IEICE TRANSACTIONS

on Information and Systems

VOL.E77-D
NO.10
OCTOBER 1994

The Institute of Electronics, Information and Communication Engineers
Kikai-Shinko-Kaikan Bldg., 5-8, Shibakoen 3chome, Minato-ku, Tokyo, 105 Japan

IEICE TRANS. INF. & SYST., VOL. E77-D, NO. 10 OCTOBER 1994

PAPER

A Shift First Strategy for Generalized LR Parsing

Yong-Seok LEE†, *Nonmember*, **Hideto TOMABECHI**† *and* **Jun-ichi AOE**†, *Members*

SUMMARY Tomita's parsing method (GLR) is a practical and successful parsing method for natural language. However, one difficulty in the GLR is that interleaved constraint processing of syntax and semantics in parallel is not trivial during parsing, because it uses the precompiled table for a fast real-time parsing. In this paper, we present a method which makes the GLR adaptable to interleaved parsing while making some limitation on its generality. For interleaved parsing, the conflicts of the LR parsing table must be resolved at the parse time. The shift-reduce conflict among the above conflicts is the most serious one for interleaved parsing because of the lack of knowledge for the conflict resolution at the parse time. Therefore, we concentrate on resolving a shift-reduce conflict by introducing a grammar which is called a *shift-first LR(k) grammar*. Our method for this is that the conflict resolution is delayed by the *shift-first strategy* which makes an unconditional choice of shift actions in the case of a shift-reduce conflict. Then, a delayed resolution that resolves the conflict, is made. Depending on the decision of the resolution, the *pseudo parsing*, which parses symbols in the LR parser stack, proceeds. Our experiments showed that our parser is efficient while attaining the interleaved parsing at real time.

key words: *shift-first grammar, shift-first parsing, pseudo parsing, Interleaved parsing*

1. Introduction

Currently syntax and semantics are interleaved in dominant parsing approaches [4]; i.e., semantic analysis is performed at intermediate points during the parse, thus allowing it to filter out semantically anomalous constituents before syntactic processing is completed. The purpose of the interleaved parsing is to make processing more efficient, since semantics prune those parses with no semantic interpretation before syntax can be built further on them.

Tomita's parsing method (GLR) [8], [10] is a practical and successful parsing method for natural language. However the GLR, defined as a variation on standard LR parsing [2], has the difficulty that interleaved constraint processing of syntax and semantics in parallel is not trivial during parsing because it uses the precompiled table for a fast real time parsing.

Parsing systems attempt to find a parse tree which best fits the semantic constraints for the sentence to be parsed. In order to make deterministic parsing by using the LR method, we must resolve the conflicts in the parsing action table of the LR parser. Aho [3] suggested a deterministic parsing strategy to resolve the conflicts in the parsing action table of an ambiguous grammar by using commonly accepted knowledge, such as precedence rules. He argued that, if disambiguating rules are carefully chosen, we can resolve the parsing action conflicts. Valuable research efforts on deterministic parsing were made by Shieber [7] and Pereira [6] in the area of natural language processing. They resolved the conflicts of the LR parsing table by methods which are modeled by the preference behavior of native speakers, producing only one parse tree. They showed that ambiguity problems could be resolved by using only syntactic knowledge, provided that the coverage of natural language phenomena is sufficiently small and the required domain knowledge is trivial.

The GLR overcomes the generality problem of Shieber but is extremely nondeterministic. The parallel property of the GLR's graph-structured stack (GSS) is useful not only for generalization of the parser, but also for the resolution of the conflicts of the LR parser. The conflicts in the LR table are classified into the next two classes: The first is related to ***syntactic acceptability***, which is whether a sentence to be parsed is syntactically acceptable or not, and the second is related to the ambiguity of a sentence. The conflicts that belong to the first class can be resolved by ***parallel analysis***, in which the moves that lead to failure are excluded. However, for the conflicts that belong to the second class, a strategy for their resolution must be prepared. The conflicts of the GLR have the following properties:

a) shift-shift conflict: This conflict is inevitable in natural language, in which one lexical item could postulate several different syntactic categories. This conflict is mainly related to syntactic acceptability and less to ambiguity of a sentence. The parallel analysis of the GLR provides a good method for the resolution of this conflict.

b) reduce-reduce conflict: The resolution of this conflict is important for deterministic parsing. However, this conflict can be resolved by Shieber's method [7] or by semantic checking method provided enough semantic constraints are at hand. Moreover, since this

Manuscript received April 6, 1993.
Manuscript revised February 1, 1994.

† The authors are with the Department of Information Science & Intelligent Systems, University of Tokushima, Tokushima-shi, 770 Japan.

conflict is also related to syntactic acceptability, not to ambiguity of a sentence, the parallel analysis of the GLR is also suitable for the resolution of this conflict in many cases.

c) shift-reduce conflict: This conflict is mainly related to the ambiguity of a sentence in contrast to the reduce-reduce conflict. Moreover, no knowledge for the resolution of this conflict is provided at the parse time. The parallel analysis of the GLR usually forces the number of parse trees to increase.

In this paper, we present a method for the resolution of the conflict problems of the GLR in the parse time concentrating on the shift-reduce conflict.

2. Shift-first and Pseudo Parsing

In this section, we will suggest a method to resolve shift-reduce conflicts. We shall use Aho's [1] convention to represent various symbols and strings for grammatical constraints.

Natural language is highly ambiguous in contrast to formal language, which is unambiguous. In LR parsing of natural language, in spite of its high ambiguity, there exist constituents which can be parsed without any conflict. If a strong phrase denotes a phrase that consists of these constituents, the partial constituents that belong to a strong phrase can not be combined with the partial constituents of other strong phrases to form another strong phrase. For example, see the next sample sentence:

"I saw a man in the park with a scope." (1)

In (1), if "I", "saw", "a man", "in", "the park", and "with a scope" are all strong phrases, "man in", "in the" and "park with" can never become strong phrases. This property of natural language parsing is valuable for providing the specific structural configurations. We want the above property to be used for deterministic parsing. For this purpose, shift-first parsing is defined as follows:

Shift-first parsing:

1) When a shift-reduce conflict occurs while parsing, only shift actions are selected bypassing the reduce actions, and then the bypassed reduce actions are tagged to the symbols shifted by the above shift actions. We will call this shift procedure ***δ-shift***, and the tagged reduce actions a ***delayed reduction***.

2) When a grammar string β is reduced by the rule $A \rightarrow \beta$, if the leftmost symbol of β has a delayed reduction, the reduced symbol A is also tagged by the delayed reduction, i.e., a delayed reduction is propagated. We will call this propagation ***δ-propagation***.

3) When a grammar string β is reduced by the rule $A \rightarrow \beta$, if all the symbols of β, except its leftmost symbol, do not contain a delayed reduction, the reduction $A \rightarrow \beta$ is unconditionally executed. We will call the above reduction a ***pure reduction***. If a reduction $A \rightarrow \beta$ is not a pure reduction, the decision of whether the delayed reduction is neglected must be made from the rightmost symbol which is tagged by a delayed reduction. We will call the above reduction an ***impure reduction***.

(1) S -> NP VP
(2) VP -> VP PP
(3) VP -> v* NP
(4) PP -> prep* NP
(5) NP -> NP PP
(6) NP -> det* n*
(7) NP -> n*

state	det*	n*	v*	prep*	$	NP	PP	VP	S
0	S10	S12				2			1
1					acc				
2			S5	S6			4	3	
3				S6	R1		7		
4			R5	R5	R5				
5	S10	S12				8			
6	S10	S12				9			
7				R2	R2				
8				R3,S6	R3		4		
9	R4	R4		R4,S6	R4		4		
10		S11							
11			R6	R6	R6				
12			R7	R7	R7				

Fig. 1 A sample grammar and its LR parsing table.

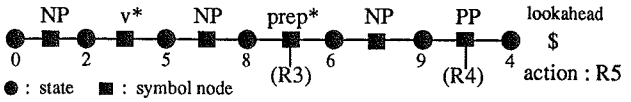


Fig. 2 The first impure reduction.

4) If the choice of a delayed reduction is made in the case of an impure reduction, the following ***pseudo parsing*** is continued after the reductions related to the delayed reduction have been done according to the shift-first parsing method.

Pseudo parsing: When a grammar string in the parser stack is $\alpha X_1 \cdots X_n$, *pseudo parsing* is the LR parsing strategy to parse the grammar string $X_1 \cdots X_n$ on the parser stack α as follows:

a) The FIRST terminal of each grammar symbol is treated as the input terminal of the standard LR parser.

b) In the case of the shift action, if the symbol to be parsed is a nonterminal, the nonterminal itself is shifted by the δ-shift and the next state is determined by the **GOTO** table of the standard LR parser, i.e. the next state =GOTO [current state, nonterminal].

c) The others follow the shift-first parsing conventions.

5) the processing of a shift-shift conflict and a reduce-reduce conflict are the same as in the GLR.

Let us use the following example to illustrate the details of the shift-first parsing.

For the parse of the sentence (1) by the grammar (Fig. 1), Fig. 2 shows the parser stack after shift-first parsing has been done by δ-shift and only pure reductions. The parser encounters an impure reduction, and a choice is made between the impure reduction R5 (NP $\rightarrow$NP PP) and the delayed reduction R4 (PP$\rightarrow$prep* NP).

Figure 3 shows the stack just after proceeding with the delayed reductdon (R4). The new stack pointer is created for the pseudo parsing. Then the pseudo parser

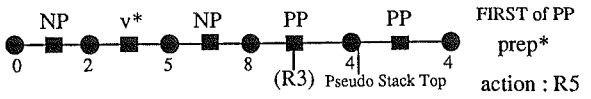


Fig. 3 After the delayed reduction.

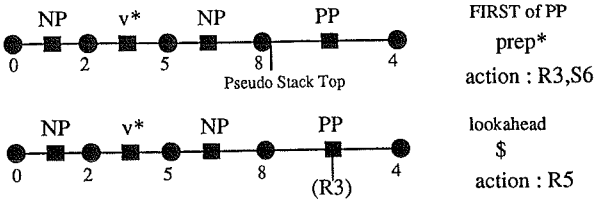


Fig. 4 After neglecting the second delayed reduction.

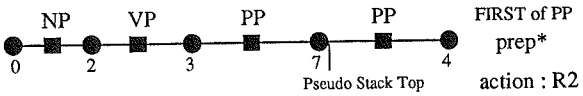


Fig. 5 After the second delayed reduction.

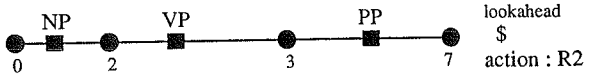


Fig. 6 The end of pseudo parsing.

meets an impure reduction R5 (NP→NP PP), and the second decision, which is the choice between the impure reduction R5 (NP→NP PP) and the delayed reduction R3 (VP→v* NP), must be made.

Figure 4 shows the stack after the pseudo parsing has been done by the choice of the impure reduction R5 in Fig. 3. A new delayed reduction R3 (VP→v* NP) is tagged to the rightmost PP during the pseudo parsing.

Figure 5 shows the stack after the pseudo parsing has been done by the choice of the delayed reduction (R3) in Fig. 3. The lookahead for the next pseudo parsing is FIRST of the rightmost PP, and the pseudo parser meets a new pure reduction R2 (VP→VP PP).

Figure 6 shows the stack just after the pseudo parsing of Fig. 5. If the decision in Fig. 3 can not be made, Fig. 4 and Fig. 6 will proceed in parallel making the parse ambiguous. The next steps of parsing will be continued with the right result.

For making a decision between an impure reduction and a delayed reduction, the knowledge such as the following could potentially be adopted: 1) unification-based augmentation; 2) frame-based semantics; 3) parameter-based pragmatics; and 4) general inference mechanisms such as discourse planning. Although a complex semantic checking method is absent, the decision could be made successfully by using simple unification based semantics and the knowledge related to the native speaker's habit postulated by Pereira [6] and Mcroy [5]. We will call the above decision making a *delayed resolution*. A delayed resolution always is strictly related to just a previous shift-reduce conflict that has to be resolved for the current viable prefix, as it was seen in the previous example. Therefore, a delayed resolution does not interfere with the correct parsing. For example, Fig. 4 shows that a new shift-reduce conflict occurs after the delayed resolution neglects the delayed reduction (VP→v* NP), and the resolution of this shift-reduce conflict is delayed for the higher level parse by the δ-shift.

The shift-first parsing has the following problems: first, all the CF grammars can not be parsed by shift-first parsing; second, there are some difficulties to be resolved to apply the shift-first parsing method to the GLR. The next two sections considers those two problems.

3. Shift-First LR(k) Grammar

A shift-first LR(k) grammar is defined as follows: Suppose a string is given to be parsed by a LR(k) parsing table of a grammar. If the parser encountered a shift-reduce conflict and it made the choice of shift-actions by the δ-shift, and if no parsing error occurs until the parser meets the first impure reduction, which needs the delayed resolution with the delayed reduction tagged by this δ-shift, the grammar is a shift-first LR(k) grammar.

The shift-first grammar has the following property: Let us assume a shift-reduce conflict occurs on stack $\sigma\alpha$, in which α is not a null string, and the stack $\sigma'\alpha'$ is the result of the reduction of the conflict. When a rule $\mathbf{A}\rightarrow\alpha\beta$ is the first impure reduction after a shift action of the conflict has been chosen and a rule $\mathbf{B}\rightarrow\alpha'\beta'\gamma$, in which β' is made by the same terminals as β, is the first reduction after a reduce action of the conflict that has been chosen, if both are the rules of a shift-first grammar, β' must be induced by β, i.e. $\beta' = *> \beta$. In other words, once input terminals are parsed to a parser stack by the shift-first strategy, the parsed stack symbols must be used for the further parsing without error.

For example, all the grammars in Fig. 7 are not shift-first LR(1) grammars. When a string *abc* is parsed by shift-first parsing, the parser fails because a parsing error will occur before the parser meets any impure reduction. All the grammars in Fig. 7 can be transformed to shift-first grammars.

Figure 8 shows the result of the transformation. The policy for the transformation is that a shift-reduce conflict can be changed into a reduce-reduce conflict by the following method:

Transformation: When a rule is $\mathbf{A}\rightarrow\alpha\beta$ and there exist a rule $\mathbf{B}\rightarrow\alpha$ and a rule $\mathbf{C}\rightarrow\mathbf{D}\beta'\gamma$, where $\mathbf{D} = *> \mathbf{B}$, if β' and β can be induced from the same terminals

S -> Abc	S -> A B c S -> a C d	S -> AB S -> aC
S -> abd	A -> a B -> b	B -> bc C -> bd
A -> a	C -> b	A -> a
(1)	(2)	(3)

Fig. 7 Some non shift-first grammars.

S -> Abc	S -> A B c S -> D C d	S -> AB S -> DC
S -> Dbd	A -> a B -> b	B -> bc C -> bd
A -> a	C -> b D -> a	A -> a D -> a
D -> a		
(1)	(2)	(3)

Fig. 8 Shift-first LR(1) grammars transformed from Fig. 7.

that are not ε, but β' can not be induced from β, add a new rule $\mathbf{E}\rightarrow\alpha$ and replace $\mathbf{A}\rightarrow\alpha\beta$ with $\mathbf{A}\rightarrow\mathbf{E}\beta$.

All CF grammars can be transformed to shift-first grammars by this method. Fortunately, if a CF grammar of natural language does not contain any lexical rule, it can trivially be written by a shift-first grammar. However, if it does include its lexical rules, it becomes very difficult to create a shift-first grammar for the language. The reason for this is that a CF grammar without lexical rules, by including its lexical rules, becomes a CF grammar that contains many rules of the same style, such as (1) and (2) of Fig. 7. Therefore, shift-first parsing is not appropriate for the parsing of a CF grammar with lexical rules.

4. Shift-First Parsing in GSS

A CF unification-based grammar for natural language contains many single productions and reduce-reduce conflicts for checking unification rules [9]. Moreover, a shift-reduce confiict could contain shift-shift conflicts in its shift and reduce-reduce conflicts in its reduction. Therefore, in order to apply the shift-first parsing to the GLR parsing, there exist the following problems to be resolved:

1) GSS construction for shift-first parsing

In the GLR parsing, the nodes with the same symbols are merged to one symbol node during the process of shift actions and ambiguity packing. However, for shift-first parsing, although the node symbols are the same, if their delayed reductions are different, they must be treated as different symbols. Moreover, in the GLR parsing, the same state nodes, which have the same shift actions, are merged to one state node after executing the reduce actions related to the same look-ahead. However, for the shift-first parsing, the state nodes with same state and shift actions become different state nodes if the nodes were made by different δ-shifts, which include the shift actions without δ-shift. Except for the above two, all the procedures of GSS construction for the shift-first parsing are the same as the GLR.

2) Reduce action in the GSS

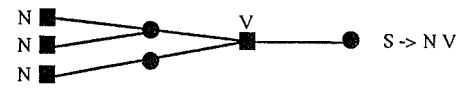


Fig. 9 A reduction in the GSS.

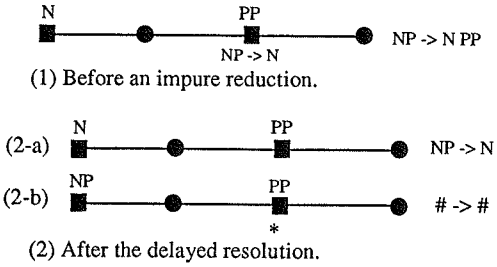


Fig. 10 Delayed resolution for a delayed resolution with single productions.

When a reduction (S→N V) is done on the GSS in Fig. 9, the reduction is equivalent to three reduce actions. Therefore, If V has a delayed reduction, the delayed resolution for the reduction (S→N V) should occur three times, i.e. the GSS in Fig. 9 must be treated as three different stacks for shift-first parsing. For this purpose, stacks for each delayed resolution should be made while searching reduced symbols for a reduction. We will call a stack related to one delayed resolution a *pseudo stack*.

3) Delayed resolution in GSS

If a reduction can be successfully reduced without consideration of any delayed reduction, this reduction is defined as a valid reduction. Figure 10 shows that a delayed reduction has a single production, and the result of the delayed reduction makes no change in the GSS structure. Since single productions do not have enough knowledge for the delayed resolution, it could be a difficult problem that a valid impure reduction is compared with a delayed reduction that has the above single productions. Therefore, in this case, the delayed resolution could become that all the possible parses are permitted. In Fig. 10, The result of the delayed resolution of (1) is (2-a) and (2-b) of (2). (2-a) is the impure reduction itself which abandons its rightmost delayed reduction and (2-b) is the result of the pseudo parsing which proceeded with the delayed reduction by using PP as the lookahead symbol. The above single production is not related to the ambiguity of a sentence, but to the syntactic acceptability. Therefore, the above delayed resolution would be successfully resolved by the parallel analysis of the GSS. However, if a delayed reduction has the productions that are not the above single productions, or there occurs a non-single production during the pseudo parsing after the processing of a single production, it is possible that the delayed resolution makes a decision for disambiguation. Therefore, if a valid impure reduction occurs, and its rightmost delayed reduction is a d-symbol, the

proceeding process is the following:

1) If the delayed reduction in the d-symbol has single productions, start the pseudo parsing which proceeds with the delayed reduction by using the d-symbol as its lookahead symbol.

2) If a non-single production, which is a valid reduction, exists in the delayed reduction, or occurs during the pseudo parsing after 1), make a decision between the impure reduction and the non single production for the delayed resolution. If the choice is the impure reduction, go to 4). If not, start or continue the pseudo parsing with the non-single production.

3) If 1) and 2) completed their pseudo parsing successfully, discard the impure reduction, and continue the parsing with the result obtained by their pseudo parsing.

4) Change the stack content by removing the delayed reduction in the d-symbol. Continue the parsing with the new stack and the impure reduction.

For example, on (1) in Fig. 11, if the delayed resolution chooses the impure reduction, the choice of (2-a) in Fig. 11 will be taken. However, in the choice of the delayed reduction, the choice of (2-b) in Fig. 11 must be taken discarding the original reduce actions in (1) of Fig. 11. If delayed resolution fails, the both will be taken.

4) Resolution of reduce-reduce conflicts

If the reduce actions in a reduce-reduce conflict are sorted in decreasing order by the length of the right side of each reduce-action rule, it is possible to check whether these reduce actions contain a valid impure reduction by testing sequentially. When there exist more than one impure reduction that is valid and non-single production by the above check, if the length of the right side of each production is deferent, the confiict of these impure reduction should be resolved by some other mechanism. Also, if there exist productions in a delayed reduction which have the above condition, the conflict resolution would be needed.

5) For more deterministic parsing

Since grammar symbols are parsed by the pseudo parsing, the information of a symbol to be parsed can be used for more deterministic parsing. For this purpose, we can make a D-FOLLOW table, also called a 'direct follow table,' as follows;

a) If there is a production $A \rightarrow \alpha B \beta$, then everything in the first grammar symbol of β except for ε is placed in D-FOLLOW(B).

b) If there is a production $A \rightarrow \alpha B$, or a production $A \rightarrow \alpha B \beta$ where FIRST(β) contains ε, then everything in D-FOLLOW(A) is in D-FOLLOW(B).

The above D-FOLLOW table might be redundant when a precise parsing table is used for parsing. However, in the case of using the LR(0) parsing table, a considerable number of redundant reduce actions are discarded by the following check: "If the lookahead symbol does not belong to the D-FOLLOW(the left symbol of a reduce rule), abandon the reduce action."

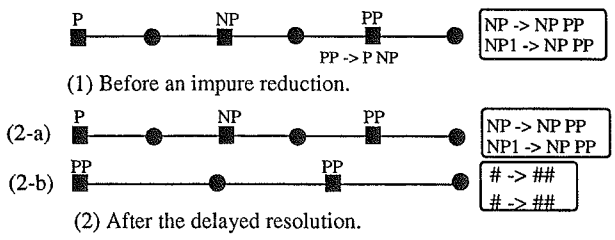


Fig. 11 Delayed resolution for a delayed resolution with non-single productions.

5. Interleaved Strategy for The GLR

The main goal of the shift-first parsing is the interleaved parsing which offers advantages for not only deterministic parsing but also efficient parsing. As seen in the previous sections, the shift-first parsing has the same result as the GLR by permitting all the delay reductions and impure reductions in parallel during the parse. There exist some problems in shift-first parsing, as same as GLR, for interleaved paring. Since actions with a shift-reduce confiict are changed into only shift-actions, the first problem of the shift-first parsing is the resolution of the shift-shift confiict and reduce-reduce conflict for interleaved parsing. If reduce-reduce conflicts and shift-shift conflicts are related to syntactic acceptability of a sentence, the parallel analysis of the GLR provides a successful method to resolve these confiicts. However, if they are related to ambiguity of a sentence, they must be resolved by interleaved parsing. A reduce-reduce conflict related to an ambiguity can be resolved at the parse time by using some knowledge including the native speaker's conventions. A shift-shift conflict related to an ambiguity, even though it is scarce, can not be resolved in parse time, and the ambiguity caused by it has to be resolved after the parse. Therefore, the shift-first parsing, the problem for interleaved parsing can be defined as that of the delayed resolution including the resolution of a reduce-reduce conflict related to ambiguity of a sentence, i.e. the problems for interleaved parsing are related only to reduce actions in the case of the shift-first parsing.

Figure 12 shows the block diagram for interleaved parsing. Syntactic processing is guided by separated

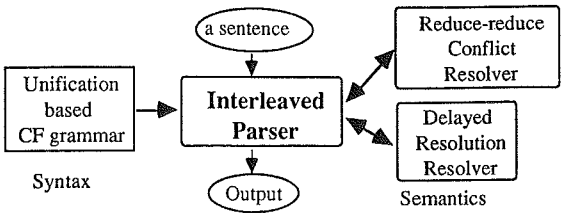


Fig. 12 An interleaved parsing strategy.

semantic handling modules at the parse time. The more detail procedure of this parser will be shown in the appendix.

6. Implementation and Discussion

Our parser, in which a unification based CF grammar has been adopted, was implemented with about 2,600 lines of CommonLisp on a Sun Sparc II Station. As the result of the implementation, if no semantic knowledge is available, i.e. all delayed resolutions fail, our parser produced the same parsing result as the GLR except for the overhead for various checking. However, by using simple semantics, which is semantic marking in a restricted domain, our parser became almost deterministic.

As seen in the previous section, using our methodology it is possible to provide semantic application during LR parsing to make the parser more deterministic. Currently our experimental parser has separate bodies for syntactic and semantic constraint postulations, which are dynamically combined only at the time of processing (during the LR parsing). Therefore, similar to other modular architectures, the maintenance of syntactic constraints and semantic (pragmatic and ontological) knowledge can be provided separately.

Table 1 shows a simple comparison with the GLR. For the comparison without semantic checking overhead, we used a random choice for a delayed resolution, and the preposition phrase attachment to increase the number of shift-reduce conflicts. This comparison shows that the execution cost of the GLR parsing, which produces many parses, is proportional to the number of shift-reduce conflicts related to ambiguity. And it also shows that the execution cost of the shift-first parsing, which produces only one parse, is proportional to the length of the string to be parsed. We can also see that the resolution of a shift-reduce conflict makes the parsing remarkably effective. Shift-First-All in Table 1 denotes the case that the shift-first parsing proceeds by permitting all delayed reductions and impure reductions.

7. Conclusion

Our parser becomes deterministic, provided that enough knowledge is available, and nondeterministic in the absence of knowledge. Also the more semantics we have, the better parsing we can achieve. Of course whether such semantic constraints are easily available is another question. One claim we would like to make here however, is that until now, even if such semantic knowledge was available, it was very difficult to use the standard GLR parser to take advantage of such knowledge. It is because in LR parsing, the LR tables are precompiled previous to the parsing and it was very difficult to do anything during the parsing since the parsing proceeds based upon the table-lookup according to the syntactically precompiled table. This was the weakness of GLR parsing compared to other methods that do everything in real-time. This weakness was inevitable in order to get the advantage in speed. Our methodology is to propose the solution to this "speed vs. inflexibility" trade off through the shift-first scheme by providing the mechanism for allowing for semantic flexibility without sacrificing the speed of GLR parsing.

Table 1 A simple comparison with the GLR.

the number of shift-reduce conflicts	the number of parses in the GLR	GLR		Shift-First		Shift-First-All	
		time	storage	time	storage	time	storage
1	2	1	1	0.8	0.6	1	1
2	5	2	2	1	0.9	2	2
3	14	3	3	1.2	1.2	4	4
4	42	4	4	1.5	1.5	8	8
5	132	5	5	1.8	1.8	16	16

References

[1] Aho, A. V. and Ullman, J. D., *The Theory of Parsing, Translation and Compiling vol. 1.*, Prentice-Hall, Englewood Cliffs, N. J., 1972.

[2] Aho, A. V. and Johnson, S. C., "LR parsing," *Computing Surveys*, vol. 6, no. 2 pp. 99-124, 1974.

[3] Aho, A. V, Johnson, S. C. and Ullman, J. D., "Deterministic parsing of ambiguous grammars," *Communication of ACM*, vol. 18, no. 8, pp. 441-452, 1975.

[4] Lytinen, S., "Dynamically combining syntax and semantics in natural language processing," *Proceedings of the Fifth National Conference on Artificial Intelligence*, Aug. 1986.

[5] Mcroy, S. W. and Girst, G., "Race-Based Parsing and Syntactic Disambiguation," *Cognitive Science*, vol. 14, pp. 313-353, 1990.

[6] Pereira, F., "A New Characterization of Attachment Preference," *Natural Language Processing. Psycholinguistic, Computational, and Theoretical Perspectives*, pp. 307-319, Cambridge, England, Cambridge University Press., 1985.

[7] Shieber, S. M., "Sentence Disambiguation by a Shift-Reduce Parsing Technique," *Proceedings of the Eighth International Joint Conference on Artificial Intelligence*, vol. 2, Aug. 1983.

[8] Tomita, M., *Efficient Parsing for Natural Language.*, Kluwer Academic Publishers, 1985.

[9] Tomita, M., "An Efficient Augmented-Context-Free Parsing Algorithm," *Computational Linguistics.*, 1987.

[10] Tomita, M. and Carbonell, J. G., "The Universal Parser Architecture for Knowledge-Based Machine Translation," *International Joint Conference on Artificial Intelligence.*, Aug. 1987.

Appendix: Interleaved Parsing Procedure for the GLR

Figure A·1 shows a ACTIVE-STACK which contains actions in the stack top. In the case of shift-reduce

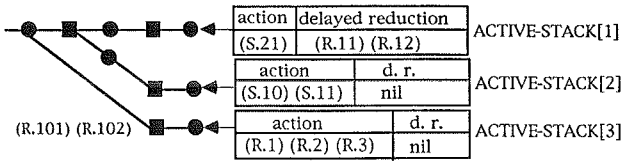


Fig. A·1 ACTIVE-STACKs.

conflict, its reduce actions are tagged to its ACTIVE-STACK, and shift actions become the current actions. The Tomita's GLR parser can be changed for shift-first parsing, making no changes to details of the method, as follows:

1) The symbol shifted by an ACTIVE-STACK with a delayed reduction is also tagged by the delayed reduction.

2) The reduce actions in an ACTIVE-STACK are supposed to be sorted in decreasing order by the length of the right side of each rule.

3) The reduce-all procedure of the GLR, which executes all the reduce actions in ACTIVE-STACK-LIST, is changed as follows:

```
FUNCTION† reduce-all (active-stack-list, lookahead);
shift-active-list <- nil;
WHILE (active-stack-list is not nil)
   a-active-stack <- car (active-stack-list);
   active-stack-list <- cdr (active-stack-list);
   IF (a-active-stack has shift actions)
      shift-active-list <- append (shift-active-list, a-active-stack);
   ELSE
      reduce-action-list <- the actions in a-stack-active;
      resolve reduce-reduce conflict of reduce-action-list, if possible;
      WHILE (reduce-action-list is not nil);
         a-reduce <- car (reduce-action-list);
         reduce-action-list <- cdr (reduce-action-list);
         (result-of-reduce, reduce-action-list) <-
            reduce-one (a-reduce, reduce-action-list,
               append (shift-active-list, active-stack-list), lookahead);
         active-stack-list <- append (active-stack-list, result-of-reduce);
return (shift-active-list);
END reduce-all;

FUNCTION†† reduce-one (a-reduce, reduce-action-list,
                       all-active-stack-list, lookahead);
result-of-reduce <- nil;
WHILE (there exists a pseudo stack in the current GSS for a-reduce)
   pseudo-stack-p <- pointers of GSS for a pseudo stack of a-reduce;
   delayed-reduction <- nil; valid-impure <- t;
   IF (the reduction related to pseudo-stack-p is a impure reduction)
      valid-impure <- check-reduce (a-reduce);
      IF ((valid-impure is not nil) or
            (valid-impure is nil and reduce-action-list is nil))
         IF (there exists a valid delayed reduction that was found by
            checking the validity from the rightmost delayed
            reduction of the pseudo-stack-p)
            d-symbol <- the symbol with the valid delayed reduction;
            delayed-reduction <- t;
            a-result-of-reduce <-
               pseudo-parse (valid-impure, pseudo-stack-p,
                             d-symbol, lookahead);
         result-of-reduce <- append (result-of-reduce,
                                     a-result-of-reduce);
         reduce-action-list <- nil;
   IF ((valid-impure is not nil) and (delayed-reduction is nil))
      a-result-of-reduce <- do the same reduction-function as the GLR
            (except for the δ-propagation), in which all-
               active-stack-list is used for ambiguity packing;
      result-of-reduce <- append (result-of-reduce, a-result-of-reduce);
return (result-of-reduce, reduce-action-list);
END reduce-one;

FUNCTION pseudo-parse (valid-impure, pseudo-stack-p,
                       d-symbol, lookahead);
active-stack-list <- create new active-stack by using d-symbol;
p-lookahead-list <- d-symbol and its right symbols;
start <- t ;
WHILE (p-lookahead-list is not nil)
   p-lookahead <- car (p-lookahead-list);
   p-lookahead-list <- cdr (p-lookahead-list);
   active-stack-list <- fill the actions of active-stack-list by using
            FIRST(p-lookahead), LR table and D-FOLLOW;
   IF (start is t)
      impure-active <- create a new active-stack that has the same state
            and actions as active-stack-list;
      set the action field of active-stack-list with the delayed reduction
         of active-stack-list, and the delayed reduction field of
            active-stack-list with nil;
      start <- nil;
   IF (valid-impure is nil) impure-active <- nil;
   active-stack-list <- p-reduce-all (impure-active, valid-impure,
                                      active-stack-list, p-lookahead);
   active-stack-list <- merge the same active-stacks in active-stack-list;
   active-stack-list <- do the same shift procedure as the GLR (except for
            δ-shift and symbol shift), in which p-lookahead
            is used as the lookahead symbol;
active-stack-list <- fill the actions of active-stack-list by using the
            current state of active-stack-list and lookahead;
return (active-stack-list);
END pseudo-parse;

FUNCTION p-reduce-all (impure-active, valid-impure,
                       active-stack-list, p-lookahead);
shift-active-list <- nil; first <- t; impure-choice <- t;
WHILE (active-stack-list is not nil)
   a-active-stack <- car (active-stack-list);
   active-stack-list <- cdr (active-stack-list);
   IF (a-stack-active has shift actions)
      shift-active-list <- append (shift-active-list, a-active-stack);
   ELSE
      reduce-action-list <- actions in a-stack-active;
      resolve reduce-reduce conflict of reduce-action-list, if possible;
      WHILE (reduce-action-list is not nil)
         a-reduce <- car (reduce-action-list);
         reduce-action-list <- cdr (reduce-action-list);
```

† The functions, such as **car, cdr** and **append**, are the same as in Lisp language.

†† This function does not consider the case that pure and impure reductions can occur concurrently in pseudo stacks for a reduce action. The function becomes more complex for this case, but that is only a problem of programming techniques.

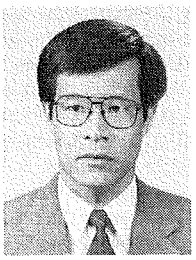
```
        valid-d-reduce <- t;
        IF ((first is t) and (impure-active is not nil))
           valid-d-reduce <- check-reduce (a-reduce);
           IF (valid-d-reduce is not nil) and (the length of
                the right side the rule for a-reduce is greater than 1)
                choice <- delayed-resolution (valid-impure,
                                              valid-d-reduce);
                first <- nil;
                IF (choice is impure-reduction)
                     return (impure-active);
                ELSE IF (choice is delayed-reduction)
                     impure-choice <- nil;
        IF (valid-d-reduce is t)
            (result-of-reduce, reduce-action-list) <-
             reduce-one (a-reduce reduce-action-list,
              append (shift-active-list, active-stack-list), lookahead);
            active-stack-list <- append (active-stack-list,
                                         result-of-reduce);
IF ((impure-choice is nil) and (shift-active-list is not nil))
     impure-active <- nil;
return (append (impure-active, shift-active-list));
END p-reduce-all;

FUNCTION delayed-resolution (valid-impure, valid-d-reduce);
make a choice between valid-impure and valid-d-reduce by using all
     the knowledge related to them;
IF (the choice is available) return ("impure" or "delayed");
ELSE IF (the choice is not available) return (nil);
END delayed-resolution ;

FUNCTION check-reduce (a-reduce) ;
return (unification values of reduced symbols and the result of
        the reduction, which is obtained by simulating a-reduce
        without consideration of any delayed reduction.);
END check-reduce;
```

Yong-Seok Lee received the B.S. degree in Electronic Engineering from the Seoul National University, Korea, in 1977, and the M.S. degree in Computer Science from the Korea Advanced Institute of Science and Technology (KAIST), in 1979. He is currently the faculty of Computer Science at the Chonbuk National University, Chonju, Korea. He is also a Ph.D. candidate of Information Science and Intelligent Systems at the University of Tokushima, Tokushima-shi, Japan. His research interests include Natural Language Processing and Knowledge Engineering.

Hideto Tomabechi was a member of the Yale AI Project, Yale Cognitive Science Program, and the Ph.D. Program in Computer Science at Yale University from 1985 to 1987. From 1987 to 1992 he was a member of the Center for Machine Translation at School of Computer Science, and the Laboratory for Computational Linguistics at Department of Philosophy at Carnegie Mellon University (CMU). He was a Visiting Research Scientist from CMU to ATR Interpreting Telephony Research Laboratories for 8 months respectively in 1990 and 1991. He joined the member of the faculty of Tokushima University in 1992 as an Assistant Professor at Department of Information Science and Intelligent Systems. He received his Ph.D. in Computational Linguistics from the Joint Ph.D. Program at CMU in 1993. He is currently an Associate Professor at Tokushima University. His research interests include multimodal natural language processing, artificial intelligence, massively parallel processing, cognitive and clinical psychology, and virtual reality.



Jun-ichi Aoe received the B.E. and M.E. degrees in electronic engineering from the University of Tokushima, Tokushima, Japan, in 1974, and 1976, respectively, and the Ph.D. degree in communication engineering from the University of Osaka, Japan, in 1980. Since 1976 he has been with the University of Tokushima. He is currently an Associate Professor in the department of Information Science and Intelligent systems at the University of Tokushima. He is the author of about 50 scientific parers. His research interests include computer algorithms in software engineering and natural language processing. Dr. Aoe is editing Computer Algorithms Series of the IEEE Computer Society Press, and two books is published: Computer Algorithms—Key Search Strategies— in 1991, and —String Pattern Matching Strategies— in 1994. He received a Best Author Award from the Information Processing Society of Japan in 1993. He is a member of the Association for Computing Machinery, The American Association for Artificial Intelligence, The Association for Computational Linguistics, the Institute of Electronics, Information and Communication Engineers of Japan, and the Information Processing Society of Japan, the Society for Software Science and Technology, the Japanese Society for Artificial Intelligence, and the Association for Natural Language Processing of Japan.