Processing Left Extraposition in a Bottom-Up Parsing System

Satoshi Kinoshita Hozumi Tanaka

Summary. In English, relative clauses and Wh-questions are constructed by left extraposition. In trace theory, we consider the phrase structure of the embedded sentence to be invariant, since an empty constituent called trace is supposed to occupy the gap made by this extraposition. In this paper, we propose a grammar formalism for dealing with left extraposition in a clear and concise manner. In addition, we describe an efficient bottom-up parsing system that uses a grammar written in the suggested formalism.

1 Introduction

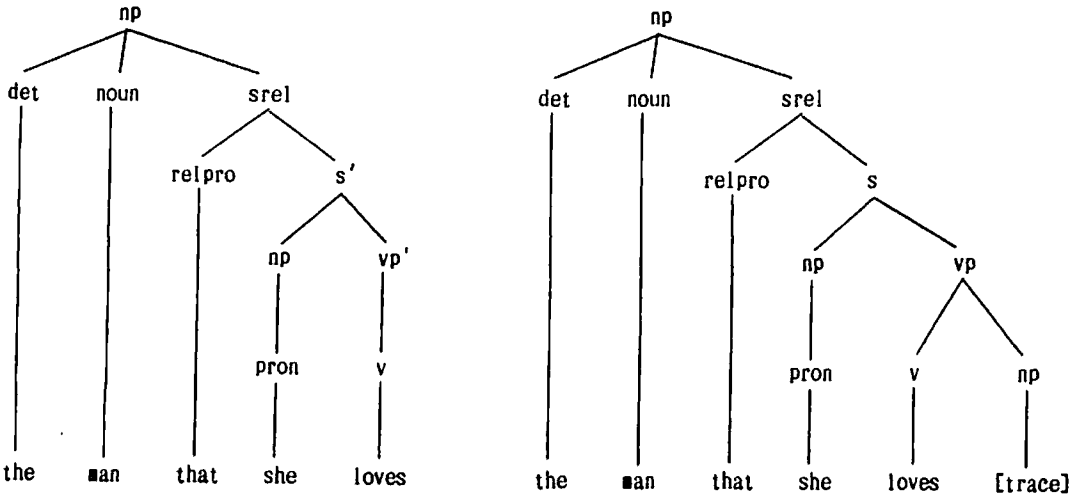
A definite clause grammar (DCG) is a grammar formalism for natural language, which is implemented, for example, in DEC-10 Prolog and C-Prolog [5]. The DCG is an extension of a context free grammar (CFG), and a DCG can be transformed directly into a Prolog program which works as a top-down parsing system. However, the derived program cannot deal with left recursive rules, since they may cause an infinite loop. To solve this problem, a parsing system called a bottom-up parser (BUP), embedded in Prolog, was developed by Matsumoto *et al.* [2,3].

Using BUP, we have been developing a middle scale English grammar which consists of about 400 DCG rules [8]. However, as the number of rules increased, the grammar gradually lost its readability, and the management became very difficult. This led us to develop a grammar formalism, an extension of DCG.

There is an important linguistic phenomenon called *left extraposition* which has been investigated in transformational grammars. Roughly speaking, left extraposition occurs when a sub-constituent of some constituent in a sentence moves to its left. This movement can been seen in constructions such as relative clauses and Wh-questions in English and other Indo-European languages. In the trace theory, an empty constituent called *trace* is proposed to occupy the gap made by the movement.

If one writes a grammar in DCG, one must write rules for every possible phrase structure. However, if a parsing system can search for a trace made by left extraposition, the number of syntactic categories and grammar rules are significantly reduced, compared to the equivalent grammar in the DCG form, and the readability of the grammar is also improved. Hereafter, we call this framework for writing grammar the *trace searching approach*, and syntactic analysis using a grammar in the framework *syntactic analysis with left extraposition.*

ISBN 0-12-037102-2



(a) The structure when we do not consider a trace (b) The structure when we take a trace into acount

Fig. 1 Phrase structure trees of a noun phrase.

This paper presents a new grammar formalism for the trace searching approach called extraposition grammar with slash category (XGS) that enables us to write a grammar more clearly than DCG. In this framework, since the parsing system searches a trace automatically, users need not write any rule for a structure made by the extraposition. Moreover, we discuss the problem of bottom-up parsing using an XGS, and propose an efficient bottom-up parsing system for it.

2 Left Extraposition and Grammar Formalisms

2.1 Left Extraposition

In English, left extraposition occurs in constructions such as relative clauses and Wh-questions. In the case of relative clauses, a noun phrase is moved from inside the embedded sentence to the position preceeding it, and usually appears as a relative pronoun at that position. So, compared to a complete declarative sentence, the structure of the embedded sentence of a relative clause lacks one noun phrase.

For example, consider the phrase structure of the noun phrase "the man that she loves." Without taking a trace into account, its phrase structure could be described as in Fig. 1(a), since the embedded sentence "she loves" and its verb phrase "loves" lack one noun phrase, the categories are named s' and vp' to distinguish them from a complete sentence and a verb phrase. However, assuming a trace, it could be described as in Fig. 1(b), the structure of the embedded sentence does not change because the trace is in the position of the extraposed noun phrase. So, the category s can be used not only for a complete declarative sentence but also for the embedded sentence of a relative clause. This means that no additional rule is necessary for parsing the embedded sentence. As a result, we could keep the grammar clear and readable.

```
s   -->  np, vp.
s   -->  np, modalp, vp.
s   -->  np, bep, adjp.
          :
np  -->  pron.
np  -->  det, noun.
          :
vp  -->  vt, np.
vp  -->  vt, np, np.
vp  -->  vi, p, np.
```

(a)

```
np  -->  det, noun, srel.   (1)
srel -->  relpro, s'.       (2)
```

(b)

```
s'  -->  np, vp'.
s'  -->  np, modalp, vp'.
          :
vp' -->  vt.
vp' -->  vt, np.
vp' -->  vi, p.
```

(c)

```
s'  -->  vp.
s'  -->  modalp, vp.
s'  -->  bep, adjp.
```

(d)

Fig. 2 Sample grammar description using a DCG.

2.2 Limitation of DCG

Before proposing the new grammar formalism, we point out the problem of the grammar description in a DCG. DCG lacks a mechanism to treat left extraposition generally. For example, suppose there is a DCG, shown in Fig. 2(a), for parsing a basic declarative sentence, and extend the grammar for parsing a relative clause.

At first, rules (1), (2) and so on, are added to the grammar as described in Fig. 2(b): rule (1) is for a noun phrase with a relative clause, and rule (2) for the relative clause. Note that the category s' is introduced for the embedded sentences to capture the fact that they lack one noun phrase compared to the declarative sentences. As a next step, rules for the category s' are added. If the noun phrase at the object position is extraposed, a new category vp' is added for an incomplete transitive verb phrase, and grammar rules for s' and vp' in Fig. 2(c) are described by refering to the rules for s and vp respectively. Furthermore, if the noun phrase at the subject position is extraposed, the rules in Fig. 2(d) are added.

Thus, it is necessary to introduce additional categories such as s' and vp' and to write rules, because DCG lacks a mechanism to treat left extraposition generally. This generalization is important for maintaining the clarity and readability of a grammar. That is, if a grammar formalism has this mechanism, those additional categories and rules are not necessary.

2.3 XGS (Extraposition Grammar with Slash Category)

As shown in Fig. 1(b), taking a trace into account, one could assign the category *s* to the embedded sentence of a relative clause, because left extraposition does not change the phrase structure. That is, if a parsing system can search for a trace and find one, one need not introduce additional grammatical categories nor write grammar rules for them. For this reason, we call this type of grammar description the *trace searching approach.*

Recently, some grammar formalisms based on this approach have been investigated. An extraposition grammar (XG) [6, 7] is an extension of the DCG formalism in which one can write a grammar for treating left extraposition with fewer rules than in DCG. However, since some XG rules have 2 non-terminals on their left-hand side, the grammar is difficult to read. Solving this problem is one of our main objectives in designing a new formalism. A generalized phrase structure grammar (GPSG) is also a grammar that falls into the trace searching type [1]. In GPSG, a grammatical category is defined as a feature set, and a special feature called *slash* denotes a category that involves a trace. Also, a grammar is defined as a set of various kinds of rules. The most interesting point is that the rules for relative clauses and interrogative sentences are generated from the rules for a basic declarative sentence by using the rules called *meta-rules.*

Here we propose a new grammar formalism, called extraposition grammar with slash category (XGS), which is an extension of the DCG; it maintains the readability of a CFG, and enables us to write grammar rules elegantly for left extraposition.

Recall the example in the previous section, in which a grammar fragment shown in Fig. 2(a) is given for analyzing a declarative sentence. In DCG, it is necessary to add rules which are shown in Figs. 2(b), 2(c) and 2(d), to analyze a relative clause. In XGS, however, one has only to add the following rules:

$$\text{np} \dashrightarrow \text{det, noun, srel../np.} \tag{3}$$

$$\text{srel} \dashrightarrow \text{relpro, s.} \tag{4}$$

The symbol "../" (called *slash*) in rule (3) is "syntactic sugar" which is added to a DCG, and the category following the slash is called a *slash category.* A compound non-terminal "a../b" denotes a trace in the phrase structure of the category *a*, which is of the slash category *b*. For example, "srel../np" in rule (3) means that there is a category *np* in the relative sentence *srel* that dominates a trace. Figure 3 illustrates an expected parse tree using rules (3) and (4): the compound non-terminal "srel../np" and the slash category that dominates the trace are connected by an arrow. This connection is called the *correspondence between a slash category and a trace.* Note that by taking a trace into account, the category of the embedded sentence of a relative clause can be regarded as *s*.

There is a constraint in making a correspondence between a slash category and a trace. For example, the following sentence is ungrammatical [6].

*The mouse that the cat that chased likes fish squeaks. (5)

In a transformational grammar, this ungrammaticality is explained by the ROSS's complex-NP constraint that forbids a noun phrase, say NP1, to be ex-

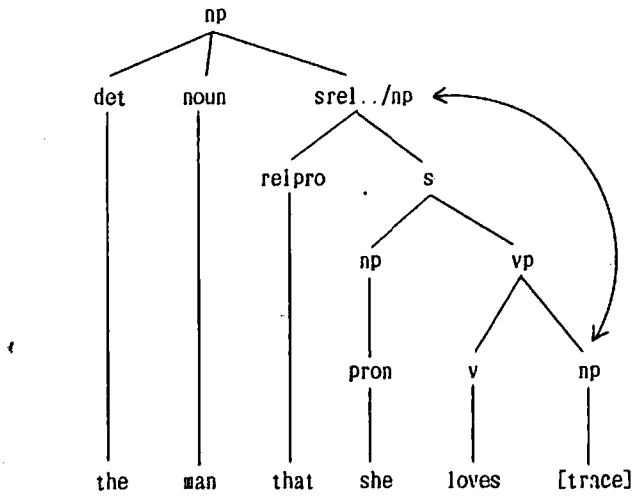


Fig. 3 Correspondence between a slash category and a trace.

traposed from a relative clause that dominates NP1 to the outside of the noun phrase that dominates the relative clause.

However, by making the correspondence shown in Fig. 4, it is possible to parse the sentence without any problem. This is because the grammar cannot designate a region where the parsing system searches for a trace. To make an appropriate correspondence, 2 symbols "⟨" and "⟩" are introduced. They are called *open* and *close* respectively, following [7].[1] Then, rule (3) could be rewritten as:

$$\text{np} \text{ --> } \text{det, noun, } \langle\text{srel../np}\rangle. \tag{6}$$

With this modification, parsing a sentence that violates the complex-NP constraint is prohibited, because a trace in the structure of *srel* can only correspond to the slash category *np*.

3 BUP-XG — Bottom-Up Parser for an XGS

This section presents an efficient bottom-up parser for an XGS, called BUP-XG, an extension of the BUP. Before its explanation, we will briefly describe the top-down parsing for an XGS.

[1] In XG, the complex-NP constraint is described as the following:

```
np --> det, noun, srel.
srel --> open, rel_marker, s, close.
open ... close --> [].
rel_marker ... trace --> rel_pronoun.
```

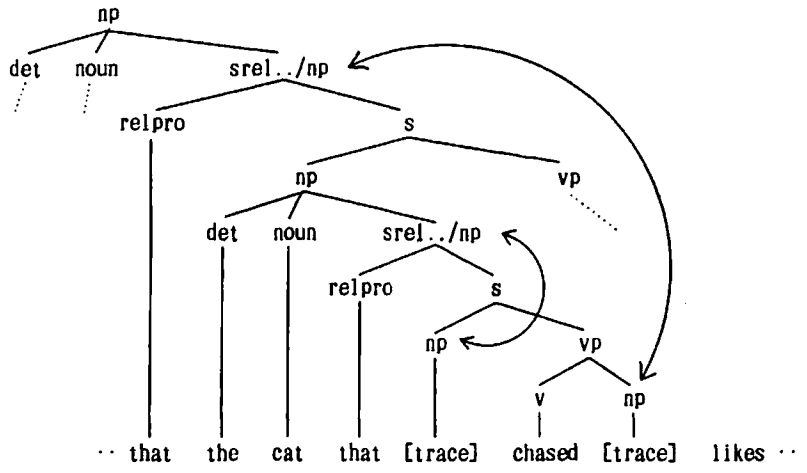


Fig. 4 Illegal sentence that violates the complex-NP constraint.

```
np --> det, noun, srel../np.                    (7)
np --> pronoun.
srel --> relpro, s.
s --> np, vp.
vp --> vt, np.
```

Fig. 5 A sample XGS.

3.1 Top-Down Parsing for an XGS

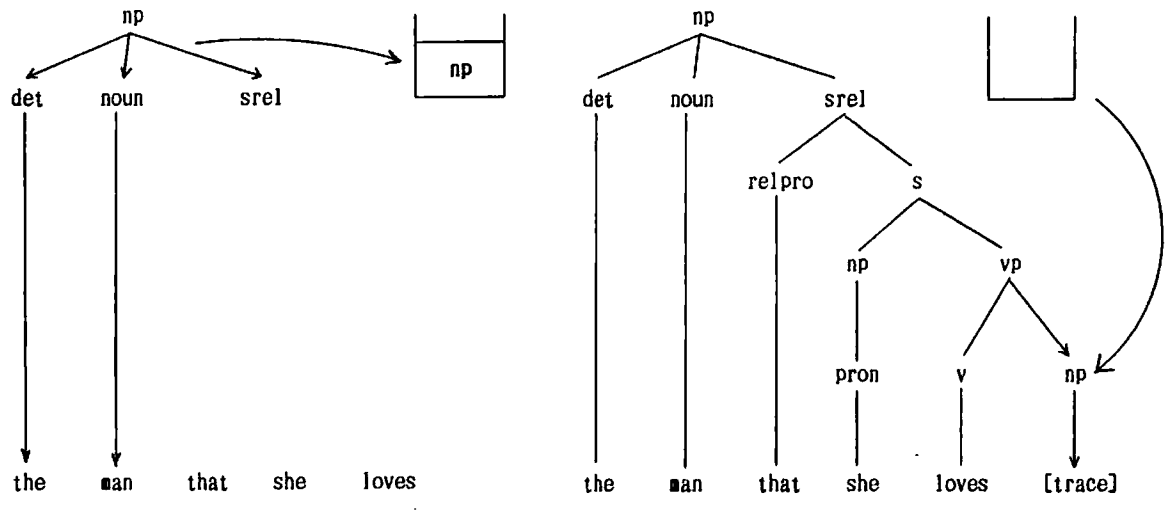
Given a grammar in a "trace searching approach", the parser must search for a trace and find one if it exists. What mechanism is necessary to realize this function?

In an XG, a stack is used for the trace search control. This could be applicable to a top-down parser for an XGS: pushing a slash category onto the stack triggers the search for a trace, and it is popped from the stack when the corresponding trace is found.

For example, let us analyze the noun phrase "the man that she loves" using the XGS fragment shown in Fig. 5.

First, using rule (7), the first two words of the phrase ("the" and "man") are recognized as a determiner and a noun, respectively, and then the next sub-goal is srel../np. At this point, the category *np* that should directly dominate a trace in *srel* is pushed onto the stack, and top-down analysis for *srel* is activated. This operation is illustrated in Fig. 6(a). During the analysis of *srel*, the system fails to find a noun phrase for the object. So, it pops the pushed category from the stack, realizes that a trace has been found, and terminates the analysis for *np* (Fig. 6(b)).

This process of the analysis corresponds to the one realized by the "Hold-Unhold" mechanism for the Hold register that is used in an ATN for the analysis of relative clauses [10]. That is, pushing a slash category *np* onto a stack corresponds



(a) *np* is pushed onto the stack, and the analysis of *srel* starts.

(b) *np* is popped from the stack, and the analysis of *np* succeeds.

Fig. 6 The expected analysis by the top-down approach.

to setting information of a noun phrase to the Hold register. Also, popping the data and terminating the analysis successfully corresponds to resetting the register.

In top-down analysis, the system can predict the position of a trace fairly well because:

1. The trace search is activated by pushing a slash category onto the stack. Therefore, searching is executed during the analysis for a category that should have a trace.
2. It is obvious when the system pops a category that dominates a trace from the stack: i.e., a slash category, say X, can only be popped when the system fails to find the constituent for the category X.

Thus, the top-down analysis seems more efficient than the naive bottom-up analysis. However, there are still essential problems with the top-down approach, such as left recursive rules.

3.2 A Problem of Bottom-Up Parsing for an XGS

One problem that occurs when using naive bottom-up parsing employing an XGS is that it is not obvious when the parser should pop the category from the stack. Since the parser cannot see a trace in the object string, it must predict the existence of a trace at every point between words whenever the stack holds a slash category. Since the analysis based on this prediction fails in most cases, the parsing is very inefficient. This is because the bottom-up parser does not predict any category to analyze next.

3.3 BUP-XG

A bottom-up parser embedded in Prolog (BUP) utilizes a bottom-up parsing algorithm with top-down prediction. Utilizing this prediction, the problem that occurs in naive bottom-up parsing in XGS is resolved.

3.3.1 Brief Introduction of BUP

In the BUP system, grammar rules and a lexicon are described in the form of DCG. The BUP translator transforms them into Prolog clauses, called *BUP clauses*.

For example, the grammar fragment in Fig. 7(a) is transformed into the Prolog clauses in Fig. 7(b): clause (g1) is the BUP clause for rule (G1), and clauses (d1) and (d2) are for lexical entries (D1) and (D2). Furthermore, some Prolog clauses called *link* clauses and *termination* clauses are generated as a result of this transformation. These Prolog clauses work as a bottom-up and depth-first parser.

The predicate *goal* which appears in clause (g1) is defined as in Fig. 8. This predicate mainly controls the bottom-up parsing. When it is activated with an input string S0 and a goal (an expected category for the string) G, the system first consults the dictionary and obtains a non-terminal symbol of the first word in the input string. Next it checks the "reachability" from the obtained category C to the current goal G using a link clause, and finally calls the goal whose predicate name is the obtained category. If the obtained category C is identical with the goal G, the termination clause terminates the process for the goal.

For example, parsing the input string "john walks" as a sentence is activated by a Prolog goal:

```
?- goal(s, _, [john,walks], [ ]).
```

Then the non-terminal *np* is obtained as a category for "john." After the

```
s --> np, vp.                                   (G1)
np --> [john].                                  (D1)
vp --> [walks].                                 (D2)
```

(a) A sample DCG

```
np(G,[ ],I) --> {link(s,G)},
                goal(vp,[ ]),
                s(G,[ ],I).                     (g1)
dict(np,[ ]) --> [john].                        (d1)
dict(vp,[ ]) --> [walks].                       (d2)
link(np,s).
link(X,X).
s(s,A,A,S,S).
np(np,A,A,S,S).
vp(vp,A,A,S,S).
```

(b) BUP clauses

Fig. 7 A sample transformation by the BUP translator.

```
goal(G,A,S0,S) :-
        dict(C,CARG,S0,S1),
        link(C,G),
        P =.. [C,G,CARG,A,S1,S],
        call(P).
```

Fig. 8 Definition of goal clause (partial).

reachability test, the BUP clause (g1) is invoked. After another reachability test, the new sub-goal to analyze the rest of the input string as *vp* is invoked. This is a top-down expectation, meaning that if the system finds the left-most non-termninal of the right-hand side of a DCG rule, the rest of the string is expected to be recognized as a sequence of other non-terminals of the rule. Thus, BUP parses an input string in a bottom-up and depth-first manner with the top-down expectation.

3.3.2 The Parsing Mechanism of BUP-XG

(A) Pushing a Slash Category onto the Stack

Utilizing the top-down expectation of BUP, BUP-XG pushes a slash category onto the stack: when a non-terminal, say a../b, is expected as a sub-goal, the slash category *b* is pushed onto the stack. However, there is one difference from the top-down approach. In BUP, since the left-most non-terminal of the right-hand side of a DCG rule is used as a key for bottom-up parsing, the top-down expectation is not carried out for the non-terminal. So, even if the left-most non-terminal has a slash category, the BUP-XG parser cannot push it onto the stack. This is a limitation of BUP-XG.

(B) Removing a Slash Category from the Stack

As described in Section 3.2, no useful information for the *pop* operation is given in naive bottom-up parsing. So, whenever the stack holds a slash category, the parser executes a pop operation, but the analysis, in most cases, ends in vain. However, in BUP-XG, since the top-down expectations are available, the prediction of the trace position is more accurate, and the number of unsuccessful analyses decreases.

In BUP-XG, there are two cases in which the parser removes a slash category from the stack. The first is when the category which is expected as the current sub-goal is on the stack top. In this case, the parser pops it up from the stack and terminates the analysis of the sub-goal immediately; this is just the same as in the top-down approach. The second is when the reachability between the category of the stack top and the current goal of analysis is attained. In this case, the parser pops the slash category, and activates a BUP clause whose predicate name is the category. These stack operations are implemented with a slight extension of BUP. This will be described in Section 4.3.

(C) Parsing Example of BUP-XG

We briefly explain here the parsing process of BUP-XG by tracing the analysis of the noun phrase "the man that loves her." We use the grammar shown in Fig. 5 (assuming that the stack is already implemented).

1. The analysis is activated by a Prolog call:

   ?- goal(np, X, [the,man,that,loves,her],[ ]).

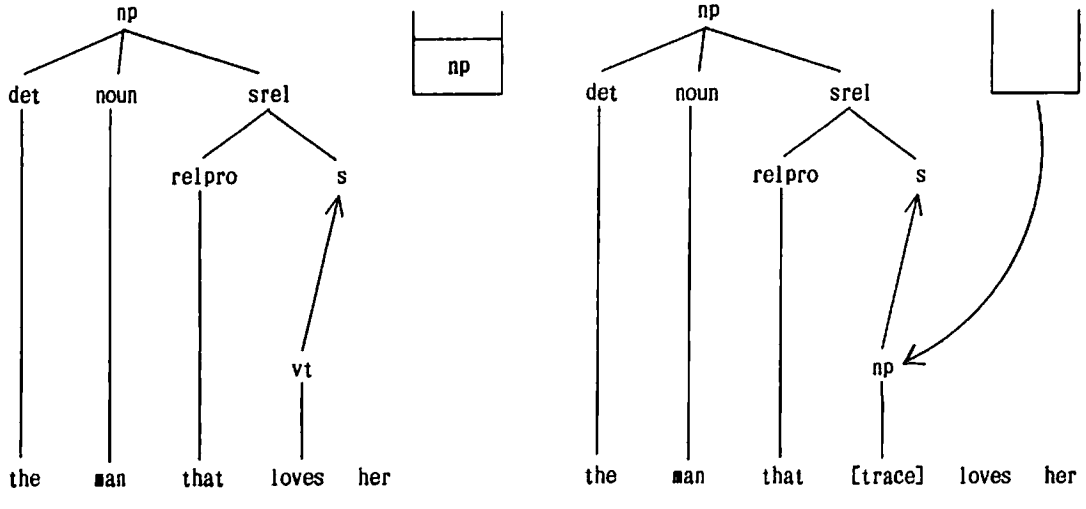
   According to its definition, the system first uses *dict* clauses, and gets the category *det* for the first word "the," and then the BUP clause that corresponds to rule (7) is activated.

2. A noun is expected as the next category, and the goal is satisfied by finding the word "man." The slash category *np* is pushed onto the stack, and the analysis for the relative clause begins.

3. By consulting the dictionary, the parser gets the category *relpro* for "that," and, using rule (8), the analysis for a sentence is activated.

4. By consulting the dictionary again, the category for "loves" is found to be *vt*. Since no sentence which begins with a verb can be derived, the reachability test fails, and the parser gives up the analysis of the sentence (Fig. 9(a)). However, since the reachability holds between the category *np*, which is at the stack top, and the current goal *s*, the parser pops the category from the stack, and, at this moment, supposes the existence of a trace and a noun phrase that dominates it (Fig. 9(b)). Then the BUP clause whose head is *np* is selected for the next execution.

5. The rest of the input string "loves her" is analyzed normally as a verb phrase (*vp*), and the analyses of *s*, *srel*, and *np* terminate successfully.



(a) No reachability found from vt to s (b) Pops np from the stack

Fig. 9 The expected analysis by BUP-XG.

4 Implementation

4.1 Stack

In BUP-XG, the stack is represented as the following structure:

x(category, argument, xlist).

where *category* is a slash category of the stack top; *argument* is the list of arguments that are attached to the category; *xlist* is the remainder of the stack. Also, an empty stack is represented by [ ]. If the current stack is bound to a variable X, the result of pushing the slash category *np* onto the stack is represented by "x(np,[ ],X)."

Furthermore, the parser must be able to check and modify the state of the stack throughout the parsing process. This is realized in just the same way as the differential list of a DCG: 2 variables, called *stack variables*, are added to every non-terminal of an XGS rule for passing the state of the stack.

For example, the XGS rule

s−−>np, vp, pp. (8)

is transformed into the DCG rule

s(X0,X3)−−>np(X0,X1), vp(X1,X2), pp(X2,X3). (9)

by the BUP-XG translator, where the first argument of every non-terminal (e.g., X0 for s) represents the stack's state before the analysis and the second one represents the stack after the analysis.

4.2 BUP-XG Translator

As described in Section 3.3, the BUP translator transforms the DCG rules into Prolog clauses, called BUP clauses. In addition to that, the BUP-XG translator adds the stack variables to every non-terminal, as described above. For example, the XGS rule

s−−>np, vp, pp.

is transformed into

```
np(G,[ ],I,X0,X1,XR) −−> {link(s,G)},
                         goal_x(vp,[ ],X1,X2),
                         goal_x(pp,[ ],X2,X3),
                         s(G,[ ],I,X0,X3,XR).
```

which is called BUP-XG clause. Note that the variables X0, X1, X2 and X3 correspond to the ones which appear in rule (9).[2] Also, the newly introduced variable XR holds the final condition of the stack, which is usually given at the beginning of parsing.

[2] Some readers may wonder whether the initial stack, which is bound to the variable X0, is neccessary in bottom-up parsing. This variable appears in the head and the final predicate of a BUP-XG clause. This is necessary for analyzing a coordinate structure which is described by a CFG rule like "vp −−> vp, conj, vp." See Section 5.1 for details.

In transformation of an XGS rule with a slash category, the translator embeds the stack operation into its BUP-XG clause. For example, the XGS rule

```
np-->det, noun, srel../np.
```

is transformed into

```
det(G,[ ],I,X0,X1,XR) --> {link(np,G)},
                          goal_x(noun,[ ],X1,X2),
                          goal_x(srel,[ ],x(np,[ ],X2),X3),
                          {depth_check(X2,X3)},
                          np(G,[ ],I,X0,X3,XR).
```

where the underlined part denotes the operation of pushing the slash category *np* to the stack. Note that the translator also added the Prolog literal "depth_check(X2, X3)." This predicate checks whether the pushed category *np* is used during the analysis of *srel* by comparing the depth of the stacks before and after its analysis.

Furthermore, the translator has to treat an XGS rule with *open* and *close* symbols. The following is an example of such an XGS rule:

```
np-->det, noun, ⟨srel../np⟩.
```

To satisfy the complex-NP constraint, the parser should not make a correspondence between a trace which appears inside *srel* and another slash category. Here is the result of this transformation:

```
det(G,[ ],I,X0,X1,XR) --> {link(np,G)},
                          goal_x(noun,[ ],X1,X2),
                          goal_x(srel,[ ],x(np,[ ],[ ]),[ ]),
                          np(G,[ ],I,X0,X2,XR).
```

To prevent an illegal correspondence, the parser has to clear the stack for an operation of *open*, push the slash category, and start the analysis of *srel*. These stack operations are embedded as the underlined *xlist*. Note also that the returning stack must be empty, and, in this case, we need not to add the Prolog literal *depth_check*. Finally, if the analysis of *srel* succeeds, the parser restores its previous state to the stack.

4.3 Modification of Predicate *goal*

Figure 10 shows a part of the definition of the predicate *goal_x*. This predicate mainly controls the bottom-up parsing, like the predicate *goal* in the BUP system. Since this first argument G is a goal category, clause (10) succeeds if the goal category and the stack top are the same. Its fourth argument X0 denotes that the stack top is removed from the stack. In the case of clause (11), the system checks the reachability from the stack top category C to the current goal G, and, if it succeeds, it removes the category from the stack, and calls the goal whose predicate name is C.

```
goal_x(G,GARG,x(G,GARG,X0),X0,A,S,S).                    (10)
goal_x(G,A,X0,X,S0,S) :-
                X0 = x(C,CARG,X1),
                C \== G,
                link(C,G),
                P =.. [C,G,CARG,A,X0,X1,X,S0,S],
                call(P).                                  (11)
```

Fig. 10 Definition of goal_x clause (partial).

5 Appraisal of the BUP-XG System

5.1 Description of an English Grammar in XGS

We have developed an English grammar in XGS, which is equivalent to an English grammar consisting of about 400 DCG rules. Table 1 shows the numbers of the grammar rules in DCG and XGS. The number of the XGS rules is about 30% less: a sharp drop in the number of the rules for Yes-No questions contributed to this.

Using XGS, the rules for Wh-questions and passive sentences can be described in the following ways.

```
swhq --> whnq, sq../obj.
sdec --> subj, bep, vp../obj.
```

Furthermore, some Yes-No questions which begin with a modal auxiliary, a tense auxiliary, or "to be" can be described as follows.

```
sq --> mopdalp, sdec../modalp.
sq --> bep, sdec../bep.
```

Here, let us discuss the problem of analyzing a coordinate structure in a relative clause. For example, the sentence

She is the girl that I love but you dislike.

has the coordinate structure of the embedded sentences. Even if the XGS rule

```
srel-->relpro, sdec, conj, sdec.                          (12)
```

Tbale 1 Comparison of the grammar size.

Purpose of rules	(1) DCG	(2) XGS	difference (1)-(2)
yes-no questions	72	7	65
verb phrases	54	46	8
relative clauses	15	7	8
⋮	⋮	⋮	⋮
Total	383	268	115

is applied, the parser cannot analyze the sentence properly, because the slash category, which is pushed before the analysis of *srel*, is popped during the analysis of the first embeded sentence "I love," and the analysis of the second fails.

To solve this problem, the stack must be reset to the previous state before the parser begins to analyze the second embeded sentence. That is, the BUP-XG clause for rule (12) must be as follows:

```
relpro(G,[ ],I,X0,X1,XR) --> {link(srel,G)},
                             goal_x(sdec,[ ],X1,X2),
                             goal_x(conj,[ ],X2,X3),
                             goal_x(sdec,[ ],X1,X2),
                             srel(G,[ ],I,X0,X3,XR).
```

Currently, special notation is introduced to describe the stack variables explicitly, and the XGS rule can be described as

srel[X0,X3] ⇒ relpo[X0,X1], sdec[X1,X2], conj[X2,X3], sdec[X1,X2].

5.2 Experiment of Syntactic Analysis

Currently, the BUP-XG system is integrated into LangLAB (A Natural Language Analysis System) [9], and the optimization for the BUP-XG clauses realized more efficient parsing than the original version. Figure 11 shows an example of syntactic analysis by the BUP-XG system. In the parse tree, you can find the symbol "t" circled, denoting a trace.

Here we discuss the parsing time of BUP and BUP-XG. Table 2 shows the result of the analysis by both systems (See Appendix for sample sentences). Because it searches for a trace during parsing, BUP-XG is expected to need more parsing time than BUP. But the result shows that the BUP-XG is as fast as BUP, and up to 6 times faster. The result also shows that, the longer the input sentence is and the more syntactic ambiguity there is, the larger the difference between parse times becomes.

Table 2 Comparison of parse time.

Sentence number	Number of words	BUP		BUP-XG	
		Number of parse trees	Parse time (sec)	Number of parse trees	Parse time (sec)
1	4	1	1.29	1	1.24
2	5	1	0.69	1	0.80
3	7	2	3.13	2	2.66
4	10	1	4.97	1	4.13
5	11	2	11.86	2	5.63
6	18	1	18.21	2	8.92
7	21	5	112.50	5	27.07
8	19	1	23.49	1	11.61
9	20	6	61.49	2	11.09

```
Input sentences
 this is the man that she loves .

No. 1    time : 267 msec
 |-sentence
    |-sdec
    |  |-sd
    |     |-np
    |     |  |-detp
    |     |     |-det
    |     |        |-demo
    |     |           |-this
    |     |-bp
    |        |-tensed_be
    |           |-be_comp
    |              |-bep
    |              |  |-be
    |              |     |-is
    |              |-pred
    |                 |-np
    |                    |-detp
    |                    |  |-det
    |                    |     |-art
    |                    |        |-the
    |                    |-nomhd
    |                    |  |-n
    |                    |     |-man
    |                    |-ncomp_t
    |                       |-srel
    |                          |-relpro
    |                          |  |-that
    |                          |-sd
    |                             |-np
    |                             |  |-ppron
    |                             |     |-she
    |                             |-vp
    |                                |-tensed_v
    |                                   |-v_comp
    |                                      |-v
    |                                      |  |-love
    |                                      |  |-suffix -- es
    |                                      |-np --(t)
    |-period
       |-.

---------------------------------
> total time :    617 msec
> tree       :    1
> wf_goal    :    17
> wf_dict    :    10
> fail_goal  :    61
---------------------------------
```

Fig. 11 A sample parsing tree.

6 Conclusions

In this paper, we proposed a new grammar formalism called XGS, an extension of DCG. This formalism enables us to describe a grammar rule for dealing with left extraposition in a very natural and clear manner. Using XGS, we developed an English grammar with about 30% fewer rules than the equivalent grammar in the DCG formalism. Next, we proposed an efficient bottom-up parser for XGS. Although the bottom-up parsing with a trace search has been considered to be inefficient, this problem is resolved by utilizing the top-down expectation mechanism of BUP.

Finally, the trace search mechanism of BUP-XG provides a device for referring

the information of a noun preceeding the relative clause in the analysis of the embedded sentence. This idea is applied to the semantic analysis of Japanese [4].

References

[1] Gazder, G., Klein, E., Pullum, G. K. and Sag, I. A. : *Generalized Phrase Structure Grammar*, Oxford, Basil Blackwell, 1985.

[2] Matsumoto, Y., Tanaka, H., Hirakawa, H., Miyoshi, H. and Yasukawa, H. : BUP: A Bottom-Up Parser Embeded in Prolog, *New Generation Computing*, Vol. 1, No. 2 (1983), pp. 145–158.

[3] Matsumoto, Y., Kiyono, M., Tanaka, H. : Facilities of the BUP Parsing System, in Dahl, V. and Saint-Dizier, P. (eds.) *Natural Language Understanding and Logic Programming*, Elsevier Science Publishers B. V. (North Holland), 1985, pp. 97–106.

[4] Okumura, M. : An Implementation of Top-Down Information Passing on BUP System, *Trans. Inf. Proc. Soc. Japan*, Vol. 29, No. 11, 1988 (in Japanese).

[5] Pereira, F. and Warren, D. : Definite Clause Grammar for Language Analysis — A Survey of the Formalism and a Comparison with Augmented Transition Networks, *Artif. Intell.*, Vol. 13 (1980), pp. 231–278.

[6] Pereira, F. : Extraposition Grammar, *Am. J. Comput. Linguist.*, Vol. 7, No. 4 (1981), pp. 243–256.

[7] Pereira, F. : Logic for Natural Language Analysis, Technical Note 275, SRI International, 1983.

[8] Tanaka, H., Takakura, S. and Konno, S. : The Development of an English Grammar on BUP System, *Proc. of Logic Programming Conference '84*, 1984 (in Japanese).

[9] Tokunaga, T., Iwayama, M., Tanaka, H. and Kamiwaki, T. : LangLAB: A Natural Language Analysis System, *Proc. of COLING '88*, 1988, pp. 655–660.

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

Appendix. Sample Sentences

1. I open the window.
2. Diagram is an augmented grammar.
3. The structural relations are holding among constituents.
4. It is not tied to a particular domain of applications.
5. Diagram analyzes all of the basic kinds of phrases and sentences.
6. This paper presents an explanatory overview of a large and complex grammar that is used in a sentence.
7. The annotations provide important information for other parts of the system that interpret the expression in the context of a dialogue.
8. For every expression it analyzes, Diagram provides an annotated description of the structural relations holding among its constituents.
9. Procedures can also assign scores to an analysis, relating some applications of a rule as probable or as unlikely.

Initially published in "Computer Software", Vol. 3, No. 2, in Japanese.

Satoshi Kinoshita
Tokyo Institute of Technology
Department of Computer Science
Ookayama 2–12–1
Meguro-ku, Tokyo
152 Japan
Present address: Toshiba Corp. R&D Center
1 Komukai-Toshiba-cho
Saiwai-ku, Kawasaki-shi
210 Japan

Hozumi Tanaka
Tokyo Institute of Technology
Department of Computer Science
Ookayama 2–12–1
Meguro-ku Tokyo
152 Japan