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

Shin-ichi Morimoto

Jan 2007, C–240
An Efficient Parsing for Highly Ambiguous Context-Free Grammars Based on Pruning

Shin-ichi Morimoto
NEC Aerospace Systems, Ltd.
smorimoto@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 [3]. 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
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 $r \in P$, $|r|$ 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 $r_0 : r_1 r_2 \ldots r_{|r|}$.

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

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

\[
(r^0) S' : S \vdash (r^1) S : X Y \alpha (r^2) X : a (r^3) X : a b (r^4) Y : Z \epsilon (r^5) Z : c (r^6) Z : b c
\]

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

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

**Definition 2 (Item).** For $r \in P$ and a nonnegative integer $n$ ($0 \leq n \leq |r|$), an item is a pair $(r, n)$, and the set of all items is denoted by $I_G$. For a production $r$ of the form $r_0 : r_1 r_2 \ldots r_{|r|}$,
an item \( (r, n) \) can be expressed as \( r_0 : r_1 r_2 \ldots r_{n-1} r_n r_{n+1} \ldots r_{nr} \). In particular, the item \( (S' : S \downarrow) \) 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 \( (t, \alpha) \), and the set of all e-items are denoted by \( J_G \). For \( x = (t, \alpha) \in J_G \), we define body(\( x \)) and input(\( x \)) by body(\( x \)) = \( t \) and input(\( x \)) = \( \alpha \), respectively. In particular, the e-item \( (i_0, \varepsilon) \) is denoted by \( e_0 \).

For a sequence \( \eta \), \( \# \eta \) denotes the length of \( \eta \), \( \eta_j \) the \( j \)-th element of \( \eta \), and \( \eta_i \) the last element of \( \eta \).

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

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

**Example 2.** Let \( G_1 \) be a CFG in example 1. We have \( (r^1, 1) \downarrow r^4, (r^1, 1) \downarrow^+_\varepsilon r^4, \) and \( (r^1, 1) \downarrow^+ r^6 \). We recall that \( r^1 = S : XYd, r^4 = Y : Ze, \) and \( r^6 = Z : bc \). The situation for \( (r^1, 1) \downarrow^+_\varepsilon r^4 \) and \( (r^1, 1) \downarrow^+ r^6 \) are shown in Fig. 3.

**Definition 5 (S0item\(_G\), S1item\(_G\)).** For a CFG \( G \), \( a \in V_T \), we define

\[ S0item_G(a) = \{(r, n, \alpha) | r_{n+1} = a \text{ for some } r \in P, \alpha \in V_T^*, 0 \leq n \leq \# r\} \]

\[ S1item_G(a) = \{(r, n, \alpha) | (r, n) \downarrow^+ r' \text{ for some } r, r' \in P, 0 \leq n \leq \# r\} \]

Hereafter we fix a CFG \( G \) and we often omit \( G \) in notations like \( S0item \).

**Definition 6 (Shift0, Shift1).** For \( a \in V_T \), \( x_0 = (\langle r^0, n_0 \rangle, \alpha_0) \in S0item(a) \), \( x_1 = (\langle r^1, n_1 \rangle, \alpha_1) \in S1item(a) \), we define Shift0(\( x_0, a \)) and Shift1(\( x_1, a \)) \( \subseteq J_G \) by

\[
\text{Shift0}(x_0, a) = \{\langle r^0, n_0 + 1 \rangle, \alpha_0 \cdot a\},
\]

\[
\text{Shift1}(x_1, a) = \{\langle r^1, 1 \rangle, \alpha_1 \cdot a \} | \langle r^1, n_1 \rangle \downarrow^+_a r'\}.
\]
Example 3. Let $G_1$ be a CFG in example 1, 
\[
\langle X : \alpha \rangle \in \text{S0eitem}(\alpha), \quad \text{S0eitem}(\langle X : a \ b, a \rangle, b) = \{ \langle X : a \ b, a \rangle \}
\]
\[
\langle S : X \ Y \ d, a \rangle \in \text{S1eitem}(b), \quad \text{S1eitem}(\langle S : X \ Y \ d, a \rangle, b) = \{ \langle X : b \ c, a \rangle \}
\]
The situation of this example is shown in Fig. 4.

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

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

**Definition 9 (Reduce0, Reduce1).** For $x = \langle \langle r, \sharp r \rangle, \alpha \rangle$ and $x' = \langle \langle r^0, n_0', \alpha' \rangle \in \text{R0eitem}(x)$, we define $\text{Reduce0}(x', x) \subseteq J_G$ by
\[
\text{Reduce0}(x', x) = \{ \langle \langle r^0, n_0' + 1 \rangle, \alpha \rangle \}.
\]
For $x = \langle \langle r, \sharp r \rangle, \alpha \rangle$ and $x' = \langle \langle r^1, n_1', \alpha' \rangle \in \text{R1eitem}(x)$, we define $\text{Reduce1}(x', x) \subseteq J_G$ by
\[
\text{Reduce1}(x', x) = \{ \langle \langle r^1, n_1' \downarrow \rangle, \alpha \rangle \mid \text{r'}_{n_1'} \downarrow \text{ for some } r'' \in P \}.
\]

Example 4. Let $G_1$ be a CFG in example 1, $x = \langle Y : Z \ e, abc \rangle$, $x'_0 = \langle S : X \ Y \ d, a \rangle$
\[
x \in \text{Redex}, \quad x'_0 \in \text{R0eitem}(x), \quad \text{Reduce0}(x'_0, x) = \{ \langle S : X \ Y \ d, a \rangle \}.
\]
For $x = \langle Z : b \ c, abc \rangle$, $x'_1 = \langle S : X \ Y \ d, a \rangle$
\[
x \in \text{Redex}, \quad x'_1 \in \text{R1eitem}(x), \quad \text{Reduce1}(x'_1, x) = \{ \langle Y : Z \ e, abc \rangle \}.
\]
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 $\alpha$ and an integer $n$, which can be seen as a parsing for $\alpha$. 


Example 5. Let $G_1$ 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. $L^e = \{e_0\}$.

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

2. For $\mu \cdot x \in L^\alpha$, $\mu \cdot y \in L^{\alpha, a}$ where $x \in S_0 \text{item}(a)$ and $y \in \text{Shift0}(x, a)$.

3. For $\mu \cdot x \in L^\alpha$, $\mu \cdot x \cdot y \in L^{\alpha, a}$ where $x \in S_1 \text{item}(a)$ and $y \in \text{Shift1}(x, a)$.

In the following case 4 and case 5 we define elements of $L^{\alpha, a}$ generated by a reduce action from elements of $L^{\alpha, a}$ by induction on $n$. The induction base is defined by

$$\text{Reduce}^{(0)}(L^\alpha, a) = \text{Shift0}^{(0)}(L^\alpha, a) \cup \text{Shift1}^{(0)}(L^\alpha, a)$$

where $\text{Shift0}^{(0)}(L^\alpha, a)$ is a set of sequences of e-items $(\mu \cdot y)$ defined in case 2 and $\text{Shift1}^{(0)}(L^\alpha, a)$ is a set of sequences of e-items $(\mu \cdot x \cdot y)$ defined in case 3.

4. For $\mu \cdot x' \cdot x \in \text{Reduce}^{(n)}(L^\alpha, a)$, $\mu \cdot y \in L^{\alpha, a}$ where $x \in \text{Redex}$, $x' \in R_0 \text{item}(x)$ and $y \in \text{Reduce0}(x', x)$.

5. For $\mu \cdot x' \cdot x \in \text{Reduce}^{(n)}(L^\alpha, a)$, $\mu \cdot x' \cdot y \in L^{\alpha, a}$ where $x \in \text{Redex}$, $x' \in R_1 \text{item}(x)$ and $y \in \text{Reduce1}(x', x)$.

Case 4 and case 5 generate $\text{Reduce}^{(n+1)}(L^\alpha, a)$ from $\text{Reduce}^{(n)}(L^\alpha, a)$ where $n$ corresponds to the times of reduce action. $\text{Reduce}^{(n+1)}(L^\alpha, a)$ is defined by

$$\text{Reduce}^{(n+1)}(L^\alpha, a) = \text{Reduce}^{(n+1)}(L^\alpha, a) \cup \text{Reduce}^{(n+1)}(L^\alpha, a)$$

where $\text{Reduce}^{(n+1)}(L^\alpha, a)$ is a set of sequences of the e-items $(\mu \cdot y)$ defined in case 4 and $\text{Reduce}^{(n+1)}(L^\alpha, a)$ is a set of sequences of the e-items $(\mu \cdot x' \cdot y)$ defined in case 5.

Example 6. Let $G_1$ be a CFG in example 1, $L^\alpha$ of $G_1$ for an input string “abcd” and its substrings are shown in Fig. 7. Each element of $L^\alpha$ is a sequence of e-items and corresponds to a parse stack in [5].

Let $\mu^2 = (e_0), x^2 = \langle X : a \cdot b, a \rangle$, and $y^2 = \langle X : ab \cdot ab \rangle$.

As $\mu^2 \cdot x^2 \in L^\alpha$, $x^2 \in S_0 \text{item}(b)$ and $y^2 \in \text{Shift0}_E(x^2, b)$,

Let $\mu^3 = (e_0), x^3 = \langle S : X \cdot Y, a, a \rangle$, and $y^3 = \langle Z : b \cdot c, ab \rangle$.

As $\mu^3 \cdot x^3, y^3$ satisfy $\mu^3 \cdot x^3 \in L^\alpha$, $x^3 \in S_1 \text{item}(b)$, $y^3 \in \text{Shift1}_E(x^3, b)$,

$\mu^3 \cdot x^3 \cdot y^3 \in L^{ab}$ by case 3 of the stack set parse algorithm.

Figure 6: parse stacks in $L^{ab}$ and corresponding parse trees
Figure 7: examples of $L^o$ for an input “abcde”

generation by case 2

$x_2 = \langle Z : b e_\text{ab} \rangle \in \text{Shift}(c, e)
\quad y_2 = \langle Z : \text{abc} \rangle \in \text{Shift}(x_2, c)

generation by case 4

$x_4 = \langle Y : Z \text{abc} \rangle \in \text{Redex}
\quad x_4 = \langle S : X \text{abc} \rangle \in \text{Redex}
\quad y_4 = \langle S : \text{abc} \rangle \in \text{Reduce}(x_4)

Figure 8: Example of parse stack generation

Let $\mu_4 = (e_0), \quad x_4 = \langle S : X \text{abc} \rangle, \quad x_4 = \langle Y : Z \text{abc} \rangle.
\quad y_4 = \langle S : \text{abc} \rangle \in \text{Reduce}(x_4)
\quad \mu_4 \cdot y_4 \in L_{abc}^e$ by case 4 of the stack set parse algorithm.

Let $\mu_5 = (e_0), \quad x_5 = \langle S : X \text{abc} \rangle, \quad x_5 = \langle Y : Z \text{abc} \rangle.
\quad y_5 = \langle S : \text{abc} \rangle \in \text{Reduce}(x_4)
\quad \mu_5 \cdot x_5 \in L_{abc}^e$ by case 4 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 $L_f$ as the initial state and construct $L_{o-a}$ from $L^o$ if the current input is $a$ 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 Parent($x$).

Algorithm 2 (Graph-structured stack parsing).

For $a \in V_f$, we define $E_a \subseteq J$, and Parent($x$) for $x \in E_a$, inductively on elements of a string $a$ and a positive integer $n$, as follows:

1. $E_e = \{e_0\}, \text{Parent}(e_0) = \emptyset$.

In the following case 2 and case 3 we define elements of $E_{a,a}$ generated by a shift action from elements of $E_a$. 

2. For \( x \in E_0 \), \( y \in E_{a-a} \) where \( x \in S0eitem(a), y \in \text{Shift}_0(x, a) \).
   For this \( y \), Parent(\( y \)) is defined by
   \[
   \text{Parent}(y) = \{ w \mid w \in \text{Parent}(z) \text{ for } z \text{ where } z \in E_0, z \in S0eitem(a), y \in \text{Shift}_0(z, a) \}\]

3. For \( x \in E_0 \), \( y \in E_{a-a} \) where \( x \) \in Sleitem(a), \( y \) \in \text{Shift}_1(x, a) \).
   For this \( y \), Parent(\( y \)) is defined by
   \[
   \text{Parent}(y) = \{ z \mid z \in E_0, z \in \text{Sleitem}(a), y \in \text{Shift}_1(z, a) \}\]

In the following case 4 and case 5 we define elements of \( E_{a-a} \) generated by a reduce action from elements of \( E_{a-a} \) by induction on \( n \). The induction base is defined by
   \[
   \text{Reduce}_0(E_0, a) = \text{Shift}_0(E_0, a) \cup \text{Shift}_1(E_0, a)
   \]
where \( \text{Shift}_0(E_0, a) \) is a set of \( e \)-items (\( y \)) defined in case 2 and \( \text{Shift}_1(E_0, a) \) is a set of \( e \)-items (\( y \)) defined in case 3.

4. For \( x \in \text{Reduce}_n(E_0, a), y \in E_{a-a} \) where \( x' \) \in Parent(x), \( x \) \in Redex, \( x' \) \in R0eitem(x), \( y \) \in \text{Reduce}(x', x).
   For this \( y \), Parent(\( y \)) is defined by
   \[
   \text{Parent}(y) = \{ w \mid w \in \text{Parent}(z') \text{ for } z, z' \text{ where } z \in \text{Reduce}_n(E_0, a),
   z' \in \text{Parent}(z), z \in \text{Redex}, z' \in \text{R0eitem}(z), y \in \text{Reduce}(z', z) \}\]

5. For \( x \in \text{Reduce}_n(E_0, a), y \in E_{a-a} \) where \( x' \) \in Parent(x), \( x \) \in Redex, \( x' \) \in R1eitem(x), \( y \) \in \text{Reduce}(x', x).
   For this \( y \), Parent(\( y \)) is defined by
   \[
   \text{Parent}(y) = \{ z' \mid \text{for } z' \text{ where } z \in \text{Reduce}_n(E_0, a),
   z' \in \text{Parent}(z), z \in \text{Redex}, z' \in \text{R1eitem}(z), y \in \text{Reduce}(z', z) \}\]

Case 4 and case 5 generate \( \text{Reduce}_{n+1}(E_0, a) \) from \( \text{Reduce}_n(E_0, a) \) where \( n \) corresponds to the times of reduce action. \( \text{Reduce}_{n+1}(E_0, a) \) is defined by
   \[
   \text{Reduce}_{n+1}(E_0, a) = \text{Reduce}_0(E_0, a) \cup \text{Reduce}_1(E_0, a)
   \]
where \( \text{Reduce}_0(E_0, a) \) is a set of \( e \)-items (\( y \)) defined in case 4 and \( \text{Reduce}_1(E_0, a) \) 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 “abcd” is shown in Fig. 9, where elements of Parent(\( x \)) are denoted by an arrow from \( x \) to each element when Parent(\( x \)) \( \neq \{ e_0 \} \). For example, there is an arrow from \( x = (Y : Z - e, abcd) \) to \( \langle S : X \_ Y d, ab \rangle \) since Parent(\( x \)) = \{ \langle S : X \_ Y d, a \rangle, \langle S : X \_ Y d, ab \rangle \} and there is no arrow from \( x' = \langle S : X \_ Y d, ab \rangle \) to \( e_0 \) since Parent(\( x' \)) = \{ \langle S : X \_ Y d, ab \rangle \}.

Let \( x^2 = \langle X : a \_ b, a \rangle \), \( y^2 = \langle X : a \_ b, ab \rangle \).
As \( x^2, y^2 \) satisfy \( x^2 \in E_{ab}, x^2 \in S0eitem(b), y^2 \in \text{Shift}_0(x^2, b), y^2 \in E_{ab} \) and Parent(\( y^2 \)) = 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 \( x^3 \in E_{ab}, x^3 \in S1eitem(b), y^3 \in \text{Shift}_1(x^3, b), y^3 \in E_{ab} \) and Parent(\( y^3 \)) = \{ x_3 \} by case 3 of the graph structured stack parse algorithm.

Let \( x^4 = \langle S : X \_ Y d, a \rangle, x^4 = (Y : Z \_ e, abcd), y^4 = \langle S : X \_ Y d, ab \rangle \).
As \( x^4, y^4 \) satisfy \( x^4 \in \text{Reduce}_0(E_{abc}, e), x^4 \in \text{Redex}, x^4 \in \text{R0eitem}(x^4), y^4 \in \text{Reduce}_0(x^4, x^4), y^4 \in E_{abc} \) and Parent(\( y^4 \)) = Parent(\( x^4 \)) = \{ e_0 \} by case 4 of the graph structured stack parse algorithm.

Let \( x^5 = \langle S : X \_ Y d, a \rangle, x^5 = (Z : b \_ c, abcd), y^5 = (Y : Z \_ e, abcd) \).
As \( x^5, y^5 \) satisfy \( x^5 \in \text{Reduce}_0(E_{ab}, e), x^5 \in \text{Redex}, x^5 \in \text{R1eitem}(x^5), y^5 \in \text{Reduce}_1(x^5, x^5), y^5 \in \text{Reduce}_1(x^5, x^5) \).
Figure 9: Graph structured stack for the input “abcd”

Generation by case 2

\[
\begin{align*}
& x_2 \\
& S : X \_ Y \_ d, \_ a, \_ b, \_ c \_ \\
& \text{Reduce}(x_2, c) = \{y_2\} \\
& \text{Parent}(y_2) = \text{Parent}(x_2) = \{x_2\}
\end{align*}
\]

\[x_3 \in S\text{Item}(c), \text{Parent}(x_2) = \{x_2\}\]
\[\text{Shift0}(x_2, c) = \{y_2\}\]
\[\text{Parent}(y_2) = \text{Parent}(x_2) = \{x_2\}\]

Generation by case 3

\[
\begin{align*}
& x_3 \\
& S : X \_ Y \_ d, \_ a, \_ b, \_ c \_ \\
& \text{Reduce}(x_3, c) = \{y_3\} \\
& \text{Parent}(y_3) = \text{Parent}(x_3) = \{x_3\}
\end{align*}
\]

\[x_3 \in S\text{Item}(c), \text{Parent}(x_3) = \{e_0\}\]
\[\text{Shift1}(x_3,c) = \{y_3\}, \text{Parent}(y_3) = \{x_3\}\]

Generation by case 4

\[
\begin{align*}
& x'_4 \\
& S : X \_ Y \_ d, \_ a, \_ b, \_ c \_ \\
& \text{Reduce0}(x'_4, c_4) = \text{Reduce0}(x'_4, x_4) = \{y_4\} \\
& \text{Parent}(x'_4) = \text{Parent}(x_4) = \{e_0\} \\
& \text{Parent}(y_4) = \text{Parent}(x'_4) = \text{Parent}(x_4) = \{e_0\}
\end{align*}
\]

\[x'_4, x'_4 \in R\text{Item}(x_4), \text{Parent}(x_4) = \{x'_4, x'_4\}\]
\[x'_5 \in R\text{Item}(x_5), \text{Parent}(x_5) = \{x'_5\}\]
\[\text{Reduce0}(x'_4, x_4) = \text{Reduce0}(x'_4, x_4) = \{y_4\}\]
\[\text{Parent}(x'_4) = \text{Parent}(x'_4) = \text{Parent}(x_4) = \{e_0\}\]
\[\text{Parent}(y_4) = \text{Parent}(x'_4) = \text{Parent}(x'_4) = \{e_0\}\]

Generation by case 5

\[
\begin{align*}
& x'_5 \\
& S : X \_ Y \_ d, \_ a, \_ b, \_ c \_ \\
& \text{Reduce}(x'_5, c_5) = \text{Reduce}(x'_5, x_5) = \{y_5\} \\
& \text{Parent}(y_5) = \text{Parent}(x'_5) = \text{Parent}(x'_5) = \{e_0\}
\end{align*}
\]

\[x'_5, x'_5 \in R\text{Item}(x_5), \text{Parent}(x_5) = \{x'_5, x'_5\}\]
\[x'_5 \in R\text{Item}(x_5), \text{Parent}(x_5) = \{x'_5\}\]
\[\text{Reduce}(x'_5, x_5) = \text{Reduce}(x'_5, x_5) = \{y_5\}\]
\[\text{Parent}(y_5) = \text{Parent}(x'_5) = \text{Parent}(x'_5) = \{e_0\}\]

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 \(x \in J_G\), if \(x \in S\text{Item}(a)\) and \(y \in \text{Shift0}(x, a)\), then \(\text{Parent}(y) = \text{Parent}(x)\).

**Proof.** If \(y\) is expressed as \(\langle \langle r, n, \alpha \cdot a \rangle, x \rangle\), \(x\) is expressed as \(\langle \langle r, n - 1, \alpha \rangle, x \rangle\) 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. \[\square\]

**Definition 10 (E^0).** For \(\alpha \in V_T^*, E^0 \subseteq J_G^*\) is defined by

\[E^0 = \{\sigma | \text{Parent}(\sigma_j) = \emptyset, \sigma_j \in \text{Parent}(\sigma_{j+1})(1 \leq j < \#\sigma), \sigma, \sigma_j \in E_\alpha\}\].

**Example 8.** For \(G1\) of example 1, \(E_{ab} = \{\sigma^1, \sigma^2, \sigma^3\}\) where

\[
\begin{align*}
\sigma^1 &= \langle a^1, \sigma^1_2 \rangle = \langle e_0, \langle X : a \_ b \_ , ab\rangle \rangle, \\
\sigma^2 &= \langle \sigma^1_2, \sigma^2_2 \rangle = \langle e_0, \langle S : X \_ Y \_ d, ab\rangle \rangle, \\
\sigma^3 &= \langle \sigma^3_1, \sigma^3_2, \sigma^3_3 \rangle = \langle e_0, \langle S : X \_ Y \_ d, a, Z : b \_ c, ab\rangle \rangle.
\end{align*}
\]

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 \(e_0\).
\[ \sigma_1^1 = \sigma_2^1 = \sigma_3^1 \]
\[ X \rightarrow_1 X \]
\[ X \rightarrow_2 X \]
\[ X \rightarrow_3 X \]
\[ X \rightarrow_4 X \]
\[ X \rightarrow_5 X \]

Figure 11: Elements of \( \sigma_1^1, \sigma_2^1, \sigma_3^1 \)

We have the basic theorems about \( L^\alpha \) and \( E^\alpha \). They are proved inductively based on definitions of case (1) to case 5 in Algorithm 1 and Algorithm 2.

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

1. \( x \in S\text{ ITEM}(a) \) and \( y \in \text{ Shift0}(x, a) \),
2. \( x \in S\text{ ITEM}(a) \) and \( y \in \text{ Shift1}(x, a) \),
3. \( x \in \text{ Redex} \) and \( y \in \text{ Reduce0}(x', x) \) for some \( x' \in \text{ R0item}(x) \),
4. \( x \in \text{ Redex} \) and \( y \in \text{ Reduce1}(x', x) \) for some \( x' \in \text{ R1item}(x) \).

The transitive closure of \( \rightarrow \) is denoted by \( \rightarrow^* \).

**Example 9.** For \( G_1 \) in example 1, \( \langle Z : b \_c, ab \rangle \rightarrow \langle Z : bc \_abc \rangle \) 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, \( \langle S : X \ Y d, a \rangle \rightarrow^* \langle S : X Y \_ d, abc \rangle \) holds because

\[ \langle S' : S \_ \_ a, e \rangle \rightarrow \langle X : a \_ a, a \rangle \rightarrow \langle S : X \ Y d, a \rangle \rightarrow \langle Z : b \_ c, ab \rangle \rightarrow \langle Z : bc \_abc \rangle \rightarrow \langle Y : Z \_ c, abc \rangle \rightarrow \langle Y : Ze \_abc \rangle \rightarrow \langle S : X Y \_ d, abc \rangle \rightarrow \langle S' : S \_ \_ a, abc d \rangle \]

By definitions of \( \rightarrow^* \) and \( \text{Parent}(x) \), we easily have:

**Proposition 1.** If \( y \in \text{ Parent}(x) \) then \( y \rightarrow^* x \).

**Definition 12 (separable).** For a set of e-items \( D \subseteq J_G \), \( D \) is called separable if \( \text{body}(x) = \text{body}(y) \) 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 \( x \in \text{ Redex}, x' \in \text{ R0item}(x) \cap \text{ Parent}(x), y \in \text{ Reduce0}(x', x) \), if \( \text{ Parent}(x) \) is separable then \( \text{ Parent}(y) = \text{ Parent}(x') \).

**Proof.** If there exists \( z' \in \text{ R0item}(x) \cap \text{ Parent}(x) \) satisfying \( y \in \text{ Reduce0}(x', x) \), \( \text{body}(x') = \text{body}(z') \) holds from \( y \in \text{ Reduce0}(x', x) \) and \( y \in \text{ Reduce0}(z', x) \). Therefore \( x' = z' \) holds if \( \text{ Parent}(x) \) is separable. It means that \( x' \) that satisfy \( y \in \text{ Reduce0}(x', x) \) is unique and \( \text{ Parent}(y) = \text{ Parent}(x') \) holds from definition of \( \text{ Parent}(y) \) in case 4of Algorithm 2. \( \square \)
Unfortunately, all parent sets are not separable.

**Example 10.** As is seen from Fig. 2, Parent((\(Y : Z_e, \text{abc}\))) 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 \(x_1, x_2 \in \text{Parent}(x)\) for an e-item \(x\), and always assume that \(\text{body}(x_1) = \text{body}(x_2) = \langle r, n \rangle\) and \(\text{input}(x_1) \neq \text{input}(x_2)\). If \(x \rightarrow^* \langle \langle r, n + 1, \alpha \rangle \rangle\) for some \(\alpha \in V_r^*\), then \(x_1 \rightarrow^* \langle \langle r, n + 1, \alpha \rangle \rangle\) and \(x_2 \rightarrow^* \langle \langle r, n + 1, \alpha \rangle \rangle\) are satisfied.

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

**Example 11.** Let \(x_1, x_2, x\) and \(x'\) be \(x_1 = \langle S : X_Yd, a \rangle, x_2 = \langle S : X_Yd, ab \rangle, x = \langle Y : Z_e, \text{abc} \rangle\) and \(x' = \langle S : XY_d, ab \rangle\). In this case \(x_1 \rightarrow^* x'\) and \(x_2 \rightarrow^* x'\) holds since \(x_1, x_2 \in \text{Parent}(x)\) and \(x \rightarrow^* x'\).

This situation is shown in Fig. 12, in which the upper part indicates a parse process corresponding to \(x_1 \rightarrow^* x \rightarrow^* x'\) and the lower part indicates a parse process corresponding to \(x_2 \rightarrow^* x \rightarrow^* x'\). 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: Z_e”, 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 \(E = \bigcup_{\alpha \in V_r^*} E_\alpha\).
Definition 14 \((I^o, E^x, I^x)\). For \(\alpha \in V_I^*, x \in E_\alpha\), we define
\[
I^o = \{ \rho \mid \exists \sigma \in E^o \text{ where } \exists \rho = \exists \sigma, \rho_j = \text{body}(\sigma_j) \quad (1 \leq j \leq \exists \rho) \}\.
\]
\[
E^x = \{ \sigma \mid \sigma \in E^o, \sigma_1 = x \}\.
\]
\[
I^x = \{ \rho \mid \rho \in I^o, \rho_i = x \}.
\]

Elements of \(I^o\) are body parts of elements of \(E^o\) and \(E^x\) is a subset of \(E^o\) whose element \(\sigma\) satisfies \(\sigma_1 = x\).

Example 12. For \(G_1\) in example 1, \(I^{abc} = \{ \rho^1, \rho^2, \rho^3 \} \) where
\[
\rho^1 = (\rho^1_0, \rho^1_1, \rho^1_2, \rho^1_3) = (i_0, S : X \cdot Y d, Z : bc_\cdot).
\]
\[
\rho^2 = (\rho^2_0, \rho^2_1, \rho^2_2, \rho^2_3) = (i_0, S : X \cdot Y d, Y : Z \cdot e),
\]
\[
\rho^3 = (\rho^3_0, \rho^3_1, \rho^3_2, \rho^3_3) = (i_0, S : X \cdot Y d, Z : c_\cdot).
\]
For \(x = \langle Y : Z \cdot e, abc \rangle\), \(I^x = \{ \rho^2 \}\).

From the definition of \(I^x\), we have

Lemma 3. For \(x \in E\),
\[
I^x = \bigcup_{y \in \text{Parent}(x)} \{ \rho \cdot \text{body}(x) \mid \rho \in I^y \}.
\]

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

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

Proposition 3. If \(x \prec y\) then \(I^x \subseteq I^y\).

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

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

2. We assume that this proposition is satisfied for \(L(x, y) \leq k\) and prove this proposition for \(L(x, y) = k + 1\). From \(x \prec y\), for any \(x' \in \text{Parent}(x)\) there exists \(y' \in \text{Parent}(y)\) that satisfies \(x' = y'\) or \(x' \prec y'\).

For case of \(x' = y'\), \(I^{x'} \subseteq I^y\) holds.

For case of \(x' \prec y'\), \(\text{input}(x') \leq k\), \(\text{input}(y') \leq k\) holds from \(x' \in \text{Parent}(x), y' \in \text{Parent}(y)\) and (A). Therefore \(L(x', y') \leq k\) 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 \(x' \in \text{Parent}(x), \rho' \in I^{x'}\) that satisfy \(\rho = \rho' \cdot \text{body}(x)\). From \(I^{x'} \subseteq I^y\), \(\rho' \in I^y\) holds. This proposition holds from \(\rho = \rho' \cdot \text{body}(x) = \rho' \cdot \text{body}(y) \in I^y\).

\[\square\]

Definition 16 (representative, representable). For \(D \subseteq E, x \in D\) and \(D_x = \{ y \mid y \in D, \text{body}(y) = \text{body}(x) \}\), \(p_x \in D_x\) 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.
Example 13. For \( x, x_1, x_2 \) in example 11, \( x_1 \) and \( x_2 \) are both representative of \( x \) because \( \text{Parent}(x) = \{x_1, x_2\} \), \( x_1 < x_2, x_2 < x_1 \) holds. The case for \( \text{Prune} (\text{Parent}(x)) = \{x_2\} \) 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 \( D \subseteq E \), \( \text{Prune}(D) \) 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 \( \text{Prune}(D) \) from proposition 4.

From definition 17, we have

**Proposition 5.** For \( D \subseteq E \), if \( D \) representable then \( \text{Prune}(D) \) 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 \( \text{Parent}(x) \) is representable for any \( x \in E \).

Example 14. \( G_1 \) is prunable because \( \text{Parent}(x) \) is representable from example 13.

Example 15. Let \( G_2 \) be a context free grammar with following productions.

\[
\begin{align*}
(r^0) & \quad S' : S \rightarrow (r^1) S : T f \\
(r^2) & \quad S : a T g \\
(r^3) & \quad T : X Y d \\
(r^4) & \quad X : a \\
(r^5) & \quad X : b \\
(r^6) & \quad Y : Z e \\
(r^7) & \quad Z : c \\
(r^8) & \quad Z : b c
\end{align*}
\]

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 \( G_2 \) is not prunable because \( \text{Parent}((Y : Z_c, abc)) \) is not representable.

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

**Proposition 6.** For \( x, y \) in Algorithm 2, if \( \text{Parent}(x) \) is separable then \( \text{Parent}(y) \) defined in case 2, 3, 4 is separable.

**Proof.** In Case 2, \( \text{Parent}(x) = \text{Parent}(y) \) from lemma 1 and \( \text{Parent}(y) \) is separable because \( \text{Parent}(x) \) is separable. In Case 3, \( \text{Parent}(y) \subseteq E_\alpha \) and \( \text{Parent}(y) \) is separable because \( E_\alpha \) is separable. In Case 4, \( \text{Parent}(x) \) is separable and \( \text{Parent}(y) \) is separable from lemma. \( \square \)
From this proposition, \( \text{Parent}(y) \) 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,
\[
\bigcup_{y \in \text{Parent}(x)} I^y = \bigcup_{y \in \text{Prune(\text{Parent}(x))}} I^y.
\]

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

From definition 13, we have

**Proposition 7.** For \( \alpha' \in V_T^* \),
\( \alpha \) is a word of \( G \)
if and only if \( \langle i_0, \varepsilon \rangle = \langle S' : \_S \_ \_!, \varepsilon \rangle \mapsto^* \langle S' : S \_ \_!, \alpha' \rangle = \langle \text{Succ}(i_0), \alpha' \rangle \) 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 \( \sigma \in E^a \) that satisfy \( \langle i_0, \varepsilon \rangle \mapsto^* \sigma ; \mapsto^* \langle \text{Succ}(i_0), \alpha \cdot \beta \rangle \) exists.

**Example 16.** For \( \alpha = ab, \beta = ced \),
\( \alpha \cdot \beta = abced \) is a word of \( G \) and \( \sigma^2 \in E^ab = E^a \) in example 8 satisfies
\( \langle i_0, \varepsilon \rangle = \langle S' : \_S \_! \_! \varepsilon \rangle \mapsto^* \sigma^2 ; \mapsto^* \langle S : X \_Y d, ab \rangle \mapsto^* \langle S' : S \_ \_!, abced \rangle = \langle \text{Succ}(i_0), \alpha \cdot \beta \rangle \) from example 9.

For \( \sigma \) that satisfies condition of proposition 8, every elements(\( \sigma_j \)) of \( \sigma \) will be popped while reading and parsing \( \beta \). After \( \sigma_j \) will be popped, \( \text{body(\( \sigma_{j-1} \))} \), which is the body part of an element of the parent set of \( \sigma_j \) will shift to \( \text{Succ(\text{body(\( \sigma_{j-1} \))})} \). After this shifted element will be popped, the body part of \( \sigma_{j-2} \) will shift to \( \text{Succ(\text{body(\( \sigma_{j-2} \))})} \) 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 \( \sigma \in E^a, \beta_j \in V_T^*, \sigma^j \in E^{a; \beta_j} \) (1 \( \leq \) \( j \leq \# \sigma - 1 \)) that satisfy following conditions.

1. \( \# \sigma^j = \| \sigma - j \|
2. \( \sigma^j_k = \sigma_k \) (1 \( \leq \) \( k \leq \# \sigma^j - 1 = \# \sigma - j - 1 \))
3. \( \sigma^j_i = \langle \text{Succ(\text{body(\( \sigma_{j-i} \))})}, \alpha \cdot \beta_j \rangle \)
4. \( \beta_{\sigma - 1} = \beta \)

**Example 17.** For \( \alpha = abc \in V_T^* \),
there exists \( \beta = ed \in V_T^* \) that satisfy \( \alpha \cdot \beta \) is a word of \( G \).
For \( \sigma^{21}, \sigma^{42} \) in Fig. 7
\( \sigma = \sigma^{21}, \beta_1 = \epsilon, \beta_2 = \beta = ed \),
\( \sigma^1 = \sigma^{42} \in E^{abc} = E^{a; \beta_1}, \sigma^2 = \langle S' : S \_ \_!, abced \rangle \in E^{abc} = E^{a; \beta_2} \)
satisfy conditions of proposition 9. Relations between elements of \( \sigma, \sigma^1, \sigma^2 \) are shown in Fig. 15.
\[
\begin{align*}
\sigma' &= \sigma \cdot \alpha \\
\sigma_1' &= \sigma_1 \cdot \beta_1
\end{align*}
\]

Between top elements of \( \sigma \) and \( \sigma_j \),
\[
\sigma_1 \rightarrow^* \sigma_1' = \langle \text{Succ(body}(\sigma_1)), \alpha \cdot \beta_1 \rangle,
\]
\[
\langle \text{Succ(body}(\sigma_{j-1})), \alpha \cdot \beta_j \rangle = \sigma_j \rightarrow^* \sigma_{j+1} = \langle \text{Succ(body}(\sigma_{j-1})), \alpha \cdot \beta_{j+1} \rangle \text{ holds for } 1 \leq j \leq \frac{\|\sigma\| - 2}{2}
\]
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 \( \sigma \in E^\alpha, \beta_j \in V_T^* \) \((1 \leq j \leq \frac{\|\sigma\|}{2})\) that satisfy following conditions

1. \( \sigma_1 \rightarrow^* \langle \text{Succ(body}(\sigma_1)), \alpha \cdot \beta_1 \rangle \).

2. \( \langle \text{Succ(body}(\sigma_{j-1})), \alpha \cdot \beta_j \rangle \rightarrow^* \langle \text{Succ(body}(\sigma_{j-1})), \alpha \cdot \beta_{j+1} \rangle \) \((1 \leq j \leq \frac{\|\sigma\|}{2}) \).

We have a following proposition because conditions of above proposition depend on the body part of \( \sigma \).

**Proposition 11.** For \( \alpha, \beta \in V_T^*, \sigma \in E^\alpha \) that satisfy the following condition
\[
\langle i_0, e \rangle \rightarrow^* \sigma_1 \rightarrow^* \langle \text{Succ}(i_0), \alpha \cdot \beta \rangle
\]
and for \( x, y \in \text{Parent}(\sigma_1) \) that satisfy \( I^x = I^y \),

if there exist \( \gamma \in V_T^*, \sigma_x \in E^x \) that satisfy
\[
\sigma_1 \rightarrow^* \langle \text{Succ(body}(\sigma_x)), \alpha \cdot \gamma \rangle \rightarrow^* \langle \text{Succ}(i_0), \alpha \cdot \beta \rangle,
\]
there exist \( \sigma_y \in E^y \) that satisfy
\[
\sigma_1 \rightarrow^* \langle \text{Succ(body}(\sigma_y)), \alpha \cdot \gamma \rangle \rightarrow^* \langle \text{Succ}(i_0), \alpha \cdot \beta \rangle.
\]

From proposition 11, let \( x, y \) be elements in \( \text{Parent}(z) \) 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 \( \sigma_x \) for all input.

**Example 18.** For \( x, x_1, x_2 \) in example 11,
let \( \sigma_{x_1}, \sigma_{x_2} \) be \( \sigma_{x_1} = (e_0, x_1), \sigma_{x_2} = (e_0, x_2) \) then \( E^{x_1} = \{ \sigma_{x_1} \}, E^{x_2} = \{ \sigma_{x_2} \} \) hold. As \( I^{x_1} = I^{x_2} \) holds,
\[
x = \sigma_1 \rightarrow^* \langle \text{Succ(body}(\sigma_{x_1})), abc \cdot e \rangle \rightarrow^* \langle \text{Succ}(i_0), abc \cdot ed \rangle \text{ and}
\]
\[
x = \sigma_1 \rightarrow^* \langle \text{Succ(body}(\sigma_{x_2})), abc \cdot e \rangle \rightarrow^* \langle \text{Succ}(i_0), abc \cdot ed \rangle \text{ hold for } \sigma \in E^x.
\]

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 \textbf{Reduce} is an array whose \( n \)-th component \( \text{\textbf{Reduce}[n]} \) corresponds to \textbf{Reduce}(\( n \))(\( E_0, a \)) in Algorithm 2. In \textbf{Is pre c} of Algorithm 5, duplicate calculation is suppressed by a table \textbf{PrecTab}. The initial value of elements of \textbf{PrecTab} is 0 and for \( x, y \in E \) satisfying \( \text{body}(x)=\text{body}(y)=i \), the value of \textbf{PrecTab}[\( i \), \#input(\( x \)) \text{, \#input(\( y \))}] is 1 if \( x < y \) holds, -1 if \( x < y \) does not hold and 0 otherwise.

I analyze the time complexity of Pruned Parsing.
Algorithm 4 (Pruned Graph-Structured Stack Parsing a)

Parse(a) /*returns true if alpha is a word, false otherwise*/
(1) $E_{ap} := \{a\}; Parent[a] := \emptyset; \n(2) for $i \in I_G$
(3) for $j$ from $0$ to $i$ for $k$ from $0$ to $i$
(4) $PrecTab[i,j,k]: = 0; \n(5) for $j$ from $1$ to $i$
(6) $E_{ap} := \text{Goto}(E_{ap}, \alpha_j)$;
(7) $E_{ap} := \text{Goto}(E_{ap}, \emptyset)$;
(8) return ($E_{ap} \neq \emptyset$);

Goto($E_{ap}$, $a$) /* Generate $E_{ap}$ from $E_a$ and $a$ */
(9) $Reduce[\text{case}]: = \emptyset; PrecTab[i, j + 1, j]: = \text{FALSE};$
(10) for $x \in E_a$
(11) if $x \in \text{Sitem}(a)$ then /* For Case 2 */
(12) $\text{Eshift}0: = \text{Shift}(x, a);$ $Reduce[\text{case}]: = \text{Reduce[case]U\text{Eshift0}};$
(13) for $y \in \text{Eshift0} /* Parent[y] in Case 2 */
(14) $\text{Parent}[y]: = \text{Parent}[x]; /* From lemma 1 */$
(15) if $x \in \text{Sitem}(a)$ then /* For case 3 */
(16) $\text{Eshift1}: = \text{Shift1}(x, a);$ $Reduce[\text{case}]: = \text{Reduce[case]U\text{Eshift1}};$
(17) for $y \in \text{Eshift1} /* Parent[y] in Case 3 */
(18) $\text{Parent}[y]: = \emptyset;$
(19) for $z \in E_a$
(20) if $z \in \text{Sitem}(a) \land y \in \text{Shift1}(z, a)$ then
(21) $\text{Parent}[y]: = \text{Parent}[y] \cup \{z\};$
(22) return Reduce(Reduce[case]);

Algorithm 5 (Pruned Graph-Structured Stack Parsing b)

Prune(D)
(36) for $x^i \in \text{Parent}[z] \cap \text{Ritem}(z)$
(37) if $y \in \text{Reduce}(x^i, z)$ then
(38) $\text{Parent}[y]: = \text{Parent}[y] \cup \{z\};$
(39) $\text{Parent}[y]: = \text{Prune(}\text{Parent}[y]);$
(40) if $\text{Reduce} \neq [n + 1] \neq \emptyset$
(41) then $E_{\text{return}}: = \text{Ereturn} \cup \text{Reduce}[n + 1];$
(42) else return $E_{\text{return}}; /*exit for loop ((24)-(42)) */

Reclose(D) /*case 4, 5 of Algorithm 3 */
(43) $\text{D_result}: = \emptyset;$
(44) for $i \in I_G$ $\text{D}^i[i]: = \emptyset;$
(45) for $x \in D$
(46) $\text{D}_x[b(x)]: = \text{D}_x[b(x)] \cup \{x\};$
(47) for $i \in I_G$
(48) if $| \text{D}_x[i] | = 1$ then
(49) $\text{D_result}: = \text{D_result} \cup \text{D}_x[i];$
(50) else if $| \text{D}_x[i] | > 1$ then
(51) $p_x: = \text{any element of D}_x[i];$
(52) for $x \in \text{D}_x[i] /* choose representatives */$
(53) if $\text{isprec}(x, p_x)$ then
(54) $p_x: = x;$
(55) for $x \in \text{D}_x[i] /*error if p_x is NOT representative*/
(56) if not $\text{isprec}(x, p_x)$
(57) then Error /* This grammar is NOT prunable*/
(58) $\text{D_result}: = \text{D_result} \cup \{p_x\};$
(59) return $\text{D_result};$

Reclose(D) /*case 4, 5 of Algorithm 3 */
(60) $\text{i}:= \text{body}(x);$ $l_x:= \text{input}(x);$ $l_y:= \text{input}(y);$
(61) if $\text{PrecTab}[i, l_x, l_y] = 0$ then /* x \& y holds */
(62) return True;
(63) if $\text{PrecTab}[i, l_x, l_y] = 1$ then /* x \& y does NOT hold */
(64) return False;
(65) $\text{flag}: = \text{True};$
(66) $x^i \in \text{Parent}[z] \{67\}$ $\text{xflag}: = \text{False};$
(67) $\text{flag}: = \text{False};$
(68) for $y \in \text{Parent}[y]$
(69) $\text{xflag}: = \text{xflag} \cup \{x^i = y\};$
(70) $\text{flag}: = \text{xflag} \cup \{x^i = y\};$
(71) if $\text{flag}$ then $\text{PrecTab}[i, l_x, l_y] = 1;$$\text{flag}:= \text{False};$
(72) else $\text{PrecTab}[i, l_x, l_y] = -1;$
(73) return $\text{flag};$

Figure 16: Pruned Graph-Structured Stack Parsing
Definition 19 (degenerated). For $D \subseteq J_G$, $D$ is called degenerated if $\text{body}(x) = \text{body}(y)$ for all $x, y \in D$.

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 $\text{Shift}(x, a)$ and assignment in this algorithm are constant in $n$.

Lemma 5. For $x, x' \in J_G$ and $a \in V_T$, the sets $\text{Shift}(x, a)$, $\text{Shift}(x', a)$, $\text{Reduce}(x', x)$ and $\text{Reduce}(x', x)$ are degenerated and computed in constant time in $n$ (exactly, $O(|J_G|)$).

Proof. Let $y$ be an element of these 4 sets. From Definitions 6 and 9, $\text{body}(y)$ depends only on $\text{body}(x')$ and $\text{body}(x)$. 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 $\text{body}(x)$ and $\text{body}(x')$ 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 $O(|J_G|)$. The body part of each element is obtained before parsing, its input part is identical, so that the set can be decided in $O(|J_G|)$. \qed

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 $\text{Rc}(D)$ and $\text{Goto}(D, a)$ in Pruned Parsing are degenerated.

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

Lemma 8. $E_{\text{top}}$ and $E_{\alpha}$ in Pruned Parsing is always separable.

Lemma 9. For $y \in E_{\text{top}}$, $\text{Parent}(y)$ is separable.

Proof. For $y \in E_{\text{top}}$, one of the following is satisfied:

a) $y \in \text{Shift}(x, a)$ for some $x \in E_{\alpha}$,

b) $y \in \text{Shift}(x, a)$ for some $x \in E_{\alpha}$,

c) $y \in \text{Reduce}(x', x)$ for some $x \in E_{\alpha}$, $x' \in \text{Parent}(x)$,

d) $y \in \text{Reduce}(x', x)$ for some $x \in E_{\alpha}$, $x' \in \text{Parent}(x)$.

For a), $\text{Parent}(y)$ is separable from induction hypothesis, because $\text{Parent}(y) := \text{Parent}(x)$ at (13). For b), this lemma holds because $\text{Parent}(y) = \{x' \mid x' \in E_{\alpha} \cap \text{S1item}(a), y \in \text{Shift}(x', a)\}$ from (16) – (20), therefore $\text{Parent}(y) \subseteq E_{\alpha}$. For c), this lemma holds from the induction hypothesis, because $\text{Parent}(y) := \text{Parent}(x')$ at (30). For d), the element of $\text{Parent}(y)$ is inserted by PruneInsert at (39) that returns a separable set. \qed

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.
Proof. The time complexity of the pruned parsing is $|\text{Parse}|$ that is equal to $n \times |\text{Goto}|$ from the loop of (2) where $n$ is the length of $\alpha$. 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, $|\text{Goto}|$ is equal to $|\text{RClosure}|$. 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, $|\text{RClosure}|$ 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 $\text{Reduce}[n]$ in (35) and $\text{Parent}[z]$ in (36) are separable, the number of $\text{Parent}[y]$ which is a parameter of Prune at (39) is less than $|I_G|$. Loops in (52) and (55) does not depend on $n$ because $|D_x[\cdot]| \leq |I_G|^2$ from $D_x[\cdot] \subseteq \text{Parent}[y](i \in I_G)$. Therefore $|\text{Prune}| = |\text{Is\_prec}|$ holds.

As $\text{Parent}[x]$ in (66) and $\text{Parent}[y]$ in (68) are separable, the time complexity to get $\text{PrecTab}[i,x,y]$ at (71) and (72) from known values of PrecTab is $O(1)$. Therefore the time complexity to get all values of $\text{PrecTab}[i,j,k]$ 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 $S_3$ be a context free grammar with following productions.

\[(r^0) \quad S' : S \rightarrow \quad (r^1) \quad S : S \quad (r^2) \quad S : a \quad (r^3) \quad S : a \]

$S_3$ 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 $S_3$ except that $r^1$ has $m$ nonterminals (S) in the right hand side.

Example 20. For $S_3$ in example 19, $L^n$ for inputs 'aa' and 'aaa' is shown in Fig. 17. In Fig. 17, \[\uparrow\text{reduce} and \downarrow\text{reduce} is omitted. \]

Stack tops of parse stacks in $L^n$ and $L^{n+1}$ for $S_3$ are shown in Fig. 18. In Fig. 18, $a^n$ in an input part is denoted as $n$.  

17
From Fig. 18, we have

**Proposition 14.**
Parent($\langle S : S_S a, n + 1 \rangle$) = \{\epsilon_0, \langle S : S_S S, k, \langle S : S_S S, l \rangle \rangle | 1 \leq k \leq n, 2 \leq l \leq n \},

Parent($\langle S : S_S S, n + 1 \rangle$) = \{\epsilon_0, \langle S : S_S S, k, \langle S : S_S S, l \rangle \rangle | 1 \leq k \leq n, 2 \leq l \leq n \},

Parent($\langle S : S_S S, n + 1 \rangle$) = \{\epsilon_0, \langle S : S_S S, k, \langle S : S_S S, l \rangle \rangle | 1 \leq k \leq n - 1, 2 \leq l \leq n - 1 \}.

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

**Proposition 15.** For $k$ and $l$ that satisfy $k < l$,

$\langle S : S_S S, k \rangle \prec \langle S : S_S S, k \rangle$,

$\langle S : S_S S, k \rangle \prec \langle S : S_S S, l \rangle$,

$\langle S : S_S S, k \rangle \prec \langle S : S_S S, l \rangle$.

From the definition of Prune and above proposition, we have

**Proposition 16.** For case of $n \geq 2$

Prune(Parent($\langle S : S_S a, n + 1 \rangle$)) = \{\epsilon_0, \langle S : S_S S, n, \langle S : S_S S, n \rangle \rangle \},

Prune(Parent($\langle S : S_S S, n + 1 \rangle$)) = \{\epsilon_0, \langle S : S_S S, n, \langle S : S_S S, n \rangle \rangle \},

Prune(Parent($\langle S : S_S S, n + 1 \rangle$)) = \{\epsilon_0, \langle S : S_S S, n - 1, \langle S : S_S S, n - 1 \rangle \rangle \}.

From above proposition, $S_3$ is prunable and we have

**Proposition 17.** The time complexity for parsing $S_3$ 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 $\langle S : S_S S, k \rangle$ ($1 \leq k \leq 4$) and their parent sets in $S_3$ 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
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), \( \langle i, j \rangle \in \text{Parent}(\langle i, k \rangle)(1 \leq j < k \leq 3), e_0 \in \text{Parent}(\langle i, k \rangle) \) hold from proposition 15. In pruned case (lower left part), \( \langle i, k-1 \rangle \in \text{Parent}(\langle i, k \rangle)(1 < k \leq 4), e_0 \in \text{Parent}(\langle i, k \rangle) \) 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), \( \langle i, j \rangle \in \text{Parent}(\langle i, 4 \rangle) \) and \( (1 \leq j < 4), e_0 \in \text{Parent}(\langle i, 4 \rangle) \) hold. In pruned case (lower middle part), \( \langle i, j \rangle \in \text{Parent}(\langle i, 4 \rangle) \) \( (j = 2, j = 3) \) and \( e_0 \in \text{Parent}(\langle i, 4 \rangle) \) 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 \( \text{Parent}(\langle i, 4 \rangle) \) by pruning. Therefore \( \langle i, 3 \rangle \in \text{Parent}(\langle i, 4 \rangle), e_0 \in \text{Parent} \) holds. In this case, \( I^{i,k}(1 \leq k \leq 4) \) is conserved after pruning.

8 Applicable and Inapplicable grammars

In this section, we show examples of grammars that can be parsed or cannot 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 \( \text{Parent}(x) \) 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 \( |\text{Parent}(x)| \) 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. \( I = O(n) \),
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.

\[
(r^0) S' : S \rightarrow (r^1) S : a S \quad (r^2) S : X \quad (r^3) X : X a \quad (r^4) X : a
\]

Hereafter we denote \( S : a_\_ S \) as \( i \) and \( X : X_\_ a \) as \( i' \).

\[|\text{Parent}(\langle i', n \rangle)| = O(n) \text{ holds because } \text{Parent}(\langle i', n \rangle) = \{\langle i, k \rangle | 2 \leq k \leq n \} \text{ and } \prec \text{ relation does not hold between } \langle i, k \rangle (1 \leq k \leq n) \text{ because } \text{Parent}(\langle i, 1 \rangle) = \{e_0\}, \text{Parent}(\langle i, n \rangle) = \{\langle i, n-1 \rangle | 2 \leq n \}. \text{ } \]

\( G_{RL} \) can NOT be parsed in \( O(n^2) \) by neither Graph Structured Stack Parsing nor Pruned Graph Structured Stack Parsing.
Example 23. Let $G_{RR}$ be context free grammar with following productions. 

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

Hereafter we denote $S : a_\ldots S$ as $i$ and $X : a_\ldots X$ as $i'$. 

Parent$(\langle i, n \rangle)$ and Parent$(\langle i', n \rangle)$ is separable because Parent$(\langle i, 1 \rangle) = \{e_0\}$, Parent$(\langle i, n \rangle) = \{\langle i, n - 1 \rangle, \langle i', n - 1 \rangle\}\quad (2 \leq n)$ 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.

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

Hereafter we denote $S : a_\ldots S$ as $i$ and $X : X_\ldots a$ as $i'$. 

$G_{LL}$ is pruneable because $\langle i, j \rangle \prec \langle i, k \rangle \quad (1 \leq j < k < n)$ hold from Parent$(\langle i, n \rangle) = \{e_0\}$, Parent$(\langle i', n \rangle) = \{\langle i, k \rangle\}\quad (1 \leq k < n)$. $G_{LL}$ 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 $\langle i, j \rangle, \langle i', k \rangle \quad (1 \leq j, k \leq n)$ 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 annotated 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.

20
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"

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(r1) of S2, 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 improve the time complexity of parsing. These approaches improve the complexity by suppressing redundant process that are already executed. This approach 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


Appendix  Proofs of Theorem 1 and Theorem 2

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

**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 \(\lambda' = \mu \cdot x \in L^{\alpha'}, x \in \mathrm{S0item}(a), y \in \mathrm{Shift0}(x, a)\).
3. \(\alpha = \alpha' \cdot a, \lambda = \mu \cdot x \cdot y\) for \(x \in J_G\), where \(\lambda' = \mu \cdot x \in L^{\alpha'}, x \in \mathrm{S1item}(a), y \in \mathrm{Shift1}(x, a)\).
4. \(\alpha = \alpha' \cdot a, \lambda = \mu \cdot x \cdot x' \in J_G\) where \(\lambda' = \mu \cdot x' \cdot x \in R^{(n)}(L^{\alpha'}, a) \subseteq E^\alpha, n \geq 0, x \in \mathrm{Redex}, x' \in \mathrm{R0item}(x), y \in \mathrm{Reduce0}(x', x)\).
5. \(\alpha = \alpha' \cdot a, \lambda = \mu \cdot x \cdot x' \cdot y\) for \(x \in J_G\) where \(\lambda' = \mu \cdot x' \cdot x \in R^{(n)}(L^{\alpha'}, a) \subseteq E^\alpha, n \geq 0, x \in \mathrm{Redex}, x' \in \mathrm{R1item}(x), y \in \mathrm{Reduce1}(x', x)\).

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.

\[\mathrm{Parent}(\lambda_{i+1}) \subseteq \emptyset, \quad (A)\]

\[\lambda_j \in \mathrm{Parent}(\lambda_{i+1}) (1 \leq j < \mu\lambda - 1) \quad (B)\]

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

\[\lambda_i \in E_\alpha, \quad (C)\]

\[\lambda_{(\lambda-1)} \in \mathrm{Parent}(\lambda_i) \quad (D)\]

From (B) and (D)

\[\lambda_j \in \mathrm{Parent}(\lambda_{i+1}) (1 \leq j < \mu\lambda) \quad (E)\]

holds. By (A), (C) and (E), \(\lambda \in E^\alpha\), and this theorem is satisfied.

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

(2-C) As \(\lambda_i = y \in \mathrm{Shift0}(x, a)\) and \(x = \lambda_i' \in E_\alpha\), \(\lambda_i = y \in E_{\alpha'} = E_\alpha\) holds.

(2-D) As \(\mu_i = \lambda_{(\lambda-1)}' \in \mathrm{Parent}(\lambda_{i-1}) = \mathrm{Parent}(x)\) (by induction hypothesis)
and \(\lambda_i \in E_\alpha\), \(\lambda_i \in \mathrm{Parent}(\lambda_i)\) holds.

(3-C) As \(\lambda_i = y \in \mathrm{Shift1}(x, a)\) and \(x = \lambda_i' \in E_\alpha\), \(\lambda_i = y \in E_{\alpha'} = E_\alpha\) holds.

(3-D) As \(x = \lambda_i' \in E_\alpha\) (by induction hypothesis) and \(\lambda_{(\lambda-1)} \in \mathrm{Parent}(y) = \mathrm{Parent}(x)\) (from lemma 1),
\(\lambda_i \in \mathrm{Parent}(y) = \mathrm{Parent}(\lambda_i)\) holds.

(4-C) As \(\lambda_i = y \in \mathrm{Reduce0}(x', x)\) and \(x \in E_\alpha\), \(\lambda_i = y \in E_{\alpha'} = E_\alpha\) holds.

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

(5-C) As \(\lambda_i = y \in \mathrm{Reduce1}(x', x)\) and \(x \in E_\alpha\), \(\lambda_i = y \in E_{\alpha'} = E_\alpha\) holds.

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

\[\square\]

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

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

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

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

For Case 1, this theorem holds, since \( E^x = \{ e_0 \} = \mathcal{L}^x. \)

For case 2, as \( \tau \cdot x \in L^{\alpha'} \) (by \( \tau \cdot x \in E^{\alpha'} \) and induction hypothesis) and \( x \in S0eitem(a), y \in \text{Shift0}(x,a) \) hold, \( \sigma = \tau \cdot y \in L^{\alpha' \cdot a} = L^\alpha \) is satisfied from case 2 of Algorithm 2.

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 \( x \in \text{Redex}, x' \in R_{(n)}(E^{\alpha'}, a), x \in \text{R0eitem}(x), y \in \text{Reduce0}(x',x) \) hold, \( \sigma = \tau \cdot y \in L^\alpha \) is satisfied from case 4 of Algorithm 2.

For case 5, \( \sigma \in L^\alpha \) is satisfied similar to case 4. \[ \square \]