KB&KS '93

ANALYSIS AND GENERATION TECHNOLOGIES

Tanaka Hozumi and Tokunaga Takenobu and K. G. Suresh and Inui Kentaro
Department of Computer Science
Tokyo Institute of Technology
2-12-1 Meguro Tokyo 152 Japan

ABSTRACT

Analysis and generation are the two main aspects in the natural language processing. In this paper we survey some of the progress made towards natural language analysis (parsing) and generation. Particularly, we consider syntactic analysis and present two frequently used parsing algorithms, namely Chart and GLR parsing algorithms. Then we go on to tell the importance of context sensitiveness in syntactic analysis by surveying probabilistic parsing methods, which are some of the recent developments made in this direction. In the second part of this paper we first give a brief survey on natural generation researches and discuss the future research direction.

Part I: NATURAL LANGUAGE ANALYSIS

1 INTRODUCTION

The idea of natural language processing emerged with the advent of the electronic computer. The parsing (syntactic analysis) and generating natural languages began with the formal linguistic theory developed by N. Chomsky who classified languages into four classes: unrestricted languages, context-sensitive languages, context-free languages and regular languages. These languages are produced by applying rewriting rules, or otherwise called production rule, in the form of $\alpha \rightarrow \beta$ in which a string α is rewritten as another string β. Chomsky pointed out that context-free grammar (CFG) alone is not enough to specify a natural language, and he insisted on the necessity of context sensitiveness. However from the point of parsing and generation, it is very difficult to device an efficient algorithm for context-sensitive grammar. Recently, Gazdar et al., advocate the power of context-free grammar in covering broad range of natural languages [17]. As many efficient algorithms have been developed for context-free grammars, most natural language processing systems use context-free grammars. Figure 1 is a sample of English context-free grammar.

In this paper we focus on the most important parsing algorithms, namely Chart [30] and GLR parsing algorithm [65], which have been frequently used in practical natural language processing as well as speech recognition.

Knuth developed a LR parsing algorithm [33] which could parse an input sentence deterministically and efficiently, but the input sentences were limited to the ones generated by a proper subset of CFG. Tomita et al., extended Knuth's LR parsing algorithm to handle general CFGs, which is called GLR parsing algorithm [65].

To make parsing algorithms robust, they must be able to handle ill-formed sentences with omitted words, unknown/misspelled words, extra noise words, redundancy, etc. We will explain the method to handle these ill-formedness based on Chart parsing algorithm [45, 29, 43].

In general, these parsing algorithms might produce a tremendous amount of parsing results (parsing trees) for a long input sentence. However only a few of them are plausible results from the semantic point of view. Recently, many researchers pay attention to statistical methods to select probable parsing results. For this purpose, probabilistic grammar [66, 68, 16] has been used. In connection with GLR, we will discuss the stochastic LR parsing to compute the probable parsing results. For the simplicity, we avoid the epsilon rule (null production such as $X \rightarrow \epsilon$) in the following discussions.

(1)	S	→	NP VP	(8)	n	→	"I"
(2)	S	→	S PP	(9)	n	→	"man"
(3)	NP	→	n	(10)	n	→	"park"
(4)	NP	→	det n	(11)	v	→	"saw"
(5)	NP	→	NP PP	(12)	det	→	"a"
(6)	PP	→	p NP	(13)	det	→	"the"
(7)	VP	→	v NP	(14)	p	→	"in"

Fig. 1 A sample of English context-free grammar

2 CHART PARSING ALGORITHM

Chart parsing algorithm [30] have been used broadly. The algorithm uses a data structure called *chart* as a book-keeping storage for all the information produced while parsing. The information in the chart is referred to avoid doing redundant works. This is essentially similar to the purpose of items in Earley's algorithm [14]. Hence, the time complexity of parsing a sentence of length n is restricted in the order of n^3 for general CFGs as in the case of using Earley's algorithm.

The information kept in the chart are divided into a set of *active edges* and a set of *complete(inactive) edges*. A complete edge represents a constituent that has been successfully parsed and completed. An active edge represents a constituent with some elements called *remainder* left to be satisfied. In this paper, we specify an "edges" in chart algorithm as in the example below. The elements after the dot represent the remainder. For example, [S → NP · VP, 0, 1] represents an active edge of constituent S starting at position 0, ending at position 1, which has already found its NP, and still needs a VP to be completed. [S → NP VP ·, 0, 4] is an example of a complete edge whose all elements have been found.

Chart provides a very general framework to natural language analysis. In fact, Earley's algorithm can be viewed as a specialization of Chart parsing algorithm which proceeds in the top-down manner. For comparison, in this section we will discuss the bottom-up version of Chart parsing.

Now we will give a left-to-right bottom-up Chart parsing algorithm. In the following, α, β, γ are a sequence of terminal and/or nonterminal symbols, and A,B,C are nonterminal symbols. S is a special symbol representing the start symbol.

Input $w_1 w_2 \ldots w_n$

Output a chart

Algorithm[1]

For $k = 0$ to $n - 1$ do

(a) For each entry C → w_{k+1} in the lexicon, span an inactive edge [C → w_{k+1}·, k, $k+1$] between positions k and $k+1$

Then for each inactive edge between positions j and $k+1 (j < k+1)$ [B → γ·, $j, k+1$], do the following until no new item can be created:

(b) for each rule A → B, span an inactive edge [A → B ·, $j, k+1$].

(c) for each rule A → B β, span an active edge [A → B ·β, $j, k+1$].

(d) for each active edge starting at position i, ending at position j and having B as the leftmost element of the remainder with form [A → α· B , i, j], create an inactive edge [A → α B ·, $i, k+1$] between positions i and $k+1$.

(e) for each active edge starting at position i, ending at position j and having B as the leftmost element of the remainder with form [A → α· B β, i, j], create an active edge [A → α B ·β, $i, k+1$] between positions i and $k+1$.

If we find an edge of the form [S → α·, $0, n$], then accept else reject.

Note that step (b) and (d) generate new inactive edges which will be in turn applied with step (b)-(e). The loop will be done until no more new item is generated.

In the above algorithm, (a) is called scanner operation, (b) and (d) is called completion operation and, (c) and (e) is called predictor operation.

To understand the algorithm more clearly, let us consider parsing the input sentence "I saw a man in the park" using the grammar in figure 1. Bottom-up Chart parsing will proceed as follows.

$k = 0, w_1$ = "I"
  Apply (a) to w_1
    (1) [n → "I" ·, 0, 1]
  Apply (b) to (1)
    (2) [NP → n ·, 0, 1]
  Apply (c) to (2)
    (3) [S → NP · VP, 0, 1]
    (4) [NP → NP · PP, 0, 1]
$k = 1, w_2$ = "saw"
  Apply (a) to w_2
    (5) [v → saw ·, 1, 2]
  Apply (c) to (5)
    (6) [VP → v · NP, 1, 2]
$k = 2, w_3$ = "a"
  Apply (a) to w_3
    (7) [det → a ·, 2, 3]
  Apply (c) to (7)
    (8) [NP → det · n, 2, 3]
$k = 3, w_4$ = "man"
  Apply (a) to w_4
    (9) [n → "man" ·, 3, 4]
  Apply (b) to (9)
    (10) [NP → n ·, 3, 4]
  Apply (c) to (10)
    (11) [S → NP · VP, 3, 4]
    (12) [NP → NP · PP, 3, 4]
  Apply (d) to (8) & (9)
    (13) [NP → det n ·, 2, 4]
  Apply (c) to (13)
    (14) [S → NP · VP, 2, 4]
  Apply (c) to (13)
    (15) [NP → NP · PP, 2, 4]
  Apply (d) to (6) & (13)
    (16) [VP → v NP ·, 1, 4]
  Apply (d) to (3) & (13)
    (17) [S → NP VP · , 0, 4]
  Apply (c) to (15)
    (18) [S → S · PP, 0, 4]
    .....

Continuing in the same way, the rest of the input would be analyzed as a preposition phrase (PP) which is represented by the following edge.

[1] In this case eliminations of epsilon rule makes the following descriptions a slightly longer one.

(19) [PP → p NP ·, 4, 7]

Then this edge would be used to create the following.

Apply (d) to (15)&(19)
(20) [NP → NP PP ·, 2, 7]
Apply (d) to (18)&(19)
(21) [S → S PP ·, 0, 7]
Apply (d) to (6)&(20)
(22) [VP → v NP · , 1, 7]
Apply (d) to (3)&(22)
(23) [S → NP VP ·, 0, 7]

As two edges of the form [S → α·, 0, 7] are generated, the input is accepted as a correct sentence with syntactic ambiguities.

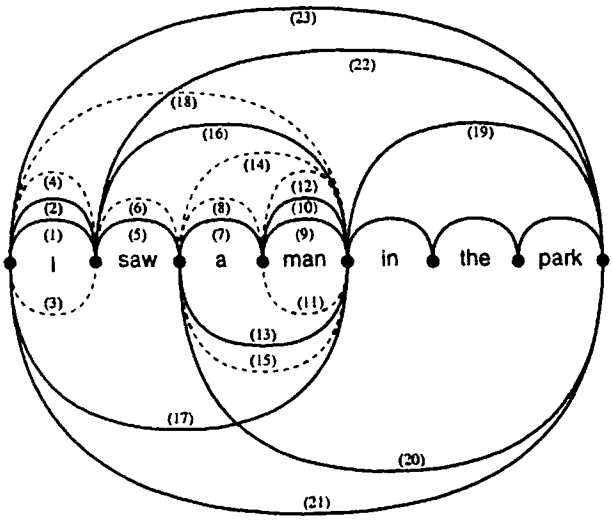


Fig. 2 The chart of parsing "I saw a man in the park"

Simply stating, bottom-up Chart parsing is started by constructing inactive edges corresponding to the individual words and their categories. If there are more than one category for a word, all edges corresponding to all of its categories will be constructed. Then the algorithm tries to construct larger constituents from those inactive edges by matching them with CFG rules or by merging them with active edges. For example, the inactive edge (1) is matched with the rule NP → n in the grammar to construct the active edge (3), and the inactive edge (9) is merged with the active edge (8) to construct the inactive edge (13). If at last some edges of the category S are spanned from position 0 to position n, the input would be parsed as a correct sentence.

A chart can be viewed as a graph where each vertex represents a position in a sentence and each arc linking vertices represents an edge. A visualized version of chart is shown in figure 2.

The bottom-up and top-down strategies have different advantages. Since a bottom-up strategy starts by looking at the input and then building up the larger structures, the structures that cannot be projected down to the input words will never be built. On the other hand, because a top-down strategy begins by looking for a sentence and then searching for its substructures, the structures that cannot be projected up to the structure of a sentence would not be constructed. Mixing both strategies in a proper way can achieve more efficient parsing [67].

3 GLR PARSING ALGORITHM

The LR(k) parser [33], is one of the most efficient shift/reduce parser based on rightmost derivations. It can parse deterministically and efficiently any input sentences generated by a LR(k) grammar which is a subset of CFG. Tomita extended the LR(k) parser to handle a general CFG in a breadth-first manner [65]. The Tomita's algorithm is known as one of the most efficient generalized LR (GLR) parser. Empirically, Tomita's algorithm is faster than Chart algorithm. In this section we will give a brief introduction of Tomita's algorithm with an example and discuss some of the problems.

GLR parsing algorithm uses a stack and a LR table constructed from general CFG to guide the parsing process. All the necessary predictor operations are compiled into a LR table in advance. LR table construction algorithm can be found in [2]. An example of a LR table of figure 1 is shown in figure 3 which consists of two fields, an Action field and a Goto field.

	Action field					Goto field			
State	det	n	v	p	$	NP	PP	VP	S
0	sh3	sh4				2			1
1				sh6	acc		5		
2			sh7	sh6			9	8	
3		sh10							
4			re3	re3	re3				
5				re2	re2				
6	sh3	sh4				11			
7	sh3	sh4				12			
8				rc1	rc1				
9			re5	re5	re5				
10			re4	re4	re4				
11			rc6	re6/ sh6	re6		9		
12				re7/ sh6	re7		9		

Fig. 3 A LR table for the grammar in figure 1

The parsing actions are determined by state (the row of the table) and a look-ahead preterminal[2] (the column of the table). Here, $ represents the end of an input sentence. The "shN" in some entries of the Action field indicates that the GLR parser has to push a look-ahead preterminal on the stack and shift to "state N". The symbol "reN" indicates that the parser has to pop several vertices (corresponding to right hand side of the rule numbered "N") from the top of the stack and then goes to the new state determined by Goto field. The symbol "acc" completes parsing successfully. If an entry contains no operation, the parser will detect an error. The LR table in figure 3 has conflict actions in state 11 and 12 of column "p" with both

[2]Preterminal is a grammatical category of a terminal symbol

a shift and a reduce action. When GLR parser encounters a conflict, every conflict action is carried out simultaneously, but the vertex created by the shift action remains inactive as long as active vertex [3] exists.

GLR parsing algorithm works with a stack called *graph-structured stack (GSS)* and a LR table by pushing or popping a pair of vertices each of which corresponds to a partially parsed tree and a state.

A GSS is initialized by pushing a state 0 in stage U_0, which becomes the root vertex of the GSS with the first word w_1 as a look-ahead word (scanning word). An input sentence is parsed stage by stage from left to right thus changing the GSS.

Upon scanning the $i+1$st word w_{i+1} as a look-ahead word, the parsing algorithm carries out the following four actions in stage U_i.

Reduce : The parser pops twice the number of vertices (corresponding to the right hand side (rhs) of the production rule specified by the reduce action) from the top of the stack and then pushes a pair of vertices in the stage U_i, namely a Goto state and a partially parsed tree created by the reduce action.

Shift : Both a look-ahead word w_{i+1} and a state is pushed in U_{i+1}. The state pushed on the top of the stack is determined by the shift action. Note that the newly created state vertex in U_{i+1} is not active until there is no active vertex in U_i.

Error : The vertex with error action will be truncated.

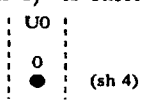
Accept : Parsing process will end with success.

In case of *Shift* and *Reduce*, a pair of vertices is pushed onto the GSS and edges are formed from the new vertex to its parents. If there exists a top vertex with the same state, they will be merged into one. The vertex after merge will have several parents. Merging vertices with the same state prevents from duplicated processing in the later stage.

Only after every vertex in the stage U_i has been processed, the parser proceeds to the stage U_{i+1} scanning the next word w_{i+2}. What kind of actions (shift; reduce; accept; error) are to be carried out is determined by the top vertices in U_{i+1}, LR table, and the preterminal of the scanning word w_{i+2}.

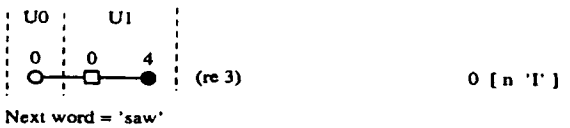
Let us consider the grammar in figure 1 and an input sentence "I saw a man in the park." GLR parsing will proceed as follows. o and □ represents a state and a partially parsed tree respectively. The • represent an active state.

At the beginning, the GSS contains only one vertex with state 0 in the stage U_0. By looking at the action table as shown in figure 3-1, the next action "shift 4 (sh 4)" is carried out.

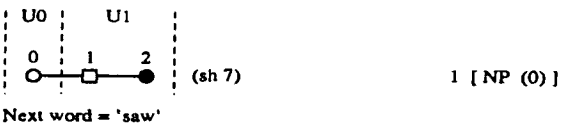


[3] "Active vertex" means there are some action(s) on the top of the stack.

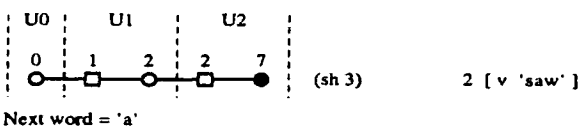
On shifting the word "I", a partially parsed tree ([n, "I"]) corresponding to the shift action is created and is pushed onto the GSS along with the state vertex. The parser enters into the stage U_1. The next action "reduce 3 (re 3)" is determined by the action part of the LR table, because the state at the top is 4 and the preterminal of the next word is "v".



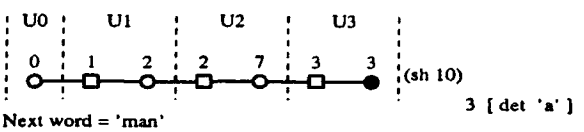
During the reduce action, a pair of vertices from the top of GSS are popped and temporarily the state of the top of GSS becomes 0, which will determine the next state by referring Goto field of NP. A new partially parsed tree (1) is created, which is pushed onto the GSS along with the state 2.



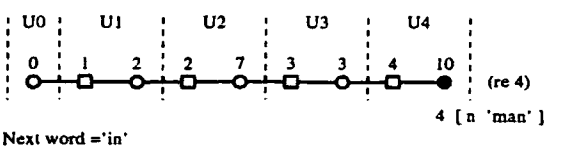
After executing shift 7, the GSS will become,



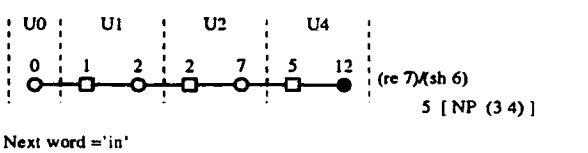
After executing shift 3, the GSS will become,



After executing shift 10 the GSS will become,

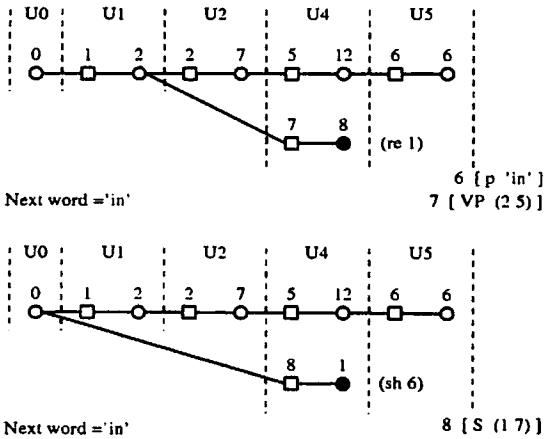


The next action reduce 4 is carried in a similar way as explained before and the resultant GSS will become,

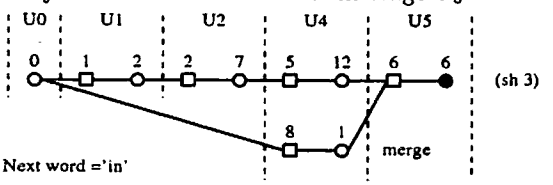


At this point, a conflict with reduce 7 and shift 6 occurs and both should be executed.

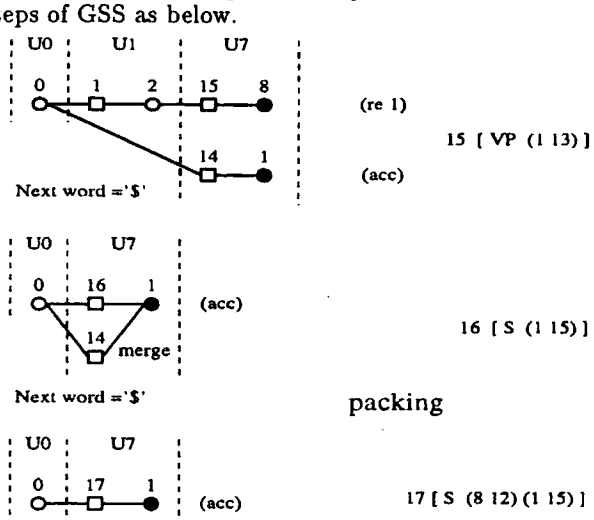
Shift 6 is executed at first and pushes a pair of vertices in stage U_5. Note that this vertex becomes inactive until there is no reduce action in stage U_4. In the GSS below, the vertex with state 8 is still active and the action reduce 1 is executed. The resultant GSS is shown below.



The action shift 6 of the vertex in stage U_4 is executed and merge operation is carried out, since there already exists the same vertices in stage U_5.



The parsing process continues in this way and shifts the words 'the' and 'park', and start recognizing the final word '$'. We skip those steps and show the final steps of GSS as below.



In the above trace, packing of trees is carried out for vertex 14 and 16 creating 17[S(8 12) (1 15)], since they share the same states immediately to its right and left. Packing suppresses the expansion of local ambiguities. If a sentence has many local ambiguity, the total ambiguity would grow exponentially. Thus packing gives efficient data structure for representing parse trees.

The Tomita's GLR parsing algorithm mentioned above is called *simplified LR (SLR)* parsing algorithm. There are other improved algorithms called *canonical LR (CLR)* and *LALR* parsing algorithm. CLR parsing algorithm is slightly efficient over SLR but, consumes more space for the LR table. LALR parsing algorithm uses more compact LR table than the one of CLR. Interested reader should refer to Aho [2] for CLR and LALR parsing algorithms.

The time complexity of Tomita's algorithm becomes $O(n^{\rho+1})$ where ρ is the length of the longest production in the grammar, and n is the length of the input sentence. If the given grammar is in the Chomsky normal form, then the time complexity will become $O(n^3)$. According to Shann [58], the efficiency of Tomita's GLR over Chart parser comes from the packed forest representation, and according to Kipps [31], inefficiency of the GLR parser is in its duplicated traversals of GSS during reduce actions.

GLR parser is also used in speech recognition researches [65, 32]. In this case, phonemes correspond to preterminals of LR table and a look-ahead preterminal/phoneme becomes an expected phoneme to be examined next in the results obtained from acoustic data and thus enable to limit the search space.

Morphological analysis of Japanese is very different from that of English, because no space is 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. Recently, Tanaka et al., proposed a method to integrate the morphological and syntactic analysis of these languages in the frame work of GLR [63]. In this method, morphological constraints are incorporated in the LR table along with syntactic constraints in CFG.

All of these algorithms mentioned above are based on breadth-first strategy and are easy to be run in parallel. The problem of GLR parsing algorithm is that it consumes a lot of memory space in the worst case. However, empirically it will not be a big problem, because of the recent rapid progress in memory integration technology.

4 ANALYSIS OF ILL-FORMED SENTENCES

As true natural language usage can be filled with many types of errors, in order to achieve a practical interactive natural language understanding system we have to provide methods for the system to recover from those errors reasonably. The system should attempt to understand what users intend to say like the way human do when they communicate to each other. Many algorithms for parsing ill-formed input are based on using chart [45, 29, 43].

Although parsing ill-formed sentences must take into account semantic and pragmatic factors, Mellish explores purely syntactic and grammar-independent techniques enabling Chart parser to recover from simple kinds of ill-formedness in textual inputs. In his method, the basic strategy is to run a bottom-up parser (BU phase) over the input and then, if this fails

to find a complete parse, to run a top-down parser (TD phase) over the resulting chart to hypothesis possible complete parses.

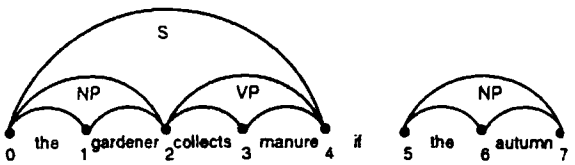


Fig. 4 Before Error Focusing

Let us explain Mellish's method briefly using figure 4. By BU phase, inactive edges, NP from 0 to 2, VP from 2 to 4 have been formed. But it fails to find a complete edge S between 0 and 7 due to the lack of inactive edge between 4 and 5. In order to find as many inactive edge as possible from 5 to 7, the BU phase skips the word between 4 and 5, forming an inactive edge NP from 5 to 7. After that, the TD phase is activated to focus possible errors as follows.

⟨Need S from 0 to 7⟩	(hypothesis)
⟨Need NP+VP from 0 to 7⟩	(by top-down rule)
⟨Need VP from 2 to 7⟩	(by fundamental rule with NP found bottom-up)
⟨Need VP+PP from 2 to 7⟩	(by top-down rule)
⟨Need PP from 4 to 7⟩	(by fundamental rule with VP found bottom-up)
⟨Need P+NP from 4 to 7⟩	(by top-down rule)
⟨Need P from 4 to 5⟩	(by fundamental rule with NP found bottom-up)

This method can efficiently parse an ill-formed sentence with one error. If multiple errors occur in the sentence, the behavior is expected to worsen dramatically. Kato [29] improved Mellish's method to process the multiple errors efficiently. Meknavin [43] applied this idea to ID/LP grammar rules to reduce the number of edges created in TD phase.

5 SCORING PARSING RESULTS BY PROBABILITIES

Generally speaking, CFG parsing algorithms explained above might produce many parsing results, out of which we want to extract most plausible parsing results for semantic processing. Recently, *probabilistic CFG (PCFG)* has been paid much attention to attach scores to parsing results.

PCFG is a set of rewriting rules with probability such as $\langle A \rightarrow \alpha, p \rangle$, where p is a probability associated with the rule. In PCFG, the following constraint must be hold: The summation of probabilities of all productions with A on its lhs is equal to 1. Figure 5 shows an example of PCFG. By the constraints of PCFG, $p1 + p2 = 1$, $p3 + p4 + p5 = 1$, $p6 = 1$, $p7 = 1$ in figure 5.

(1)	S	→	NP VP,	$p1$
(2)	S	→	S PP,	$p2$
(3)	NP	→	n,	$p3$
(4)	NP	→	det n,	$p4$
(5)	NP	→	NP PP,	$p5$
(6)	PP	→	p NP,	$p6$
(7)	VP	→	v NP,	$p7$

Fig. 5 A sample of English PCFG

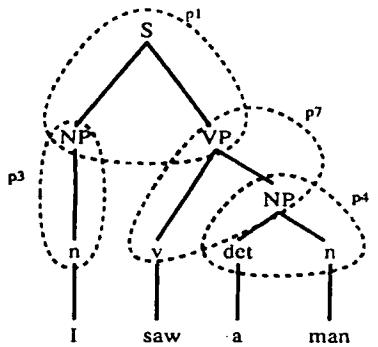


Fig. 6 Probability of a parsing tree: $p1 \times p3 \times p7 \times p4$

Resulting probability of the parsing tree in figure 6 becomes, $p1 \times p3 \times p7 \times p4$. The parsing tree with the highest score will be the most probable one.

In order to assign probability to each rule of PCFG, we have to use a set of parsing trees and count the occurrences of each rule in the parsing trees. After the normalization, we can attach a probability to each rule of PCFG. However, it is very difficult to get large volume of correct parsing trees. Lari and Young [35] give us a method for estimating probability from the given raw text corpus, which is called Inside-Outside method.

On using GLR to calculate the probability of a parse tree, it forces us to delay the calculation of probability until a reduce action is executed. Wright [68] proposed a method of GLR parsing which uses a ***probabilistic LR parsing table*** in which each action is associated with a probability. Using the action probabilities in the probabilistic LR parsing table, the probability of a partially parsed tree is calculated whenever an action is performed. Thus even in a shift action the probability is calculated. The key idea proposed by Wright is to give a method of distributing a probability in a PCFG rule to shift and/or reduce actions in a LR table.

The problem with the above method is that the attached probability to a CFG rule is always the same regardless of the context. Consider the CFG rule as NP → pronoun. This rule seems to be more frequently used in the subject position than in the object position in English. This rule should have higher probability in the subject position than object position, but PCFG is not sufficient to express this context sensitiveness [13].

Briscoe took the advantage of GLR parsing algorithm to handle mild context-sensitivity. The right context corresponds to a look-ahead preterminal. The left context corresponds to a state of the top of the

stack after popping several vertices by the reduce action. Therefore depending on the state after popping, different probability might be assigned to the same reduce action in the LR table. This reflects the mild (left and right) context-sensitivity. The probability is simply calculated by the frequency of the execution of each action (of LR table) in parsing a corpus[4]. Figure 5-3 shows a part of LR table given by Briscoe [7].

	$	det	n	pron	vt	...
0		sh 3 (0.50)		sh 2 (0.50)		
1	re 1 (0.83)					
2	re 5				re 5 (0.50)	
3			sh 4 (1.00)			
4	re 10 (3.11; (6.11)		re 10 (3.17; (5.22)		re 10 (3.11; (5.11)	
5	...	...	...	...	...	

(X.YY) X: the state after popping
YY: attached probability

Fig. 7 A part of Briscoe's LR table

6 CONCLUSION & DISCUSSIONS

There are still many interesting issues in the field of natural language parsing not mentioned above. In this section, we will discuss some of them briefly.

One of the most active issues is parallel parsing. Exploiting parallel characteristic of independent tasks, a parallel parser using a collection of processors can achieve substantial speed-up over a traditional serial parser. There are a variety of methods developed from different points of view. Some of them explore existing (serial) algorithms and try to adapt them to parallel parsing on certain hardwares or programming languages [38, 37, 62]. On the other hands, other methods design new hardwares to accommodate certain algorithms. There are also some researchers whose main interest in parallel processing is not to improve the performance of parsers but to simulate human language processing with connectionist networks [15, 25]. More references to papers on parallel parsing can be found in [48].

Parsing free word order languages is another interesting issue. Several languages, e.g. Walpiri, Japanese, German and Thai, exhibit significant word order variation. Using CFG to describe such languages is cumbersome because a numerous number of rules would be needed to enumerate all possible configurations of a constituent. Generalized Phrase Structure Grammars(GPSG) [17] provides a solution to this problem by decomposing the grammar rules into Immediate Dominance (ID) rules and Linear Precedence(LP) rules. Using ID/LP grammars, free word order languages can be easily and concisely described. However, how to combine the separate constraints together in parsing becomes a problem. One obvious way is to compile an ID/LP grammar into a CFG which can then be parsed with any existing CFG parsing algorithm. However, this approach yields a huge object grammar which can degrade the parsing performance. Shieber [59] presents an ID/LP *direct* parsing algorithm by modifying Earley's algorithm to use ID/LP grammars directly. Meknavin et al. [44] improves Shieber's algorithm by compiling ID and LP rules into generalized discrimination networks and Hasse diagrams, and using bottom-up chart algorithm instead of Earley's. Another top-down approach of direct ID/LP parsing is presented in [1].

All parsing algorithms mentioned in this paper are concerned with only CFG. As described earlier, however, there are some languages which cannot be described by CFG, called context-sensitive languages. The attempt to handle such languages have led to inventions of various grammar formalisms. Since coping with the class of fully context-sensitive grammars is difficult and may not be required in practical natural language processing because of their extraneous power, some are concerned with only so called "mildly context-sensitive grammars" (Tree adjoining grammars (TAGs) [26]. Lexical functional grammars (LFGs) [27], on the other hand, have much more power than CFGs and TAGs. Designing parsing algorithms for all of these grammars is another interesting field of researches [9].

Finally, we would like to mention about the relation between logic programming language and natural language parsing. Pereira et al., discussed about the advantages of using logic programming language such as Prolog for natural language processing. Interested reader can refer to [56, 39, 55] for top-down and bottom-up parsing in logic programming. Handling of long dependency is another difficult task in parsing, but most interested works have been based on the logic programming. Interested reader should refer to [54, 10, 9, 64, 3].

Part II: NATURAL LANGUAGE GENERATION

7 INTRODUCTION

As mentioned in Part I, natural language analysis is the process in which the system extracts the semantic meaning of the input text. The system may also be required to infer the intentions of the speaker from the text. On the other hand, natural language generation is the process of generating the text from the semantic content, which is represented in some semantic representation. The generation system may also begin with the communicative goals that the system is to achieve by means of text.

Let us see an example of the text generated by Moore's explanation generation system [47]. The system is the dialog interface of the expert system that

[4]Note that the probability attached to each action is independent of PCFG.

can suggest how to enhance a given lisp program. The system, for example, produces the sentence below to motivate the user to do an action:

> You should replace `(SETQ X 1)` to `(SETF X 1)`.

But the user may ask "Why?" instead doing the suggested action immediately. If so, the communicative goal that the system should achieve now is to persuade the user that the suggested action is a good means of enhancing the current program. Then the system generates the next utterance:

> I'm trying to enhance the maintainability of the program by applying transformations that enhance maintainability. `SETQ-to-SETF` is a transformation that enhance maintainability.

Text generation requires two functions: content planning and linguistic planning (see figure 8). The function of content planning is to decide *what to say* to achieve the given communicative goals. In the case of the second utterance above, the contents are that applying appropriate transformations can be a good methods to enhance maintainability and that the suggested action is one of the appropriate transformations. Content planning also includes content organization: the decision on ordering the fragments of the contents to maintain the coherence of the text. Coherence is an important factor that affects how easily the hearer can follow the unfolding text. The function of linguistic planning is to decide *how to say* the intended semantic contents. It includes choices of sentence structures, words, conjunctions, referring expressions, etc.

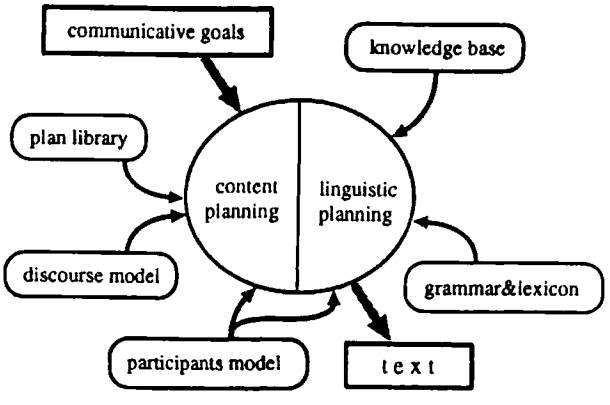


Fig. 8 A model of natural language generation

In the following sections, we overview the current status of research on text generation. Then we point out the major problems in this field, followed by future directions. In this paper, we can refer only to a handful of topics. For further references, readers would refer to, for example, the books[11, 19, 52].

7.1 The resources

Generation systems employ various kinds of resources as shown in the figure.

Plan library

An utterance can be seen as an action or complex actions to achieve speaker's communicative goals. Therefore we can apply the AI techniques of action planning to content planning. A plan library is a set of plan operators, each of which consists of the definitions of its preconditions and effects. For example, the plan operator for the action *inform* may be defined as follows [4].

Action:
$inform(Speaker, Hearer, Proposition)$
Preconditions:
$location(Speaker) = location(Hearer) \wedge$
$intend(Speaker, know(Hearer, Proposition)) \wedge$
$know(Speaker, Proposition)$
Effects:
$mutually\_know(Speaker, Hearer,$
$inform(Speaker, Hearer, Proposition))$

Discourse model

Although the content fragments of the text could be gathered using the plan library, the coherent order of the fragments should also be considered. Most systems employ some discourse model as the constraints on the coherent order of utterances. One method to maintain coherence is to extract general patterns of unfolding text from text corpus and construct some kind of content grammar. McKeown [42], Paris [51, 50] and McCoy [41] take this approach.

Another distinguishing approach is to describe the structure of text using rhetorical relations; in particular, Rhetorical Structure Theory (RST) [36] is applied in many systems [21, 22, 57]. In RST, each segment of text is related with one another in terms of their functional relation, i.e. the role of the segments, such as motivation, elaboration, etc (see figure 9).

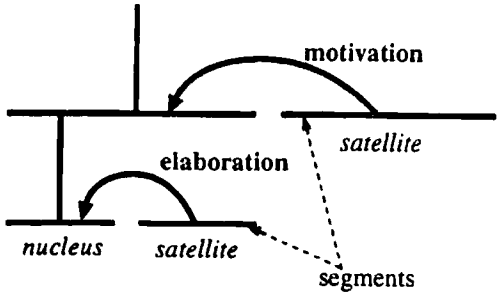


Fig. 9 An example of RST

Because of its description from the functional perspective, the definition of each rhetorical relation can be seen as a plan operator. That is, the effects of an operator correspond to the role of a segment in RST, and the subgoals correspond to constraints on the subsegments. Moore's explanation generator mentioned in the above example uses the customized RST schemata as the plan operators [47].

Another example of the system that considers both goal planning and coherence maintenance is Cawsey's EDGE system [8]. Her system uses both the content plan library and the dialog plan library.

Besides discourse model, the system should also manage focus shift appropriately to maintain coherence [60, 61]. Hovy proposes a method to apply the constraints on the patterns of focus shift to planning the rhetorical structure [21]. The focus shift constraints are also used in Moore's system.

Participants model

To decide the contents to convey, the system also needs to consider what the hearer is supposed to know, or not to know. This consideration also affects various choices on *how to say* including referring expression.

Appelt's KAMP system deals with participants knowledge model using the possible world formalism. The system plans utterances by inferring the effects of them on hearer's knowledge, which can change dynamically [4].

To manage the knowledge/belief model, it is important to consider how to obtain the current state of the hearer. A typical approach is to categorize the types of the hearers, say *expert* or *novice* [49]. For dialog system, one may take the strategy in which the system tends to be reactive to the user's queries rather than to infer the user's knowledge deliberately [47]. In this strategy, the system replies according to shallow inference and expects the user to ask follow-up questions if the system's answer is not sufficient. In Cawsey's system, the user model is changed according to the information the system obtains in dialog.

Ontological knowledge base

In generation systems, knowledge base is the term to refer to knowledge of basic facts and concepts shared by both the system and the user.

Grammar and lexicon

In linguistic planning, the system decides grammatical and lexical choices. Systemic Grammar (SG) [18] tends to be used as the description of linguistics resource because SG describes the grammatical and lexical choice points and its alternatives explicitly. SG is used in many systems [40, 53]. These systems implements SG in different ways. The Nigel grammar in Penman system is implemented as a set of procedures [40], while Patten's system implements the grammar as the production rules [53]. On the other hand, Kasper studies how to implement SG in the formalism of Functional Unification Grammar (FUG) [28], and Bateman shows how to translate SG to Typed Feature Structure (TFS) [5].

7.2 PROCESS

As stated earlier, text generation requires two functions: content planning and linguistic planning. However, this does not necessarily means these two functions are realized in separate modules and executed sequentially in this order.

The various kinds of decisions should be made during generation process and they are dependent on one another. Therefore, it is difficult to find an appropriate order of decisions. Handling dependencies among decisions has been one of the important research theme in natural language generation. The approaches to this problem can be categorized into the following groups:

(1) fix the decision order in advance of generation process,

(2) delay the decisions if necessary [46].

(3) change the decision order dynamically [20, 4],

(4) introduce the revision process [23, 24].

Many generation systems takes the first approach. Due to the interdependencies among decisions, this does not always produce a good text. The second approach is a variant of the first. It delays a decision until all the information necessary to made the decision is available. In the third approach, decision order is changed dynamically during generation process. Each decision module invoke other module by referring to the current situation. One of the drawbacks of this approach is that the control of the process tends to be complicated. The forth approach introduces revision process after the initial generation. In this approach, every decision made in the initial generation is treated as a tentative one, and they may be changed in the revision process.

8 PROBLEMS

In this section, we discuss some of the important problems that natural language generation researches are faced with.

Starting Point of Generation

We assumed that the generation system starts from communicative goal of the speaker as shown in figure 8. However, it is not the case for most of the current systems. Most of the systems start from some sort of structured object, which has system dependent information and representation. A few of them start from numerical data, such as stock market data, weather forecast data and so forth [6, 34]. Many researchers seem to have a consensus on that the start point of natural language generation should be speaker's communicative goals [12], but there is no consensus on what sort of information it should contain and how it should be represented. This is one of the crucial problem in the natural language generation research, that is, the input to the system differs in each system. This problem makes difficulties in evaluating and comparing the generation systems.

Natural language understanding research does not suffer from this problem. The start point is obviously strings (a sequence of ASCII code !!) in their case. However, they suffer from the problem in the end point of the process, that is, the question "what is the output of *understanding*?" This is the other aspect of the input problem of natural language generation. The focus of natural language understanding research shifted from morpho-syntactic analysis to semantic-discourse analysis in 1980's. Many important works in natural language generation were done in the same period. This is not a coincidence but a natural consequence.

Evaluation of Output Text

In case of natural language analysis, one of the important issues is to resolve the various kinds of ambiguities, such as lexical, syntactic, semantic ones and so on. Many theories and heuristics have been proposed for disambiguation and their performance have been evaluated. This is possible because the correct answers for the inputs are fairly easy to give, thus the categorical judgement of the correctness is easy task. In case of natural language generation, however, in order to evaluate the output text, we need a criteria to evaluate texts in general. This is very difficult because we can not judge which output is the best. There are some researches in evaluation of texts in the field of machine translation, but they are still ongoing researches and have not given remarkable results.

Large-scale Knowledge for Generation

As mentioned in section 7, natural language generation requires several kinds of resources. We need large-scale resources in order to enlarge the coverage of systems. There have been several projects to build large-scale knowledge resources for natural language processing, such as EDR. They do not always contains all the information necessary for natural language generation. They tend to be analysis-oriented knowledge resources. We need further research to find what kinds of information is necessary to produce high-quality text and to incorporate them into existing large-scale knowledge resources.

9 FUTURE RESEARCH DIRECTION

Dialog

Research on dialog has been getting more and more attention from the natural language processing community. However, much of the attention are paid to dialog analysis. In order to realize a better dialog system, we need a good generation module as well as a good analysis module. Dialog research is a rendezvous point between natural language analysis and generation. We need a tight relation between these researches.

Multi-media/Multi-modal Generation

Multi-media generation is one of the active research themes in the field of natural language generation. Many of them aim to integrate text with graphics and sound. Key issues are:

- how to select the proper media depending on the information type,
- how to synchronize the presentation of each information.

Multi-media generation is an important technology for human-machine interface, and also for dialog systems and will be more active theme in near future.

References

[1] H. Abramson. On top-down ID/LP parsing with logic grammars. *submitted for publication*, 1992.

[2] A. V. Aho and J. D Ullman. *The Theory of Parsing, Translation, and Compiling*, Vol. I & II. Prentice Hall, 1972.

[3] H. Alshawi, editor. *Core Language Engine*. The MIT Press, 1992.

[4] D. E. Appelt. Some pragmatic issues in the planning of definite and indefinite noun phrases. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics*, pp. 198–203, 1985.

[5] J. A. Bateman. The nondirectional representation of systemic functional grammars and semantics as typed feature structure. In *Proceedings of the International Conference on Computational Linguistics*, 1992.

[6] L. Bourbeau, D. Carcagno, E. Goldberg, R. Kittredge, and A. Polguère. Bilingual generation of weather forecasts in an operations environment. In *Proceedings of the International Conference on Computational Linguistics*, pp. 90–92, 1990.

[7] T. Briscoe and J. Carroll. Generalized probabilistic LR parsing of natural language (corpora) with unification-based grammars. *Computational Linguistics*, Vol. 19, No. 1, pp. 25–59, 1993.

[8] A. Cawsey. *Explanation and Interpretation*. The MIT Press, 1992.

[9] S. Crain and J. D. Fodor. How can grammars help parsers? In D.R. et al. Dowty, editor, *Natural language parsing*. Cambridge University Press, 1985.

[10] V. Dahl and H. Abramson. On gapping grammars. In *Proceedings of the Second International Logic Programming Conference*, pp. 77–88, 1984.

[11] R. Dale, E. Hovy, D. Rösner, and O. Stock, editors. *Aspects of Automated Natural Language Generation*, Vol. 587 of *Lecture Notes in Artificial Intelligence*. Springer-Verlag, 1992.

[12] R. Dale, C. Mellish, and M. Zock. Introduction. In R. Dale, C. Mellish, and M. Zock, editors, *Current Research in Natural Language Generation*, chapter 1, pp. 1–15. Academic Press, 1990.

[13] Magerman. D.M. and Marcur. M. Pearl: A probabilistic chart parser. In *Proceedings of Conference of European Chapter of the Association for Computational Linguistics*, pp. 15–20, 1991.

[14] J. Earley. An efficient augmented-context-free parsing algorithm. *Communications of the ACM*, Vol. 13, No. 1-2, pp. 95–102, 1970.

[15] M.A. Fanty. Context-free parsing in connectionist networks. Technical Report TR No. 714, Computer Science Department, University of Rochester, 1985.

[16] T. et al. Fujisaki. A probabilistic parsing method for sentence disambiguation. In M. Tomita, editor, *Current Issues in Parsing Technologies*, pp. 139–152. Kluwer Academic Publishers, 1991.

[17] G. Gazdar, E. Klein, G.K. Pullum, and Sag I.A. *Generalized Phrase Structure Grammar.* Basil Blackwell, 1985.

[18] M. A. K. Halliday. *An Introduction to Functional Grammar.* Edward Arnold, 1985.

[19] H. Horacek and M. Zock, editors. *New Concepts in Natural Language Generation.* Pinter Publishers, 1993.

[20] E. H. Hovy. *Generating Natural Language under Pragmatic Constraints.* Lawrence Erlbaum Associates, 1988.

[21] E. H. Hovy. On the study of text planning and realization. In *Proceedings of the AAAI Workshop on Text Planning and Realization*, pp. 17–29, 1988.

[22] E. H. Hovy, J. Lavid, E. Maier, V. Mittal, and C. Paris. Employing knowledge resources in a new text planner architecture. In *Aspects of Automated Natural Language Generation*, pp. 57–72. Springer-Verlag, 1992.

[23] K. Inui, T. Tokunaga, and H. Tanaka. Text revision: A model and its implementation. In R. Dale, E. Hovy, D. Rösner, and O. Stock, editors, *Aspects of Automated Natural Language Generation*, pp. 215–230. Springer-Verlag, 1992. Lecture Notes in Artificial Intelligence Vol. 587.

[24] K. Inui, T. Tokunaga, and H. Tanaka. Dependency directed unification of functional unification grammar in text generation. In *Natural Language Understanding and Logic Programming*, pp. 114–128, 1993.

[25] A. N. Jain and A. H. Waible. Parsing with connectionist networks. In M. Tomita, editor, *Current Issues in Parsing Technologies.* Kluwer Academic Publishers, 1991.

[26] A. K. Joshi. Tree adjoining grammars: How much context-sensitivity is required to provide reasonable structural descriptions? In D. R. et al. Dowty, editor, *Natural language parsing*, pp. 206–250. Cambridge University Press, 1985.

[27] R. M. Kaplan and J. Bresnan. Lexical-functional grammar: A formal system for grammatical representation. In *The Mental Representation of Grammatical Relations*, pp. 173–281. The MIT Press, 1982.

[28] R. Kasper. Systemic Grammar and Functional Unification Grammar. In *Systemic Perspective on Discourse*, chapter 9, pp. 176–199. Ablex, 1987.

[29] T. Kato. Yet another chart-based technique for parsing ill-formed input. Technical report, SIGNL 83-10, Information processing society of Japan, 1991.

[30] M. Kay. *Algorithm Schemata and Data Structures in Syntactic Processing.* pp. 35–70. Readings in Natural Language Processing. Morgan Kaufmann Publishers Inc., 1980.

[31] J.N. Kipps. GLR parsing in time $o(n^3)$. In M. Tomita, editor, *Generalized LR Parsing*, pp. 43–59. Kluwer Academic Publishers, 1991.

[32] K. Kita, T. Kawabata, and H. Saito. HMM continuous speech recognition using predictive LR parsing. In *IEEE, Proceedings of ICASSP-89*, p. S13.3, 1989.

[33] D.E. Knuth. On the translation of languages left to right. *Information and Control*, Vol. 8, No. 6, pp. 607–639, 1965.

[34] K. Kukich. Fluency in natural language reports. In D. D. McDonald and Leonard Bolc, editors, *Natural Language Generation Systems*, chapter 8, pp. 280–311. Springer-Verlag, 1988.

[35] K. Lari and S.J. Young. The estimation of stochastic context-free grammars using the inside-outside algorithm. *Computer speech and languages*, Vol. 4, pp. 35–56, 1990.

[36] W. C. Mann and S. A. Thompson. Rhetorical Structure Theory: A theory of text organization. Technical Report ISI/RR-87-190, USC-ISI, 1987.

[37] Y. Matsumoto. *Natural Language Parsing Systems based on Logic Programming.* 博士論文, Kyoto University, 1989.

[38] Y. Matsumoto and R. Sugimura. A parsing system based on logic programming. In *Proceedings of the International Joint Conference on Artificial Intelligence*, pp. 671–674, 1987.

[39] Y. Matsumoto, H. Tanaka, H. Hirakawa, H. Miyoshi, and H. Yasukawa. BUP: A bottom-up parser embedded in Prolog. *New Generation Computing*, Vol. 1, No. 2, pp. 145–158, 1983.

[40] C. Matthiessen and J. Bateman. *Text Generation and Systemic-functional Linguistics: Experiences from English and Japanese.* Printer Publishers, 1991.

[41] K. F. McCoy. Reasoning on a highlighted user model to respond to misconceptions. *Computational Linguistics*, Vol. 14, No. 3, pp. 52–63, 1988.

[42] K. R. McKeown. *Text Generation.* Cambridge University Press, 1985.

[43] S. Meknavin. *A Method of ID/LP Parsing Using Generalized Discrimination Networks.* 博士論文, Tokyo Institute of Technology, 1993.

[44] S. Meknavin, M. Okumura, and H. Tanaka. A chart-based method of ID/LP parsing with generalized discrimination networks. In *Proceedings of the International Conference on Computational Linguistics*, pp. 401–407 (Vol.1), 1992.

[45] C. S. Mellish. Some chart-based techniques for parsing ill-formed input. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics*, pp. 102–109, 1989.

[46] M. W. Meteer. *The Generation Gap: The Problem of Expressibility in Text Planning.* 博士論文, University of Massachusetts, 1990.

[47] J. D. Moore and W. R. Swartout. A reactive approach to explanation. In *Proceedings of the International Joint Conference on Artificial Intelligence*, pp. 1504–1510, 1989.

[48] A. Nijholt. Overview of parallel parsing strategies. In M. Tomita, editor, *Current Issues in Parsing Technologies*. Kluwer Academic Publishers, 1991.

[49] C. L. Paris. Descriptions strategies for naive and expert users. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics*, pp. 238–245, 1985.

[50] C. L. Paris. Tailoring object descriptions to a user's level of expertise. *Computational Linguistics*, Vol. 14, No. 3, pp. 64–78, 1988.

[51] C. L. Paris and K. R. McKeown. Discourse strategies for descriptions of complex physical objects. In G. Kempen, editor, *Natural Language Generation*, chapter 8, pp. 97–116. Martinus Nijhoff, 1987.

[52] C. L. Paris, W. R. Swartout, and W. C. Mann, editors. *Natural Language Generation in Artificial Intelligence and Computational Linguistics*. Kluwer Academic Publishers, 1991.

[53] T. Patten. *Systemic text generation as problem solving*. Cambridge University Press, 1988.

[54] F. C. N. Pereira. Extraposition Grammars. *American Journal of Computational Linguistics*, Vol. 7, No. 4, pp. 243–256, 1981.

[55] F. C. N. Pereira and S. M. Shieber. *Prolog and Natural Language Analysis*. CSLI, Stanford university, 1987.

[56] F. C. N. Pereira and D. H. D. Warren. Definite Clause Grammars for language analysis – A survey of the formalism and a comparison with Augmented Transition Networks. *Artificial Intelligence*, Vol. 13, No. 3, pp. 231–278, 1980.

[57] D. Rösner and M. Stede. Customizing RST for the automatic production of technical manuals. In *Aspects of Automated Natural Language Generation*, pp. 199–214. Springer-Verlag, 1992.

[58] P. Shann. *Experiments with GLR and Chart Parsing*, pp. 17–34. Generalized LR Parsing. Kluwer Academic Publishers, 1991.

[59] S. M. Shieber. Direct parsing of ID/LP grammars. *Linguistics and Philosophy*, Vol. 7, pp. 135–154, 1984.

[60] C. L. Sidner. *Towards a Computational Theory of Definite Anaphora Comprehension in English Discourse*. 博士論文, MIT, 1979.

[61] C. L. Sidner. Focusing in the comprehension of definite anaphora. In M. Brady and R. C. Berwick, editors, *Computational Models of Discourse*, pp. 267–330. MIT Press, 1983.

[62] H. Tanaka, S. Ishizaki, A. Uehara, and H. Uchida. Research and development of cooperation project on a machine translation system for Japan and it's neighboring countries. In *Proc. of MT summit II*, pp. 146–151, 1989.

[63] H. Tanaka, T. Tokunaga, and M. Aizawa. Integration of morphological and syntactic analysis based on LR parsing algorithm. In *Proceedings of International Workshop on Parsing Technologies*, pp. 101–109, 1993.

[64] T. Tokunaga, M. Iwayama, T. Kamiwaki, and H. Tanaka. LangLAB: A natural language analysis system. In *Proceedings of the International Conference on Computational Linguistics*, 1988.

[65] M Tomita. *An Efficient Parsing for Natural Languages*. Kluwer, Boston, Mass, 1986.

[66] C.S. Wetherell. Probabilistic languages: A review and some open questions. *Computing surveys*, Vol. 12, No. 4, pp. 361–379, 1980.

[67] T. Winograd. *Language as a Cognitive Process*, Vol. 1:Syntax. Addison-Wesley, 1983.

[68] J.H. Wright. LR parsing of probabilistic grammars with input uncertainty for speech recognition. *Computer Speech and Language*, Vol. 4, pp. 297–323, 1990.