Integration of Morphological and Syntactic Analysis Based on LR Parsing Algorithm

Tanaka Hozumi, Tokunaga Takenobu And Aizawa Michio

Department of Computer Science, Tokyo Institute of Technology
2-12-1 Ôokayama Meguro Tokyo 152 Japan
email: take@cs.titech.ac.jp

Abstract

Morphological analysis of Japanese is very different from that of English, because no spaces are placed between words. The analysis includes segmentation of words. However, ambiguities in segmentation is not always resolved only with morphological information. This paper proposes a method to integrate the morphological and syntactic analysis based on LR parsing algorithm. An LR table derived from grammar rules is modified on the basis of connectabilities between two adjacent words. The modified LR table reflects both the morphological and syntactic constraints. Using the LR table, efficient morphological and syntactic analysis is available.

1 Introduction

Morphological analysis of Japanese is very different from that of English, because no spaces are placed between words. This is also the case in many Asian languages such as Korean, Chinese, Thai and so forth. In the Indo-European family, some languages such as German have the same phenomena in forming complex noun phrases. Processing such languages requires the identification of the boundaries of words in the first place. This process is often called *segmentation* which is one of the most important tasks of morphological analysis for these languages.

Segmentation is a very important process, since the wrong segmentation causes fatal errors in the later stages such as syntactic, semantic and contextual analysis. However, correct segmentation is not always possible only with morphological information. Syntactic, semantic and contextual information may help resolve the ambiguities in segmentation.

Over the past few decades a number of studies have been made on the morphological and syntactic analysis of Japanese. They can be classified into the following three approaches:

Cascade: Separate the morphological and syntactic analysis and execute them in a cascade manner. The morphological and syntactic constraints are represented separately.

Interleave: Separate the morphological and syntactic analysis and execute them interleavingly. The morphological and syntactic constraints are represented separately.

Single Framework: Represent both the morphological and syntactic constraints in a single framework such as context free grammars (CFGs) and make no distinction between the two analysis.

Representing the morphological and syntactical constraints separately as in the first two approaches, Cascade and Interleave, makes maintaining and extending the constraints easier. This is an advantage of these approaches. Many natural language processing systems have used these two approaches. For example, Mine proposed a method to represent the morphological constraints in regular grammar and the syntactic constraints in CFG, and interleave the morphological and syntactic analysis (Mine et al., 1990). Most other systems use a connection matrix instead of a regular grammar (Miyazaki et al., 1984;

Sugimura et al., 1989). The main drawbacks of these approaches are as follows:

- It may require two different algorithms for each analysis.
- It must retain all ambiguities from the morphological analysis until the syntactic analysis begins. This wastes memory space and computing time.

On the other hand, from a viewpoint of processing, it is preferable to integrate the morphological and syntactic analysis into a single framework, since some syntactic constraints are useful for morphological analysis and vice versa. The last approach fulfills this requirement. There have been several attempts to develop CFG that covers both the morphological and syntactic constraints (Kita, 1992; Sano–Fukumoto, 1992). However, it is empirically difficult to describe both constraints by using only CFG. The difficulty arises due to the timing of connectability checks, but also increases the number of CFG rules. For example, in figure 1, in order to check the connectability between adjacent words, w_i and w_{i+1}, the morphological attributes of each word should be propagated up to their mother nodes B and C, and the check is delayed until the application of the rule A → B C. Therefore, problems such as the possibility of delays in connectability checking and propagation of morphological attributes to upper nodes make the algorithm of connectability checking more complex and can cause difficulties in representing morphological and syntactical constraints by CFG.

However, by using connection matrices for morphological analysis as in the Cascade/Interleave approaches, connectability checks between adjacent words is performed very easily. Therefore, it is desirable to represent the morphological and syntactic constraints separately as in Cascade/Interleave, and to integrate the execution of both analysis into a single process as in Single Framework. In our method, we have captured these advantages by representing the morphological constraints in connection matrices and the syntactic constraints in CFGs, then compiling both constraints into an LR table (Aho et al., 1986). The already existing, efficient LR parsing algorithms can be used with minor modifications, enabling us to utilize both the morphological and syntactic constraints at the same time.

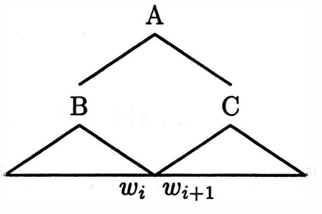


Fig. 1 Connectability check by CFG

In the next section, we first give a brief introduction to Japanese morphological analysis using an example sentence. In section 3, we describe the method of generating an LR table from a connection matrix and CFG rules, then in section 4 we explain the detail of our method based on generalized LR parsing algorithm with an example. Our algorithm is principally the same as Tomita's generalized LR parsing algorithm (Tomita, 1986), but the input is not a sequence of preterminals, but a sequence of characters.

2 Morphological analysis of Japanese

A simple Japanese sentence consists of a sequence of postpositional phrases (PPs) followed by a predicate. The PP consists of a noun phrase (NP) followed by a postposition which indicates the case role of the NP. The predicate consists of a verb or an adjective, optionally followed by a sequence of auxiliary verbs (Morioka, 1987).

We illustrate the Japanese morphological analysis with an example sentence "KaORuNi-AIMaSu (meet Kaoru)." [1] We use a simple Japanese dictionary shown in figure 2, and a connection matrix shown in figure 3 which gives us the connectabilities between adjacent morphological categories (mcat). For example in figure 3, the symbol "o" at the intersection of row 2 (p1) and column 3 (vs4k) indicates that the morphological category **vs4k** can immediately follow the morphological category **p1**.

[1] Each capitalized one or two character(s) corresponds to a single Japanese character (Kana character).

entry	cat	mcat	meaning
KaO	n	n1	face
KaO	vs	vs4r	smell sweet
Ru	ve	ve4r3	(connect to nominal)
KaORu	n	n1	person's name
Ni	p	p1	(dative)
A	vs	vs4k	open
A	vs	vs4w	meet
Ki	ve	ve4k2	(connect to verb)
I	ve	ve4k2i	(connect to verb)
I	ve	ve4w2	(connect to verb)
Tu	ve	ve4w2t	(connect to verb)
MaSu	ax	ax1	(polite form)
Ta	ax	ax2	(past form)

n: noun, p: case marker, vs: verb stem,
ve: verb ending, ax: aux verb

Fig. 2 A simple Japanese dictionary

RIGHT

LEFT	n1	p1	vs4k	vs4r	vs4w	ve4k2	ve4k2i	ve4r3	ve4w2	ve4w2t	ax1	ax2	$
n1		o											
p1	o		o	o	o								
vs4k						o	o						
vs4r								o					
vs4w									o	o			
ve4k2											o		
ve4k2i												o	
ve4r3													o
ve4w2											o		
ve4w2t												o	
ax1													o
ax2													o

Fig. 3 An example of connection matrix

Using only the dictionary, we can obtain the following twelve candidates of segmentation for the sentence "KaORuNiAIMaSu."

	KaO	Ru	Ni	A	I	MaSu
(1)	n1	ve4r3	p1	vs4k	ve4k2i	ax1
(2)	n1	ve4r3	p1	vs4k	ve4w2	ax1
(3)	n1	ve4r3	p1	vs4w	ve4k2i	ax1
(4)	n1	ve4r3	p1	vs4w	ve4w2	ax1
(5)	vs4r	ve4r3	p1	vs4k	ve4k2i	ax1
(6)	vs4r	ve4r3	p1	vs4k	ve4w2	ax1
(7)	vs4r	ve4r3	p1	vs4w	ve4k2i	ax1
(8)	vs4r	ve4r3	p1	vs4w	ve4w2	ax1

	KaORu	Ni	A	I	MaSu
(9)	n1	p1	vs4k	ve4k2i	ax1
(10)	n1	p1	vs4k	ve4w2	ax1
(11)	n1	p1	vs4w	ve4k2i	ax1
(12)	n1	p1	vs4w	ve4w2	ax1

By also referring to the connection matrix, we can filter out illegal segmentations. From the examples above, we find (1)–(4) violate the connectability between "KaO (n1)" and "Ru (ve4r3)", and that (5)–(8) violate the connectability between "Ru (ve4r3)" and "Ni (p1)." Also (9) and (11) violate the connectability between "I (ve4k2i)" and "MaSu (ax1)", and (11) violates the connectability between "A (vs4w)" and "I (ve4k2i)." Thus by process of elimination we obtain the morphologically correct candidate, (12). However, a long input sentence generally gives many more ambiguities which need to be resolved in later stages using syntactic, semantic and contextual information.

3 Generating A Modified LR Table

Connection matrices and CFG rules have been used for morphological analysis and syntactic analysis respectively by most Japanese processing systems. Because CFG rules were mainly used for syntactic analysis and connection matrices for morphological analysis, they have been developed independently of each other.

In this section, we propose a method to integrate morphological and syntactic constraints in the framework of LR parsing algorithm, and thus capturing the advantages of Cascade/Interleave and Single Framework described in section 1.

In order to combine connection matrices and CFG rules, the first step we have to take is to extend the CFG rules by relating the syntactic categories in the CFG rules with the morphological categories in a connection matrix. This is realized by adding CFG rules called morphological rules each of which is a unit production rule with a syntactic category in the LHS and a morphological category in the RHS.

From the dictionary shown in figure 2, we can extract a set of new CFG rules as shown in figure 6, which are simply added to the CFG rules in figure 4 to get an extended set of CFG rules with morphological constraints.

s → v ax	(1)	v → vs ve	(3)
s → pp s	(2)	pp → n p	(4)

Fig. 4 A simple set of CFG rules for Japanese

state	ACTION													GOTO							
	n1	p1	vs4k	vs4r	vs4w	ve4k2	ve4k2i	ve4r3	ve4w2	ve4w2t	ax1	ax2	$	s	v	ax	pp	vs	ve	n	p
0	sh6		sh7	sh8	sh9									1	2		3	4		5	
1													acc								
2											sh11	sh12				10					
3	sh6		sh7	sh8	sh9									13	2		3	4		5	
4						sh15	sh16	sh17	sh18	sh19									14		
5		sh21																			20
6		re5																			
7						re7	re7	re7*	re7*	re7*											
8						re8*	re8*	re8	re8*	re8*											
9						re9*	re9*	re9*	re9	re9											
10													re1								
11													re15								
12													re16								
13													re2								
14											re3	re3									
15											re10	re10*									
16											re11*	re11									
17											re12*	re12*									
18											re13	re13*									
19											re14*	re14*									
20	re4		re4	re4	re4																
21	re6		re6	re6	re6																

Fig. 5 LR table generated from rules (1)–(16)

n → n1	(5)	ve → ve4k2i	(11)
p → p1	(6)	ve → ve4r3	(12)
vs → vs4k	(7)	ve → ve4w2	(13)
vs → vs4r	(8)	ve → ve4w2t	(14)
vs → vs4w	(9)	ax → ax1	(15)
ve → ve4k2	(10)	ax → ax2	(16)

Fig. 6 A morphological rules derived from the dictionary in Fig. 2

We can generate an LR table as shown in figure 5 from the extended CFG rules (1) through (16) from figure 4 and 6. Note that the extended CFG rules do not include any information about connectability represented in the connection matrix in figure 3. For example, rules (3), (8) and (13) allow the structure "`v(vs(vs4r),ve(ve4w2))`" which violates the connectability between `vs4r` and `ve4w2` as shown in figure 3.

For each reduce action **A** with a morphological
    rule in each entry of LR table {
  if (Not *Connect*(*RHS*(*Rule*(**A**)), *LA*(**A**)) {
    delete **A** from the entry;
  }
}

where each function is defined as follows:

Rule : *action* → *rule*;
    returns a rule used by the reduce action.

LA : *action* → *symbol*;
    returns a look ahead symbols of the action.

Connect : *symbol* × *symbol* → $\{T, F\}$;
    returns true or false with respect to
    the connectability of the two symbols.

RHS : *rule* → *symbol*;
    returns a right hand side symbol of the rule.

Fig. 7 A procedure to modify an LR table

The second step is to introduce the constraints on connectability into the LR table by deleting illegal reduce actions. This is carried out by modifying the LR table with the procedure shown in figure 7.

Deleting reduce actions by applying the above procedure prohibits the application of morphological rules which violates the connectability between two adjacent words, namely the current scanned word and its lookahead word. Note that given an LR table and a connection matrix, this procedure can be performed automatically without human intervention.

It is possible to incorporate this procedure into the LR table generation process, however, it is better to keep them separate. Since this procedure is applicable to any type of LR table, separating this process from LR table generation enables us to use the already existing LR table generation program.

For example, in figure 5, the reduce action `re7` in row 7 and column `ve4r3` is deleted, since the connection between `vs4k`, the RHS of rule (7), and `ve4r3`, the lookahead preterminal, is prohibited as shown in the connection matrix in figure 3. Similarly, reduce action `re7` in row 7 and column `ve4w2` will be deleted and so forth. These deletions are marked with asterisks (*) in figure 5. The overview of generating a modified LR table is shown in figure 8.

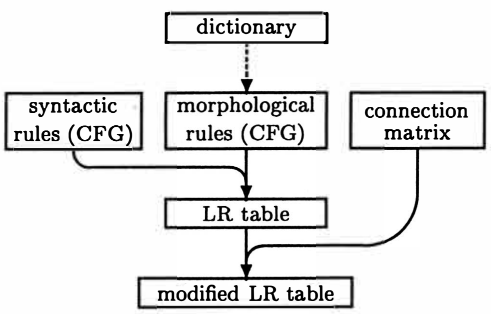


Fig. 8 Outline of generating a modified LR table

Generally speaking, the size of the LR table is on the exponential order of the number of rules in the grammar. Introducing the morphological rules into the syntactic rules can cause an increase in the number of states in LR table, thereby exponentially increasing the size of the overall LR table in the worst case. In our method, the increase of the number of states is equal to that of the morphological rules introduced. Suppose we add a morphological rule $X \rightarrow x$ to the grammar. Only the items in the form of $[A \rightarrow \alpha \cdot X\beta]$ can produce a single new item $[X \rightarrow \cdot x]$ from which only a single new state $\{[X \rightarrow x\cdot]\}$ can be created. Thus the increase of the number of the states is equal to that of the morphological rules introduced, and the size of the LR table will not grow exponentially.

```
(1)  initialize stack
(2)  for CS = 0 ... N {
(3)    for each stack top node in stage CS {
(4)      Look-aheads = lookup-dictionary(CS);
(5)      for each look ahead preterminal LA
           in Look-aheads {
(6)        do reduce while "reduce" is applicable;
(7)        if "shift" is applicable {
(8)          do shift creating a new node
               in stage (CS + length(LA));
(9)        }
(10)       if "acc" { accept }
(11)       if no action { reject }
(12)     }
(13)   }
(14) }
```

Fig. 9 Outline of our parsing algorithm

4 Algorithm for Integrating Morphological and Syntactic Analysis

The LR parsing algorithm with the modified LR table is principally the same as Tomita's generalized LR parsing algorithm. The only difference is that Tomita's algorithm assumes a sequence of preterminals as an input, while our algorithm assumes a sequence of Kana characters[2]. Thus the dictionary reference process needs to be slightly modified. Figure 9 illustrates the outline of our parsing algorithm.

In figure 9 the stage number (CS) indicates how many Kana characters have been processed. The procedure begins at stage 0 and ends at stage N, the length of an input sentence. In stage 0, the stack is initialized and only the node with state 0 exists (step (1)). In the outer-most loop (2)–(14), each stack top in the current stage is selected and processed. In step (4), the dictionary is consulted and look-ahead preterminals are obtained. An important point here is that look-ahead preterminals may have different Kana character lengths. A new node is introduced by a shift action at step (8) and is placed into a stage which is ahead of the current stage by the length of the look-ahead word.

The following example well illustrates the algorithm in figure 9. The input sentence is

[2]We assume the input sentences consist of only Kana characters for brevity. Other types of characters, such as Kanji, can be also handled.

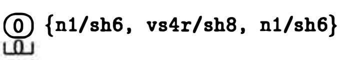
"KaORuNiAIMaSu$ (meet Kaoru)." and we assign position numbers between adjacent Kana characters.

Input:	Ka	O	Ru	Ni	A	I	Ma	Su	$
Position:	0	1	2	3	4	5	6	7	8 9

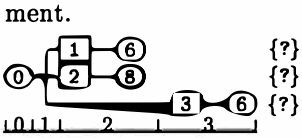
In the following trace, the numbers in circles denote state numbers, and the numbers in squares denote the subtree number shown below the diagrams. The symbols enclosed by curly brackets denote a look ahead preterminal followed by the next applicable action, separated by a slash (/). The stage numbers are shown below the stacks.

Current stage: 0
Dictionary reference:
n1("KaO") at 0–2
vs4r("KaO") at 0–2
n1("KaORu") at 0–3

We find three look ahead preterminals, n1, vs4r, and n1 by consulting the dictionary in figure 2. A shift actions is applied for each of them according to the LR table in figure 5.



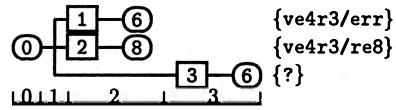
After the shift actions, three new nodes are created at stage 2 or stage 3 depending on the length of look ahead words. At the same time subtrees 1–3 are constructed. The current stage is updated from 0 to 2, since there is no node in stage 1. The look ahead preterminals are unknown at this moment.



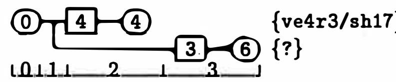
1 : n1("KaO")
2 : vs4r("KaO")
3 : n1("KaORu")

Current stage: 2
Dictionary reference:
ve4r3("Ru") at 2–3

Dictionary reference gives one look ahead preterminal from position 2. Since the current stage number is 2, only the first two stack tops are concerned at this stage. No action is taken of the first stack, because the LR table has no action in the entry for state 6 and a look ahead preterminal ve4r3. As the result, the first stack is rejected. The reduce action (re8) is taken for the second stack.

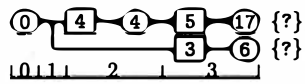


After re8, a shift action (sh17) is carried out for the first stack.



4 : vs(2)

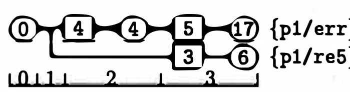
After sh17, we can proceed to stage 3.



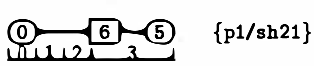
5 : ve4r3("Ru")

Current stage: 3
Dictionary reference:
p1("Ni") at 3–4

We obtain preterminal p1 by consulting the dictionary. Because the first stack can take no more action, it is rejected. The reduce action (re5) is then applied to the second stack.

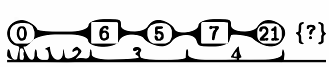


The shift action (sh21) is applied to the following stack.



6 : n(3)

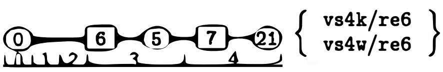
After the shift action (sh21), new nodes are created in stage 4.



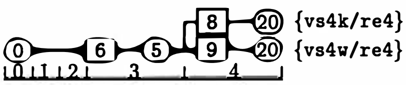
7 : p1("Ni")

Current stage: 4
Dictionary Reference:
vs4k("A") at 4–5
vs4w("A") at 4–5

Dictionary reference provides two look ahead preterminals for the next word.

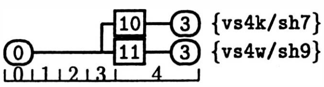


After the two reduce actions (re6), we get two nodes with the same state 20, but they are not merged as the look ahead preterminals are different each other. See stage 5 for the reason.

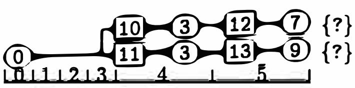


8 : p(7)
9 : p(7)

The process in stage 4 continues as follows.



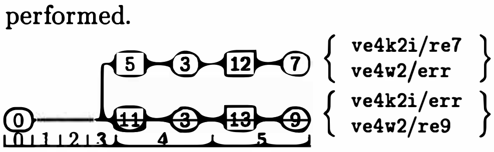
10 : pp(6,8)
11 : pp(6,9)



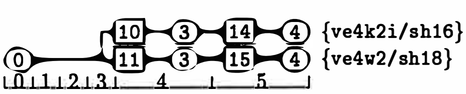
12 : vs4k("A")
13 : vs4w("A")

Current stage: 5
Dictionary Reference:
ve4k2i("I") at 5–6
ve4w2("I") at 5–6

We have two look ahead preterminals and two stack tops. The reduce actions (re7 and re9) are performed.

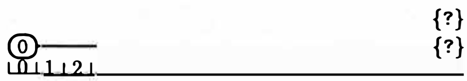


Note that we can not merge the stack tops with the same state 4 since the look ahead preterminals are different (ve4k2i/ve4w2). [3]



14 : vs(12)
15 : vs(13)

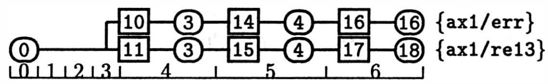
After the shift actions (sh16 and sh18), we proceed to stage 6.

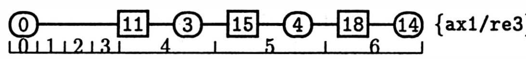


16 : ve4k2i("I")
17 : ve4w2("I")

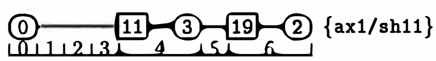
Current stage: 6
Dictionary reference:
ax1("MaSu") at 6–8

The process in stage 6 proceeds as follows.

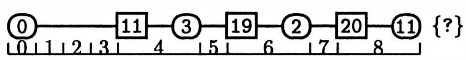




18 : ve(17)

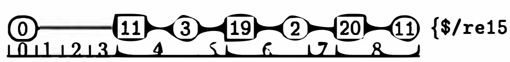


19 : v(15,18)

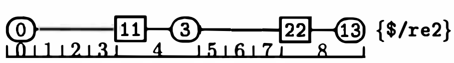


20 : ax1("MaSu")

Current stage: 8
Dictionary reference:
"$" at 8–9

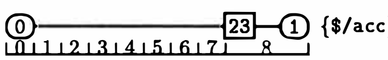


21 : ax(20)



22 : s(19,21)

The input sentence is automatically segmented and accepted, giving a final parse result 23 as shown in figure 10.



23 : s(11,22)

---

[3] If two stack tops are merged and then different shift actions (sh16 and sh18) are carried out, we might have invalid combinations of structure such as (14, 17) and (15, 16).

5 Conclusion

We have proposed a method representing the morphological constraints in connection matrices and the syntactic constraints in CFGs, then compiling both constraints into an LR table. The compiled LR table enables us to make use of the already existing, efficient generalized LR parsing algorithms through which integration of both morphological and syntactic analysis is obtained.

Advantages of our approach can be summarized as follows:

- Morphological and syntactic constraints are represented separately, and it makes easier to maintain and extend them.
- The morphological and syntactic constraints are compiled into a uniform representation, an LR table. We can use the already existing efficient algorithms for generalized LR parsing for the analysis.
- Both the morphological and syntactic constraints can be used at the same time during the analysis.

We have implemented our method using the EDR dictionary with 300,000 words (EDR, 1993) from which 437 morphological rules are derived. This means only 437 new states are introduced to LR table and this does not cause an explosion in the size of the LR table.

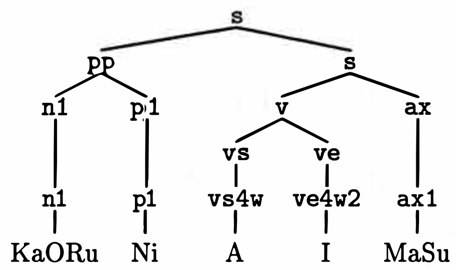


Fig. 10 An analysis of "KaORuNiAIMaSu"

References

Aho, A.V. — Sethi,R. — Ullman, J.D. (1986) *Compilers Principles, Techniques, and Tools.* Massachusetts: Addison-Wesley.

Japan Electronic Dictionary Research Institute (1993) *EDR Dictionary Manual.*

Kita, K. (1992) *A Study on Language Modeling for Speech Recognition.* PhD thesis, Waseda University.

Mine,T. — Taniguchi, R. — Amamiya, M. (1990) "A parallel syntactic analysis of context free grammars." pp. 452–453. the 40th Annual Convention IPS Japan.

Miyazaki, M. (1984) "An Automatic Segmentation Method for Compound Words using Dependency Analysis." Transactions of Information Processing Society of Japan. Vol. 25, No. 6, pp. 970–979

Morioka, K. (1987) *Formation of a Vocabulary.* Meiji-Shoin.

Sano, H. — Fukumoto, F. (1992) "On a Grammar Formalism, Knowledge Bases and Tools for Natural Language Processing in Logic Programming." in Proceedings of FGCS92.

Sugimura,R. — Akasaka, K. — Kubo, Y. — Matsumoto, Y. (1989) "Logic Based Lexical Analyzer LAX." Logic Programming '88 (Lecture Notes in Artificial Intelligence). pp. 188–216. Springer-Verlag.

Tomita, M. (1986) *Efficient Parsing for Natural Language: A Fast Algorithm for Practical Systems.* Boston: Kluwer Academic Publishers.