. 15.

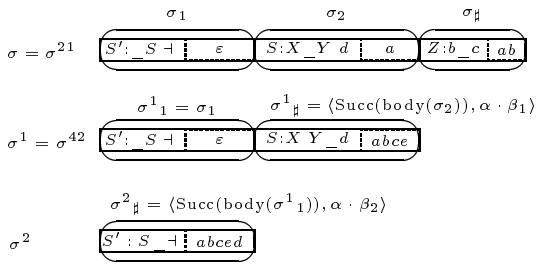


Figure 15: example of proposition.9 in $G1$

Between top elements of σ and σ^j,
$\sigma_\sharp \mapsto^* \sigma^1{}_\sharp = \langle \mathrm{Succ}(\mathrm{body}(\sigma_{\sharp\sigma-1})), \alpha \cdot \beta_1 \rangle$,
$\langle \mathrm{Succ}(\mathrm{body}(\sigma_{\sharp\sigma-j})), \alpha \cdot \beta_j \rangle = \sigma^j{}_\sharp \mapsto^* \sigma^{j+1}{}_\sharp = \langle \mathrm{Succ}(\mathrm{body}(\sigma_{\sharp\sigma-j-1})), \alpha \cdot \beta_{j+1} \rangle$ holds for $1 \leq j \leq \sharp\sigma - 2$ and we have

Proposition 10. *For $\alpha \in {V_T}^*$,*
there exists $\beta \in {V_T}^*$ that satisfy $\alpha \cdot \beta$ is a word of G
if and only if $\sigma \in E^\alpha, \beta_j \in {V_T}^*$ $(1 \leq j < \sharp\sigma)$ that satisfy following conditions

1. $\sigma_\sharp \mapsto^* \langle \mathrm{Succ}(\mathrm{body}(\sigma_{\sharp\sigma-1})), \alpha \cdot \beta_1 \rangle$.
2. $\langle \mathrm{Succ}(\mathrm{body}(\sigma_{\sigma-j})), \alpha \cdot \beta_j \rangle \mapsto^* \langle \mathrm{Succ}(\mathrm{body}(\sigma_{\sigma-j-1})), \alpha \cdot \beta_{j+1} \rangle$ $(1 \leq j \leq \sharp\sigma - 2)$.

We have a following proposition because conditions of above proposition depend on the body part of σ.

Proposition 11. *For $\alpha, \beta \in {V_T}^*, \sigma \in E^\alpha$ that satisfy the following condition*
$\langle i_0, \varepsilon \rangle \mapsto^* \sigma_\sharp \mapsto^* \langle \mathrm{Succ}(i_0), \alpha \cdot \beta \rangle$
and for $x, y \in \mathrm{Parent}(\sigma_\sharp)$ that satisfy $I^x = I^y$,
if there exist $\gamma \in {V_T}^*, \sigma_x \in E^x$ that satisfy
$\sigma_\sharp \mapsto^* \langle \mathrm{Succ}(\mathrm{body}((\sigma_x)_\sharp)), \alpha \cdot \gamma \rangle \mapsto^* \langle \mathrm{Succ}(i_0), \alpha \cdot \beta \rangle$,
there exist $\sigma_y \in E^y$ that satisfy
$\sigma_\sharp \mapsto^* \langle \mathrm{Succ}(\mathrm{body}((\sigma_y)_\sharp)), \alpha \cdot \gamma \rangle \mapsto^* \langle \mathrm{Succ}(i_0), \alpha \cdot \beta \rangle$.

From proposition 11, let x, y be elements in $\mathrm{Parent}(z)$ that satisfy $I^x = I^y$,
for any $\sigma_x \in E^x$ there exists $\sigma_y \in E_y$ whose parse process is same as σ_x for all input.

Example 18. For x, x_1, x_2 in example 11,
let σ_{x1}, σ_{x2} be $\sigma_{x1} = (e_0, x_1), \sigma_{x2} = (e_0, x_2)$ then $E^{x_1} = \{\sigma_{x1}\}, E^{x_2} = \{\sigma_{x2}\}$ hold. As $I^{x_1} = I^{x_2}$ holds,
$x = \sigma_\sharp \mapsto^* \langle \mathrm{Succ}(\mathrm{body}((\sigma_{x1})_\sharp)), abc \cdot e \rangle \mapsto^* \langle \mathrm{Succ}(i_0), abc \cdot ed \rangle$ and
$x = \sigma_\sharp \mapsto^* \langle \mathrm{Succ}(\mathrm{body}((\sigma_{x2})_\sharp)), abc \cdot e \rangle \mapsto^* \langle \mathrm{Succ}(i_0), abc \cdot ed \rangle$ hold for $\sigma \in E^x$.

6 Complexity of Pruned Graph-Structured Stack Parsing

The modified algorithm Pruned Parsing is shown in Algorithm 4 and Algorithm 5. In this pseudo-code the scopes of nesting structures such as loops are specified by indentation. In this algorithm Parent sets are expressed by an array whose index is an e-item and its component is a set of e-items, and `Reduce` is an array whose n-th component (`Reduce[n]`) corresponds to $\mathrm{Reduce}^{(n)}(E_\alpha, a)$ in Algorithm 2. In `Is_prec` of Algorithm 5, duplicate calculation is suppressed by a table `PrecTab`. The initial value of elements of `PrecTab` is 0 and for $x, y \in E$ satisfying body(x)=body(y)=i, the value of `PrecTab`[i, #input(x), #input(y)] is 1 if $x \prec y$ holds, -1 if $x \prec y$ does not hold and 0 otherwise.

I analyze the time complexity of Pruned Parsing.

```
Algorithm 4  (Pruned Graph-Structured Stack Parsing a)
Parse(α)/*returns true if alpha is a word, false otherwise*/
( 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) /* Generate E_{α·a} from E_α and a */
( 9) Reduce_base:=∅; PrecTab[i,♯α + 1, j]:=FALSE;
(10) for x ∈ E_α
(11)     if x ∈ S0eitem(a) then /* For Case 2 */
(12)       E_shift0:=Shift0(x,a);
           Reduce_base:=Reduce_base∪E_shift0;
(13)       for y ∈ E_shift0 /* Parent[y] in Case 2 */
(14)         Parent[y]:=Parent[x]; /* From lemma 1 */
(15)     if x ∈ S1eitem(a) then /* For case 3 */
(16)       E_shift1:=Shift1(x,a);
           Reduce_base:=Reduce_base∪E_shift1;
(17)       for y ∈ E_shift1 /* Parent[y] in Case 3 */
(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) /* case 4, , 5 of Algorithm 3 */
(23) n:=0;Reduce[0]:=D; Reduce[1]:=∅; E_return:=Reduce[0];
(24) loop /* The exit of this loop ((24) − (42)) is (42) */
(25)     for x ∈ Reduce[n]∩Redex
(26)       for x' ∈ Parent[x]
(27)         if x' ∈ R0eitem(x) then /* For case 4*/
(28)           E_reduce0:=Reduce0(x',x);
               Reduce[n+1]:=Reduce[n+1]∪E_reduce0;
(29)           for y ∈ E_reduce0 /* Parent[y] in case 4*/
(30)             Parent[y]:=Parent[x']; /* From lemma 2 */
(31)         if x' ∈ R1eitem(x) then/* For case 5*/
(32)           E_reduce1:=Reduce1(x',x);
               Reduce[n+1]:=Reduce[n+1]∪E_reduce1;
(33)           for y ∈ E_reduce1 /* Parent[y] in case 5*/
(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; /*exit for loop ((24)-(42)) */

Algorithm 5 (Pruned Graph-Structured Stack Parsing 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:= any element of D_x[i];
(52)       for x ∈ D_x[i] /* choose representatives */
(53)         if Is_prec(p_x,x) then
(54)           p_x := x;
(55)       for x ∈ D_x[i] /*error if p_x is NOT representative*/
(56)         if not Is_prec(x,p_x)
(57)           then Error /* This grammar is NOT prunable */
(58)       D_result:=D_result∪ {p_x};
(59) return D_result;

Is_prec(x,y) /* returns true if x ≺ y holds, false otherwise */
(60) i=body(x); l_x := ♯input(x);l_y := ♯input(y);
(61) if PrecTab[i,l_x,l_y]=1 then /* x ≺ y holds */
(62)     return True;
(63) if PrecTab[i,l_x,l_y]= −1 then /* x ≺ y does NOT hold */
(64)     return False;
(65) flag:=True;
(66) for x' ∈Parent[x] (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;
```

Figure 16: Pruned Graph-Structured Stack Parsing

Definition 19 (degenerated). *For $D \subseteq J_G$, D is called degenerated if* $\mathrm{body}(x) = \mathrm{body}(y)$ *for* $\forall x, y \in D$.

From definitions 12 and 14, we have

Lemma 4. *Let $D, D_1, D_2 \subseteq J_G$. If D is degenerated then D is separable.*

Lemmas 5 and 6 state that the time complexity of basic operations such as computation of $\mathrm{Shift0}(x, a)$ and assignment in this algorithm are constant in n.

Lemma 5. *For $x, x' \in J_G$ and $a \in V_T$, the sets* $\mathrm{Shift0}(x, a)$, $\mathrm{Shift1}(x, a)$, $\mathrm{Reduce0}(x', x)$ *and* $\mathrm{Reduce1}(x', x)$ *are degenerated and computed in constant time in n (exactly, $(O(|I_G|))$).*

Proof. Let y be an element of these 4 sets. From Definitions 6 and 9, $\mathrm{body}(y)$ depends only on $\mathrm{body}(x')$ and $\mathrm{body}(x)$. As a body part of an e-item is an element of the finite set I_G, we can compute the body part of these 4 sets for each $\mathrm{body}(x)$ and $\mathrm{body}(x')$ before parsing, and we can get them in constant time in parsing. From Definitions 6 and 9, for each of these 4 sets, all elements of the set have the same input part. Thus all these sets are degenerated.

Since each of the 4 sets is degenerated, the number of its elements is at most $O(|I_G|)$. The body part of each element is obtained before parsing, its input part is identical, so that the set can be decided in $O(|I_G|)$. □

Lemma 6. *For $x \in J_G$ and the sets of e-items $D, D' \subseteq J_G$, if D is separable then both the time complexities of membership $x \in D$ and assignment $D' := D$ are constant in n.*

Lemmas 7, 8 and 9 show that some of the important sets appearing in Pruned Parsing are separable. From Lemma 6 we obtain the following lemma.

Lemma 7. *If D is degenerated then the sets* $\mathrm{Rclosure}(D)$ *and* $\mathrm{Goto}(D, a)$ *in Pruned Parsing are degenerated.*

As the set $\{e_0\}$ is degenerated we obtain the following lemma from Lemma 7.

Lemma 8. E_top *and* E_alpha *in Pruned Parsing is always separable.*

Lemma 9. *For $y \in$* E_top, $\mathrm{Parent}(y)$ *is separable.*

Proof. For $y \in$ E_top, one of the following is satisfied:

a) $y \in \mathrm{Shift0}(x, a)$ for some $x \in$ E_alpha,
b) $y \in \mathrm{Shift1}(x, a)$ for some $x \in$ E_alpha,
c) $y \in \mathrm{Reduce0}(x', x)$ for some $x \in$ E_alpha, $x' \in \mathrm{Parent}(x)$,
d) $y \in \mathrm{Reduce1}(x', x)$ for some $x \in$ E_alpha, $x' \in \mathrm{Parent}(x)$.

For a), $\mathrm{Parent}(y)$ is separable from induction hypothesis, because $\mathrm{Parent}(y) := \mathrm{Parent}(x)$ at (13). For b), this lemma holds because $\mathrm{Parent}(y) = \{x' \mid x' \in \text{E\_alpha} \cap \mathrm{S1eitem}(a), y \in \mathrm{Shift1}(x', a)\}$ from (16) – (20), therefore $\mathrm{Parent}(y) \subseteq$ E_alpha. For c), this lemma holds from the induction hypothesis, because $\mathrm{Parent}(y) := \mathrm{Parent}(x')$ at (30). For d), the element of $\mathrm{Parent}(y)$ is inserted by PruneInsert at (39) that returns a separable set. □

Hereafter we denote the time complexity of a function f as $|f|$.

Proposition 12. *The time complexity of Algorithm 4 is $O(n^2)$ except for the time complexity of* `Prune`.

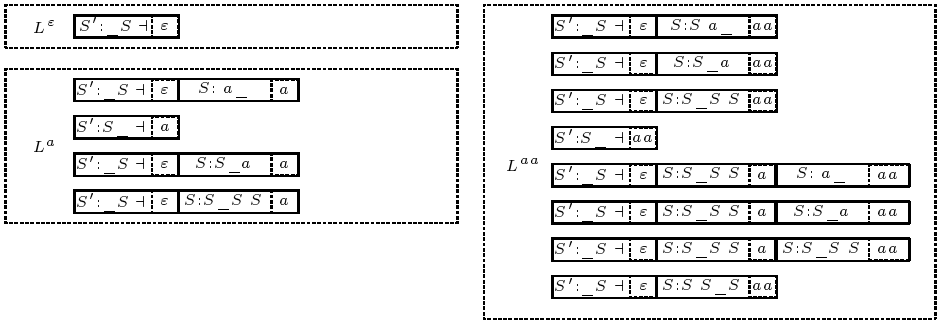


Figure 17: example of L^{α} in $S3$

Proof. The time complexity of the pruned parsing is |`Parse`| that is equal to $n \times$ |`Goto`| from the loop of (2) where n is the length of α. As operations in loops of (10), (13), (17) and (19) are executed less than constant ($|I_G|$) time from lemmas 5 and 9, and the complexity of these operations are constant from lemmas 4 and 7. Therefore, |`Goto`| is equal to |`Rclosure`|. The complexity of operations in the loop (24) is also constant, and the number of execution of these operations in the loop is equal to the number of reductions for each input that is at most n. Therefore, |`Rclosure`| is equal to $O(n)$. From these considerations the time complexity of this algorithm is $O(n^2)$. □

Proposition 13. *The time complexity of* `Prune` *is* $O(n^2)$.

Proof. As `Reduce`$[n]$ in (35) and `Parent`$[z]$ in (36) are separable, the number of `Parent`$[y]$ which is a parameter of `Prune` at (39) is less than $|I_G|$. Loops in (52) and (55) does not depend on n because $|D_x[i]| \leq |I_G|^2$ from $D_x[i] \subseteq \mathrm{Parent}[y](i \in I_G)$. Therefore |`Prune`| = |`Is_prec`| holds.

As `Parent`$[x]$ in (66) and `Parent`$[y]$ in (68) are separable, the time complexity to get `PrecTab`$[i, l_x, l_y]$ at (71) and (72) from known values of `PrecTab` is $O(1)$. Therefore the time complexity to get all values of `PrecTab`$[i, j, k]$ is $O(n^2)$. □

From Proposition 12 and Proposition 13, we have the following theorem.

Theorem 5. *The time complexity of Pruned Parsing is* $O(n^2)$.

7 An Example for a Grammar with Strong Ambiguity

In this section, as an example of this Pruned Graph-Structured Stack Parsing, we show that a grammar $S_m (m \geq 3)$ is parsed in $O(n^2)$. The time complexity for parsing S_m is $O(n^3)$ in [3] and $O(n^{m+1})$ in [11].

Example 19. Let $S3$ be a context free grammar with following productions.

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

$S3$ is a grammar used in [3] as an example grammar that forces the worst-case behavior for algorithms in [11] and [3]. We call a grammar that has same productions with $S3$ except that r^1 has m nonterminals (S) in the right hand side.

Example 20. For $S3$ in example 19, L^{α} for inputs 'aa' and 'aaa' is shown in Fig. 17. In Fig. 17, ⇑reduce and ⇓reduce is omitted.

Stack tops of parse stacks in L^{a^n} and $L^{a^{n+1}}$ for $S3$ are shown in Fig. 18. In Fig. 18, a^n in an input part is denoted as n.

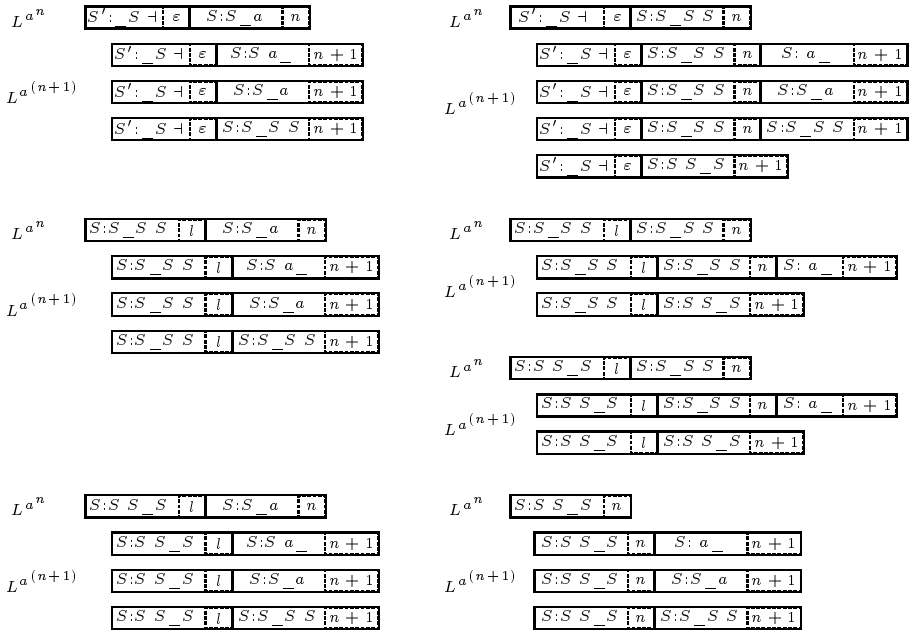


Figure 18: Generation stack top elements of $L^{a^{(n)}}$ from $L^{a^{(n+1)}}$ in $S3$

From Fig. 18, we have

Proposition 14.
Parent$(\langle S : S_a, n+1\rangle) = \{e_0, \langle S : S_S\ S, k\rangle, \langle S : S\ S_S, l\rangle\}(1 \le k \le n, 2 \le l \le n)$,
Parent$(\langle S : S_S\ S, n+1\rangle) = \{e_0, \langle S : S_S\ S, k\rangle, \langle S : S\ S_S, l\rangle\}\ (1 \le k \le n, 2 \le l \le n)$,
Parent$(\langle S : S\ S_S, n+1\rangle) = \{e_0, \langle S : S_S\ S, k\rangle, \langle S : S\ S_S, l\rangle\}(1 \le k \le n-1, 2 \le l \le n-1)$.

From the definition of $\prec$ and above proposition, we have

Proposition 15. *For k and l that satisfy $k < l$,*
$\langle S : S\_a, k\rangle \prec \langle S : S\_a, l\rangle$,
$\langle S : S\_S\ S, k\rangle \prec \langle S : S\_S\ S, l\rangle$,
$\langle S : S\ S\_S, k\rangle \prec \langle S : S\ S\_S, l\rangle$.

From the definition of Prune and above proposition, we have

Proposition 16. *For case of $n \ge 2$*
Prune(Parent$(\langle S : S_a, n+1\rangle)) = \{e_0, \langle S : S_S\ S, n\rangle, \langle S : S\ S_S, n\rangle\}$,
Prune(Parent$(\langle S : S_S\ S, n+1\rangle)) = \{e_0, \langle S : S_S\ S, n\rangle, \langle S : S\ S_S, n\rangle\}$,
Prune(Parent$(\langle S : S\ S_S, n+1\rangle) = \{e_0, \langle S : S_S\ S, n-1\rangle, \langle S : S\ S_S, n-1\rangle\}$.

From above proposition, S_3 is prunable and we have

Proposition 17. *The Time complexity for parsing $S3$ by Pruned Graph-Structured Stack Parsing is $O(n^2)$.*

Similar in proposition 17, S_m $(m > 3)$ is also prunable and the time complexity for parsing S_m by Pruned Graph-Structured Stack Parsing is $O(n^2)$.

Example 21. Relations between $\langle S : S\_S\ S, k\rangle$ $(1 \le k \le 4)$and their parent sets in $S3$ in Structured Graph Stack Parsing and Pruned Structured Graph Stack Parsing are shown in Fig. 19 in which the upper part indicates the case for Structured Graph Stack Parsing and the

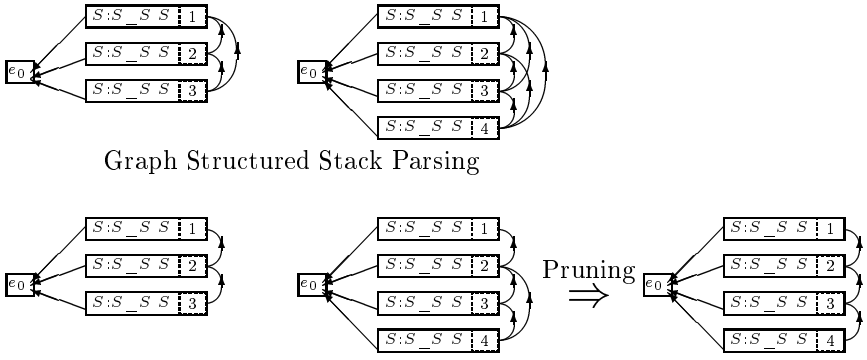


Figure 19: Generation of $\langle S : S\_S\ S, 4\rangle$ in $S3$ (pruned and unpruned version)

lower part indicates the case for Pruned Structured Graph Stack Parsing. In this figure, a^n in an input part is denoted as n and $S : S\_S\ S$ is denoted as i.

The left part shows relations between $\langle i, k\rangle (1 \leq k \leq 3)$ and their parent sets after reading "aaa". In unpruned case (upper left part), $\langle i, j\rangle \in \mathrm{Parent}(\langle i, k\rangle)(1 \leq j < k \leq 3)$, $e_0 \in \mathrm{Parent}(\langle i, k\rangle)$ hold from proposition 15. In pruned case (lower left part), $\langle i, k-1\rangle \in \mathrm{Parent}(\langle i, k\rangle)(1 < k \leq 4)$, $e_0 \in \mathrm{Parent}(\langle i, k\rangle)$ hold from proposition 16.

The middle part shows relations between $\langle i, k\rangle (1 \leq k \leq 4)$ and their parent sets after reading "aaaa". In unpruned case (upper middle part), $\langle i, j\rangle \in \mathrm{Parent}(\langle i, 4\rangle)$ and $(1 \leq j < 4)$, $e_0 \in \mathrm{Parent}(\langle i, 4\rangle)$ hold. In pruned case (lower middle part), $\langle i, j\rangle \in \mathrm{Parent}(\langle i, 4\rangle)$ $(j = 2, j = 3)$ and $e_0 \in \mathrm{Parent}(\langle i, 4\rangle)$ hold.

The right part shows relations between $\langle i, k\rangle (1 \leq k \leq 4)$ and their parent sets after pruning. As $\langle i, 2\rangle \prec \langle i, 3\rangle$ holds, $\langle i, 2\rangle$ is deleted from $\mathrm{Parent}(\langle i, 4\rangle)$ by pruning. Therefore $\langle i, 3\rangle \in \mathrm{Parent}(\langle i, 4\rangle), e_0 \in \mathrm{Parent}$ holds. In this case, $I^{\langle i,k\rangle}(1 \leq k \leq 4)$ is conserved after pruning.

8 Applicable and Inapplicable grammars

In this section, we show examples of grammars that can be parsed or can not be parsed in $O(n^2)$ by Graph Structured Stack Parsing and Pruned Graph Structured Stack Parsing.

From proposition 13, for a grammar G, if $\mathrm{Parent}(x)$ is separable for any $x \in E$, then G can be parsed in $O(n^2)$ by Graph Structured Stack Parsing. From proposition 13 and 14, if G is prunable, then G can be parsed in $O(n^2)$ by Pruned Graph Structured Stack Parsing. Furthermore, similar for proposition 13 and 14, we can prove that for a grammar G, if $|\mathrm{Parent}(x)|$ is constant(not depend on n), then G can be parsed in $O(n^2)$.

The condition for not parsing in $O(n^2)$ by Graph Structured Stack Parsing and Pruned Graph Structured Stack Parsing is that there exists $x \in E$ whose parent set has elements $y_1, \cdots, y_l$ that satisfy following conditions.

1. $l = O(n)$,
2. $y_j \prec y_k$ does not hold for any $j, k (1 \leq j, k \leq l)$.

Example 22. Let G_{RL} be context free grammar with following productions.

$$(r^0)\ S' : S \dashv \quad (r^1)\ S : a\ S \quad (r^2)\ S : X \quad (r^3)\ X : X\ a \quad (r^4)\ X : a$$

Hereafter we denote $S : a\_S$ as i and $X : X\_a$ as i'.
$|\mathrm{Parent}(\langle i', n\rangle)| = O(n)$ holds because $\mathrm{Parent}(\langle i', n\rangle) = \{\langle i, k\rangle\}(2 \leq k \leq n)$ and $\prec$ relation does not hold between $\langle i, k\rangle (1 \leq k \leq n)$ because $\mathrm{Parent}(\langle i, 1\rangle) = \{e_0\}, \mathrm{Parent}(\langle i, n\rangle) = \{\langle i, n-1\rangle\}(2 \leq n)$. G_{RL} can NOT be parsed in $O(n^2)$ by neither Graph Structured Stack Parsing nor Pruned Graph Structured Stack Parsing.

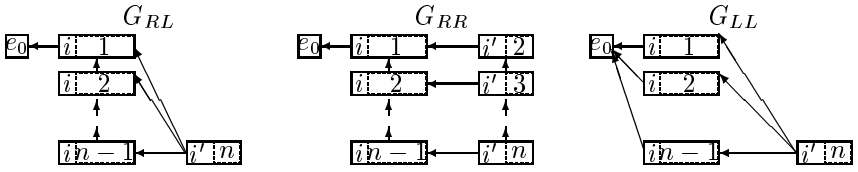


Figure 20: Relations among Parent sets

Example 23. Let G_{RR} be context free grammar with following productions.

$(r^0)\ S' : S \dashv \quad (r^1)\ S : a\ S \quad (r^2)\ S : X \quad (r^3)\ X : a\ X \quad (r^4)\ X : a$

Hereafter we denote $S : a\_S$ as i and $X : a\_X$ as i'.

Parent($\langle i, n\rangle$) and Parent($\langle i', n\rangle$) is separable because Parent($\langle i, 1\rangle$) = $\{e_0\}$, Parent($\langle i, n\rangle$) = $\{\langle i, n-1\rangle\}$, Parent($\langle i', n\rangle$) = $\{\langle i, n-1\rangle, \langle i', n-1\rangle\}(2 \leq n)$ hold. G_{RR} can be parsed in $O(n^2)$ by Graph Structured Stack Parsing.

Example 24. Let G_{LL} be context free grammar with following productions.

$(r^0)\ S' : S \dashv \quad (r^1)\ S : S\ X \quad (r^2)\ S : X \quad (r^3)\ X : X\ a \quad (r^4)\ X : a$

Hereafter we denote $S : a\_S$ as i and $X : X\_a$ as i'.

G_{LL} is prunable because $\langle i, j\rangle \prec \langle i, k\rangle (1 \leq j < k < n)$ hold from Parent($\langle i, n\rangle$) = $\{e_0\}$, Parent($\langle i', n\rangle$) = $\{\langle i, k\rangle\}(1 \leq k < n)$. G_{LL} can NOT be parsed in $O(n^2)$ by Graph Structured Stack Parsing but can be parsed by Pruned Graph Structured Stack Parsing .

Parent relation between $\langle i, j\rangle$, $\langle i', k\rangle$ $(1 \leq j, k \leq n)$ in G_{RL}, G_{RR}, G_{LL} are shown in Fig. 20.

9 Application for Natural Language Processing

In this section, application of this algorithm for natural language processing is stated. In natural language processing, large-scale grammars are prerequisite for parsing a great variety of sentences. Such grammars are derived from a large-scale syntactically annoted corpus because it is difficult to build such grammars by hand [6]. As these grammars are derived automatically, they are highly ambiguous and some input sentence generates many parse trees.

Current approaches to reduce ambiguity are

(A) Use of probabilistic context free grammars [2].

(B) Modifying the language grammar [6].

Probabilistic context free grammar is a context free grammar in which probabilities are assigned to each productions or shift/reduce actions in an LR table. Therefore probabilistic context free grammar can be regarded to assign total order(probabilities) between productions or shift/reduce actions to determine the production or action to be selected. On the other hand, as $\prec$ relation of definition 15 can be regarded as partial order over e-items (to be exact, $\prec$ is preorder because it is reflexive and transitive but not antisymmetric), this algorithm assigns partial order between e-items to be selected. As probabilities are assigned to each productions, it is difficult to assign consistent probabilities over productions. In this algorithm, $\prec$ relation is consistent over e-items because it is defined recursively over e-items.

According to [6], the major ambiguities of a grammar for Japanese are the following.

(B-1) Compound noun structure

(B-2) Adnominal phrase attachment

(B-3) Conjunctive structure

The production for the compound noun structure is "CompoundNoun : CompoundNoun CompoundNoun" and it is modified to "CompoundNoun : Noun CompoundNoun" to reduce ambiguity.

The productions for the adnominal phrase attachment are "NounPhrase : AdnominalPhrase NounPhrase" and "AdnominalPhrase : NounPhrase Particle" . The latter production is modified to "AdnominalPhrase : NounPhrase' Particle" where NounPhrase' is a nonterminal symbol that derives same symbols as NounPhrase except AdnominalPhrase.
The productions for the conjugate structure is "NounPhrase : NounPhrase Particle NounPhrase" and it is modified to "AdnominalPhrase : NounPhrase' Particle".
The situations of above modifications are shown in Fig.21, Fig.22 and Fig.23. These figures are modified from figures in [6].

As these modifications are done manually, it is difficult to make consistent modifications over large-scale grammar. Productions mentioned above are essentially similar to the production(r^1) of S_2, therefore ambiguities are reduced by this algorithm without modifying productions.

10 Conclusion

In this report, I introduce a parsing algorithm that improves the time complexity by pruning the edges of the graph structured stack. Many approaches such as use of tables and memo functions have been proposed to improves the time complexity of parsing. These approaches improve the complexity by suppressing redundant process that are already executed. This approach(pruning) improves the complexity by suppressing redundant process that will be executed. This approach uses stacks and data in stack can be regarded as data processed in future because all data in stack will be processed and popped in future. Therefore it is possible to find redundant process that will be executed in future in this algorithm.

References

[1] Earley, J. : An efficient context free parsing algorithm, Comm. ACM, **13** (1970) 94–102.

[2] 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, **5** (1998) 33–52.

[3] Kipps, J. R. : GLR parsing in time $O(n^3)$, in [10], Chap. 4, 43–59.

[4] Leemakers, R. : A recursive ascent Earley parser, Info. Process. Lett. , **41** (1992) 87–91.

[5] Nederhof, M. J. : Generalized left-corner parsing, Proc. 6th Conf. on European Chapter of the ACL, 1993, 305–314.

[6] Noro, T. , Hashimoto, T. , Tokunaga, T. and Tanaka, H. : Building a Large-Scale Japanese Grammar. Journal of Natural Language Processing, **12** (2005) 3–31. (in Japanese)

[7] Schabes, Y. : Polynomial time and space shift-reduce parsing of arbitrary context-free grammars, Proc. 29th ACL, 1991, 106–113.

[8] Shann, P. : Experiments with GLR and chart parsing, in [10], Chap. 2, 17–34.

[9] Sippu, S. and Soisalon-Soininen, E. : Parsing Theory, Vols. I and II, Springer-Verlag, Berlin, 1988, 1990.

[10] Tomita, M. , ed. : Generalized LR Parsing, Kluwer Academic Publishers, 1991.

[11] Tomita, M. , Ng, S. K. : The generalized LR parsing algorithm, in [10], Chap. 1, pp. 1–16 (1991).

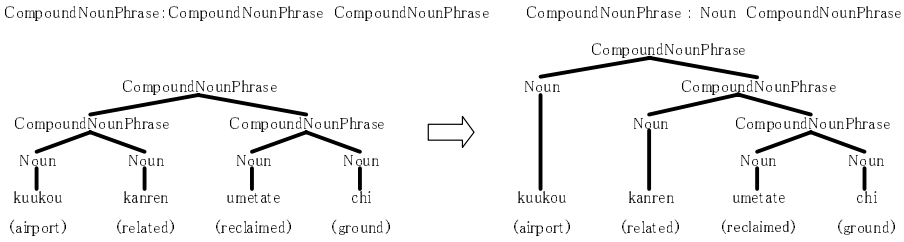


Figure 21: Production modification for Compound Noun Structure

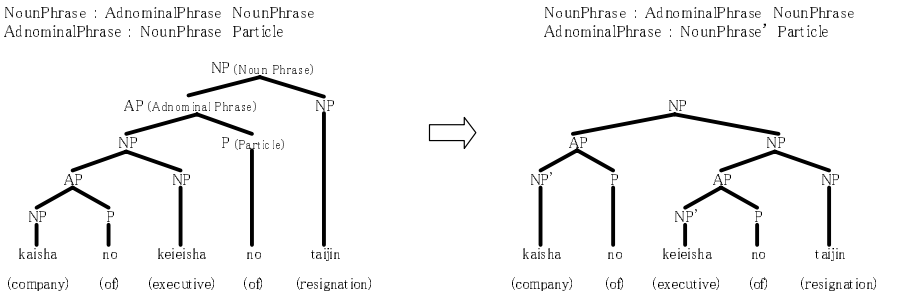


Figure 22: Production modification for Adnominal Phrase Attachment

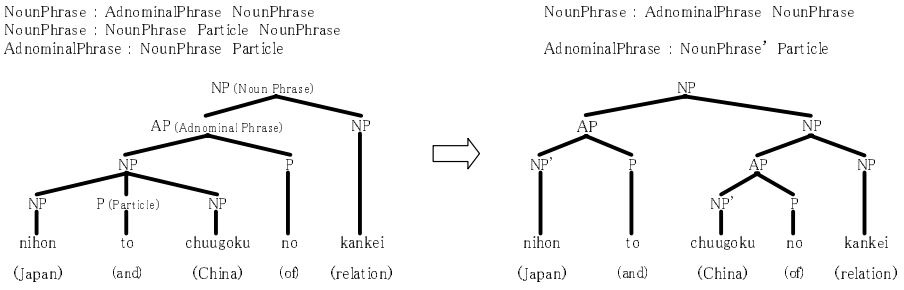


Figure 23: Production modification for Conjunctive Structure

Appendix Proofs of Theorem 1 and Theorem 2

I hereafter abbreviate $\mathrm{Reduce}^{(n)}(L^\alpha, a)$ by $\mathrm{R}^{(n)}(L^\alpha, a)$, and $\mathrm{Reduce}_{(n)}(E^\alpha, a)$ by $\mathrm{R}_{(n)}(E^\alpha, a)$.

Theorem 1. *For $\alpha \in {V_T}^*$, if $\lambda \in L^\alpha$ then $\lambda \in E^\alpha$.*

Proof. From Algorithm 1, $\lambda \in L^\alpha$ is expressed as one of the following five forms:

1. $\alpha = \varepsilon$, $\lambda = e_0$.
2. $\alpha = \alpha' \cdot a$, $\lambda = \mu \cdot y$ for $x \in J_G$, where $\lambda' = \mu \cdot x \in L^{\alpha'}$, $x \in \mathrm{S0eitem}(a)$, $y \in \mathrm{Shift0}(x, a)$.
3. $\alpha = \alpha' \cdot a$, $\lambda = \mu \cdot x \cdot y$ for $x \in J_G$, where $\lambda' = \mu \cdot x \in L^{\alpha'}$, $x \in \mathrm{S1eitem}(a)$, $y \in \mathrm{Shift1}(x, a)$.
4. $\alpha = \alpha' \cdot a$, $\lambda = \mu \cdot y$ for $x, x' \in J_G$ where $\lambda' = \mu \cdot x' \cdot x \in \mathrm{R}^{(n)}(L^{\alpha'}, a) \subseteq E^\alpha$, $n \geq 0$, $x \in \mathrm{Redex}$, $x' \in \mathrm{R0eitem}(x)$, $y \in \mathrm{Reduce0}(x', x)$.
5. $\alpha = \alpha' \cdot a$, $\lambda = \mu \cdot x' \cdot y$ for $x \in J_G$ where $\lambda' = \mu \cdot x' \cdot x \in \mathrm{R}^{(n)}(L^{\alpha'}, a) \subseteq E^\alpha$, $n \geq 0$, $x \in \mathrm{Redex}$, $x' \in \mathrm{R1eitem}(x)$, $y \in \mathrm{Reduce1}(x', x)$.

For Case 1, this theorem holds, because $L^\varepsilon = \{e_0\} = E^\varepsilon$.
For Cases 2, 3, 4 and 5 following relations (A) and (B) hold from the induction hypothesis.

$\mathrm{Parent}(\lambda_1) = \emptyset$, (A)

$\lambda_j \in \mathrm{Parent}(\lambda_{(j+1)})(1 \leq j < \sharp\lambda - 1)$. (B)

Moreover, (C) and (D) hold in cases 2, 3, 4 and 5.

$\lambda_\sharp \in E_\alpha$, (C)

$\lambda_{(\sharp\lambda-1)} \in \mathrm{Parent}(\lambda_\sharp)$. (D)

From (B) and (D)

$\lambda_j \in \mathrm{Parent}(\lambda_{(j+1))}(1 \leq j < \sharp\lambda)$ (E)

holds. By (A), (C) and (E), λ is in E^α, and this theorem is satisfied.

Proofs for (C) and (D) in cases 2, 3, 4 and 5 are as follows.

(2−C) As $\lambda_\sharp = y \in \mathrm{Shift0}(x, a)$ and $x = \lambda'_\sharp \in E_{\alpha'}$, $\lambda_\sharp = y \in E_{\alpha' \cdot a} = E_\alpha$ holds.

(2−D) As $\mu_\sharp = \lambda'_{(\sharp\lambda'-1)} \in \mathrm{Parent}(\lambda'_\sharp) = \mathrm{Parent}(x)$ (by induction hypothesis)
and $\mathrm{Parent}(y) = \mathrm{Parent}(x)$ (from lemma 1),
$\lambda_{(\sharp\lambda-1)} = \mu_\sharp \in \mathrm{Parent}(x) = \mathrm{Parent}(y) = \mathrm{Parent}(\lambda_\sharp)$ holds.

(3−C) As $\lambda_\sharp = y \in \mathrm{Shift1}(x, a)$ and $x = \lambda'_\sharp \in E_{\alpha'}$, $\lambda_\sharp = y \in E_{\alpha' \cdot a} = E_\alpha$ holds.

(3−D) As $x = \lambda'_\sharp \in E_{\alpha'}$ (by induction hypothesis) and
$\mathrm{Parent}(y) = \{z \mid z \in E_{\alpha'}, z \in \mathrm{S1eitem}(a), y \in \mathrm{Shift1}(z, a)\}$ (from case 3 in Algorithm 2),
$\lambda_{(\sharp\lambda-1)} = x \in \mathrm{Parent}(y) = \mathrm{Parent}(\lambda_\sharp)$ holds.

(4−C) As $\lambda_\sharp = y \in \mathrm{Reduce0}(x', x)$ and $x \in E_\alpha$, $x' \in \mathrm{Parent}(x)$, $\lambda_\sharp = y \in E_\alpha$ holds.

(4−D) As $x' = \lambda'_{(\sharp\lambda'-1)} \in \mathrm{Parent}(\lambda'_\sharp) = \mathrm{Parent}(x)$ (by induction hypothesis)
and $\mathrm{Parent}(x) \subseteq \mathrm{Parent}(y)$ (from case 4 of Algorithm 2),
$\lambda_{(\sharp\lambda-1)} = x' \in \mathrm{Parent}(x) \subseteq \mathrm{Parent}(y) = \mathrm{Parent}(\lambda_\sharp)$ holds.

(5−C) As $\lambda_{\sharp\lambda} = y \in \mathrm{Reduce1}(x', x)$ *and* $x \in E_\alpha$, $x' \in \mathrm{Parent}(x)$ $\lambda_\sharp = y \in E_\alpha$ holds.

(5−D) As $x' = \lambda'_{(\sharp\lambda'-1)} \in \mathrm{Parent}(\lambda'_{\sharp\lambda'}) = \mathrm{Parent}(x)$ (by induction hypothesis) and
$x' \in \mathrm{Parent}(y)$ (from case 5 of Algorithm 2 and $x = \lambda'_\sharp \in E_\alpha \subseteq \mathrm{R}_{(n)}(E^\alpha, a)$),
$\lambda_{(\sharp\lambda-1)} = x' \in \mathrm{Parent}(y) = \mathrm{Parent}(\lambda_\sharp)$ holds. □

Theorem 2. *For $\alpha \in {V_T}^*$, if $\sigma \in E^\alpha$ then $\sigma \in L^\alpha$.*

Proof. From Algorithm 2, $\sigma \in E^\alpha$ is expressed as one of the following five forms:

1. $\alpha = \varepsilon, \sigma = e_0$.

2. $\alpha = \alpha' \cdot a,\ \sigma = \tau \cdot y$ for $x \in J_G$ where $\tau \cdot x \in E^{\alpha'},\ x \in \mathrm{S0eitem}(a),\ y \in \mathrm{Shift0}(x, a)$.

3. $\alpha = \alpha' \cdot a,\ \sigma = \tau \cdot x \cdot y$ where $\tau \cdot x \in E^{\alpha'},\ x \in \mathrm{S1eitem}(a),\ y \in \mathrm{Shift1}(x, a)$.

4. $\alpha = \alpha' \cdot a,\ \sigma = \tau \cdot y$ for $x',\ x \in J_G$ where $\tau \cdot x' \cdot x \in E^{\alpha'}$,
   $n \geq 0,\ x \in \mathrm{Redex},\ x' \in \mathrm{R}_{(n)}(E^{\alpha'}, a),\ x \in \mathrm{R0eitem}(x), y \in \mathrm{Reduce0}(x', x)$.

5. $\alpha = \alpha' \cdot a,\ \sigma = \tau \cdot x' \cdot y$ for $x \in J_G$ where $\tau \cdot x' \cdot x \in E^{\alpha'}$,
   $n \geq 0,\ x \in \mathrm{Redex},\ x' \in \mathrm{R}_{(n)}(E^{\alpha'}, a),\ x \in \mathrm{R1eitem}(x),\ y \in \mathrm{Reduce1}(x', x)$.

For Case 1, this theorem holds, since $E^{\varepsilon} = \{e_0\} = L^{\varepsilon}$.
For case 2, as $\tau \cdot x \in L^{\alpha'}$ (by $\tau \cdot x \in E^{\alpha'}$ and induction hypothesis) and $x \in \mathrm{S0eitem}(a),\ y \in \mathrm{Shift0}(x, a)$ hold, $\sigma = \tau \cdot y \in L^{\alpha' \cdot a} = L^{\alpha}$ is satisfied from case 2 of Algorithm2.
For case 3, $\sigma \in L^{\alpha' \cdot a} = L^{\alpha}$ is satisfied similar to case 2.

For case 4, as $\tau \cdot x' \cdot x \in L^{\alpha'}$ (by $\tau \cdot x' \cdot x \in E^{\alpha'}$ and induction hypothesis) and $x \in \mathrm{Redex},\ x' \in \mathrm{R}_{(n)}(E^{\alpha'}, a),\ x \in \mathrm{R0eitem}(x), y \in \mathrm{Reduce0}(x', x)$ hold, $\sigma = \tau \cdot y \in L^{\alpha}$ is satisfied from case 4 of Algorithm2.
For case 5, $\sigma \in L^{\alpha}$ is satisfied similar to case 4. □