LangLAB : A Natural Language Analysis System

TOKUNAGA Takenobu, IWAYAMA Makoto, TANAKA Hozumi
Department of Computer Science
Tokyo Institute of Technology

KAMIWAKI Tadashi
Hitachi Research Laboratory
Hitachi Ltd.

Abstract

This paper presents a natural language analysis system LangLAB based on BUP-XG which parses with a bottom-up and depth-first strategy and has ability to handle left extraposition. We have already developed a grammar formalism XGS, which is a superset of DCG. With XGS, left extraposition phenomena is naturally expressed in grammar rules. We have also optimized BUP-XG clauses. Experiments showed that in comparison to the original BUP-XG system, the analysis sped up 10 times in the interpreter mode and 4 times in the compiled mode. The TRIE structured dictionary in LangLAB requires less memory, provides faster dictionary reference and also handles complicated idioms with versatility. Consequently, the utilization of LangLAB for practical purposes has become feasible.

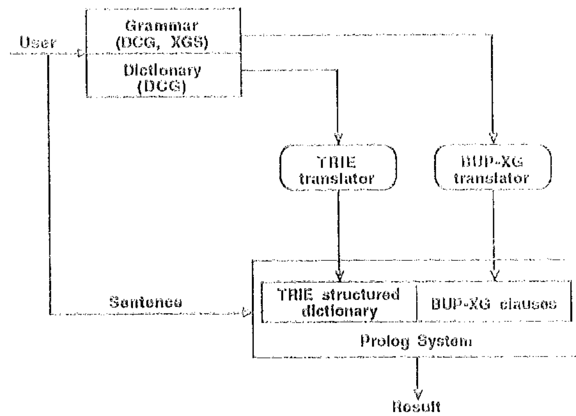

Figure 1: Structure of LangLAB

1 Introduction

So far, several grammar formalism based on logic programming paradigm such as Metamorphosis Grammar [2] and DCG [9] have been presented. In Metamorphosis Grammar, each grammar rule is translated into a Horn Clause, and the Prolog interpreter parses the input sentence with these Horn Clause using a top-down and depth-fast strategy. Unlike in the past where parsers had to be constructed for syntactic analysis, in this method, we do not have to because the Prolog interpreter itself works as one. Metamorphosis Grammar also provides a natural language processing method which interleaves syntactic analysis and semantic analysis. This is a desirable feature from the point of view of cognitive science.

Following Metamorphosis Grammar, Pereira et al. developed a grammar formalism called Definite Clause Grammar(DCG) and Extraposition Grammar(XG) [8]. The grammar rules written in DCG are also translated into a Prolog program and the Prolog interpreter works as a top-down and depth-first parser interleaving syntax analysis and semantic analysis. XG is the extended version of DCG capable of handling left extraposition.

However, top-down parser have a problem that the program falls into the infinite loop when a left recursive rule appears in the grammar rules. This problem can be solved by either translating grammar rules with left recursive rules into ones without left recursive rules or by using a bottom-up parsing strategy. Since the former solution may give unnatural parsing results, the latter is preferable.

Matsumoto of Electrotechnical Laboratory developed a system in which the grammar rules written in DCG are translated into Horn clauses called BUP clauses and Prolog interpreter works as a bottom-up and depth-first parser for these rules [14]. Matsumoto's system is called the BUP system. The BUP system can handle left recursive rules and, treat grammar rules and the dictionary separately.

Konno of Tokyo Institute of Technology extended the BUP system to BUP-XG system [5] which can handle the left extraposition phenomena elegantly. BUP-XG system introduced the grammar description form called XGS (Extraposition Grammar with Slash Category).

This paper presents a natural language analysis system LangLAB based on Konno's BUP-XG system. Figure 1 shows the structure of the LangLAB system. Users should prepare grammar rules written in XGS and a dictionary written in DCG. Both grammar rules and a dictionary are translated into BUP-XG clauses and TRIE structured dictionary respectively by translators. Translated results are consulted by the Prolog system and the Prolog interpreter works as a parser.

In chapter 2, we briefly explain the fundamentals of the BUP system and the grammar description form XGS adopted in LangLAB. We will also describe BUP-XG translator which translates the grammar written in XGS into BUP-XG clause and its optimizations. In chapter 3, we will touch on the TRIE structured dictionary adopted in LangLAB. TRIE structured dictionary requires less memory and provides faster dictionary reference and provides flexible idiom handling. In chapter 4, we shall present results of experiments verifying the effect of the optimization described in chapter 2. Experiments showed that the analysis sped up 10 times in the interpretive mode and 4 times in the compiled mode. The authors believe that LangLAB performs well enough to be of practical use.

```
s     --> np, vp.            (d-1)
np    --> pron.              (d-2)
pron  --> [you].             (d-3)
vp    --> [walk].            (d-4)
```

Figure 2: Sample grammar written in DCG

```
np(G) --> {link(np,G)},        (b-1)
   goal(vp),
   s(G).
pron(G) --> np(G).             (b-2)
dict(pron) --> [you].          (b-3)
dict(vp) --> [walk].           (b-4)
```

Figure 3: BUP clauses translated from figure 2

2 XGS and BUP–XG

In this chapter, we shall explain the grammar description form XGS adopted in LangLAB and the BUP–XG translator which translates grammar rules written in XGS into the BUP–XG clauses. Before explaining BUP–XG, we will briefly explain the mechanism of the BUP system, the predecessor of BUP–XG. Basic parsing mechanism of BUP is left-corner parsing with top-down prediction.

2.1 BUP system

In BUP system, grammar rules written in DCG (Figure 2) are translated into the rules called BUP clauses which are also of DCG format and some Prolog program (link clauses and termination clauses : explained later).

Figure 3 shows results of the translation. These BUP clauses are then translated into a Prolog program (Figure 4) by the DCG translator which is embedded in the Prolog system. Two more arguments are added to each predicate which denotes nonterminal symbol in figure 4. These arguments constitutes a difference list which represents the input string. With the special predicate **goal** which is necessary for bottom up parsing, this Prolog program can parse the input string with a bottom-up and depth-first strategy. Figure 5 shows the definition of the predicate **goal**.

Now, we shall give a step by step explanation of the parsing algorithm of the BUP system. We will use the grammar shown in figure 4 and input sentence "you walk" as an example. Calling the predicate **goal** activates the parsing process:

```
?- goal(s, [you,walk],[]).
```

```
np(G,X,Z) :- link(np,G),          (p-1)
   goal(vp,X,Y),
   s(G,Y,Z).
pron(G,X,Y) :- np(G,X,Y).         (p-2)
dict(pron,[you|X],X).             (p-3)
dict(vp,[walk|X],X).              (p-4)
```

Figure 4: Prolog programs translated from figure 3

```
goal(G,X,Y) :-                        (g-1)
  ( wf_goal(G,X,_)
    ;
    fail_goal(G,X),!,fail
   ),!,
    wf_goal(G,X,Y).
goal(G,X,Y) :-                        (g-2)
    dict(C,X,Y),
    link(C,G),
    P =.. [C,G,Y,Z],
    call(P),
    assertz(wf_goal(P)).
goal(G,X,Y) :-                        (g-3)
    assertz(fail_goal(G,X)),!,
    fail.
```

Figure 5: Definition of the goal clause

This calling checks to see if :

> A parse tree the root of which is the category "s", can be constructed from the input string denoted by the difference between the list `[you, walk]` and the list `[ ]` (`[you, walk]` in this example).

The first call of **goal** invokes the clause `(g-1)` in the figure 5. The clause `(g-1)` checks to see if the same analysis have been made before, to avoid recomputation using the information previously asserted as **wf_goal** and **fail_goal**.

As the execution of the clause `(g-1)` fails in this case, the system chooses the next clause `(g-2)`. In the body of the clause `(g-2)`, the system consults the dictionary by calling "`dict(C,[you,walk],Y)`". This predicate call picks `(p-3)` in figure 4 and the system matches "pron" with variable C and "[walk]" with variable Y.

In the second line of `(g-2)`, the system calls the predicate **link** to see if the category which is obtained by the previous dictionary consultation ("pron" in this example) can be left-corner of the current goal ("s" in this example). The **link** clauses are calculated by the BUP translator. Suppose this test succeeds, the system calls the predicate "pron" :

```
P =.. [pron,s,[walk],[]],
call(P).
```

Calling "`pron(s,[walk],[])`" invokes `(p-2)`. Then, the system executes its body that is, "`np(s,[walk],[])`".

Calling "`np(s,[walk],[])`" invokes the clause `(p-1)`. After calling the predicate **link** to check a reachability from "np" to "s", the system invokes "`goal(vp,[walk],[])`". At this point, the system has analyzed the string "you" as "np" and is predicting that the trailing string "walk" should be bundled up to the category "vp".

In the same manner, a bottom-up analysis with a top-down prediction proceeds until the execution of **goal** with the termination clauses succeeds. See [14] for the detail of the termination clauses.

Results once succeeded or failed in an analysis are asserted as **wf_goal** in the end of `(g-2)` and **fail_goal** in the clause `(g-3)` respectively. This information is used in `(g-1)` as described.

```
s      --> np, vp.                    (x-1)
np     --> pron.                      (x-2)
np     --> det, noun, s_rel../np.     (x-3)
vp     --> vt, np.                    (x-4)
s_rel  --> rel_pron, s.               (x-5)
```

Figure 6: Sample grammar written in XGS

2.2 BUP-XG system

The embedded sentence which appears in relative clauses in English can be viewed as a structure in which a noun phrase is missing from declarative sentence. A gap is formed as a result of moving the antecedent from within the declarative sentence to the left of the relative clause. Linguists call such phenomena "Left extraposition". By considering the gap left by the moved constituents as a "trace", and incorporating a mechanism that looks for such a "trace" automatically, the number of grammar rules can be decreased and the grammar rules become easier to read. Moreover incorporating such mechanism contributes to making analysis speed faster.

Top-down parsers like ATNG [13], [12] and XG [8] incorporate such a mechanism. The top-down parser can predict what category the trailing input string may be bundled up to. Efficient trace searching is possible as the system assumes the existence of traces only when a particular category is predicted as a goal.

A pure bottom-up parser is not capable of such predictions and inefficiency results because of the necessity to assume the existence of a trace between every two words. However, since the BUP system incorporates top-down prediction in the bottom-up parsing strategy described in 2.1, it is possible to implement the mechanism to look for the traces efficiently. Konno developed a BUP-XG system which incorporated such a mechanism [5].

The XGS adopted in LangLAB provides grammar writers the facility with which left extraposition can be naturally expressed in grammar rules. Figure 6 shows a small English grammar which is written in XGS.

The notation "`../`" (called "slash") in the rule `(x-3)` is introduced in XGS. This rule means that there exists the syntactic category "np" which dominates the "trace" under the syntactic category "s_rel" ("s_rel" means relative sentence). This idea is influenced by the "slash category" in GPSG [3]. We call the category after "`../`" " slash category". Rule `(x-3)` also shows that the category "np" consists of the categories "det", "noun" and "s_rel" and that the trace left behind by the left extraposition of the noun phrase consisting of "det" and "noun" is dominated by "s_rel". During an analysis, when the system finds the trace under "s_rel", as shown in figure 7, its associates the trace in the embedded sentence with the moved phrase ("the man").

XGS also provides a notation to represent "Ross's Complex NP constraint" [10]. Following is an example of this notation. This notation is called "open (<)" and "close (>)" following Pereira [8].

```
a --> b, c, <d>.
```

This rule means that category "a" consists of categories "b", "c" and "d". Open-close notation defines the scope of extraposition. This example says that the movement from under "b" or "c" to the outside of "a" is permissible, but the movement from under "d" to the outside of "a" is not. Sentences violating "Ross's Complex Np constraint" are rejected by modifying `(x-3)` to become `(x-3')`

```
np     --> det, noun, <s_rel../np>.   (x-3')
```

With `(x-3')`, the trace which is dominated by slash category "np" under "s_rel" can only correspond to the noun phrase which consists of "det" and "noun".

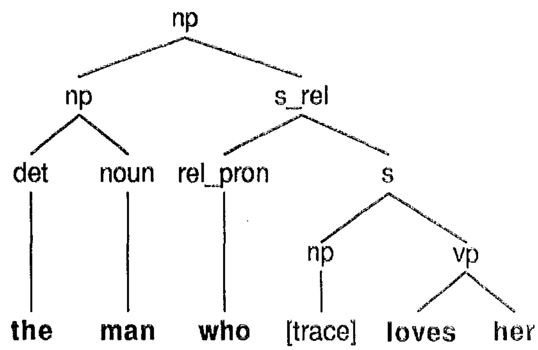

Figure 7: Matching between slash category and its trace

In addition, XGS also provides a double arrow notation (==>) and the notation to describe X lists (explained later) explicitly. With these notation, "coordinate structure" can be represented in a natural way (see [5]).

2.3 BUP-XG translator

Just like the BUP system, the grammar rules written in XGS are translated into BUP-XG clauses, link clauses and termination clauses by the BUP-XG translator. The BUP-XG translator in the LangLAB system has been improved so as to generate BUP-XG clauses more optimized than that in the original BUP-XG system. Furthermore, it is also equipped with a new function which inserts parse tree information automatically. The translator takes about three seconds to translate a grammar of about 200 rules. The following subsection explains these improvements.

2.3.1 Representation of link clauses

As the number of grammar rules increases, more link clauses are generated by the translator. For example, from about 200 grammar rules of English which we have developed, the BUP-XG translator generates about 700 link clauses. Shortening the search time of link clauses would contribute to an efficient analysis.

Link clauses are called in the body of BUP-XG clauses and in the predicate goal. Since both the arguments of link are atoms in the both cases, a link

```
link(a,b).
```

which denotes the reachablity from the category "a" to "b" can be change to the form

```
a(b) :- !.
```

This form of representation reduces the search space of the reachablity test. The BUP-XG translator in LangLAB generates link information of this form.

2.3.2 Indexes for difference list

As described in subsection 2.1 input string are represented by a difference list and intermediate analysis results are asserted with the predicate **wf_goal** and **fail_goal**. Since the last two arguments of the **wf_goal** constitutes a difference list of the input string, the longer the input string becomes, the more memory **wf_goal** consumes. By indexing difference lists, the amount of memory required is reduced, and faster reference to intermediate results is possible.

For example, when the system gets the input string "you walk", the predicates **text** are asserted as follows :

```
text(s0, []).
text(s1, [walk]).
text(s2, [you, walk]).
```

The dictionary reference program gets a difference list by calling **text** with indexes (s1,s2,...) as the key, before consulting the dictionary.

2.3.3 Representation of intermediate results

Generally, a long input string gives rise to more **wf_goals** and **fail_goals** which results in longer search time for intermediate analysis results. **Wf_goals** and **fail_goals** have as their arguments, the index to the difference list denoting the partial input string, and its analysis. As described in 2.1, **goal** first consults **wf_goals** and **fail_goals** with the indexes of input string as the key. In LangLAB system, the predicate names of intermediate analysis result are the indexes to the difference list instead of "wf_goal" or "fail_goal". This modification reduces the search space of the intermediate analysis results and speeds up the analysis process.

2.3.4 Insertion of parse tree information

Users sometimes require the results of syntactic analysis to be expressed as parse trees, and in both the BUP system and the original BUP–XG system, users are required to insert an argument in each category to accommodate parse tree information. However, it is not a difficult task to make the translator insert this information automatically. In the BUP–XG translator of LangLAB, this information is inserted automatically unless instructed otherwise. This function is similar to the one in the McCord's MLG(Modular Logic Grammar) [7]. However, unlike MLG, all the nonterminal symbols can be a node of parse trees.

2.3.5 Example of translation

Figure 8 shows the BUP–XG clauses translated from the grammar in figure 6. The variables beginning with "X" in the figure 8 are introduced to handle left extraposition. This variable is called **X list** (extraposition list) which were introduced in XG [8]. Information pertaining to slash categories is pushed into the **X list** and is then transferred from category to category during the analysis process. The predicate **goal_x** is an extended version of the predicate **goal** in the BUP system, which pops up the slash category from the **X** list when the trace is found. Note that variables for parse tree information, the names of which begin with "T", are automatically inserted and that the representation of link information (in braces) is also modified.

```
np(Goal,[T1],Info,X0,X1,XR) --->
    { s(Goal) },
    goal_x(vp,[T2],X1,X2),
    s(Goal,[[s,T1,T2]],Info,X0,X2,XR).
pron(Goal,[T1],Info,X0,X1,XR) --->
    { np(Goal) },
    np(Goal,[[np,T1]],Info,X0,X1,XR).
det(Goal,[T1],Info,X0,X1,XR) --->
    { np(Goal) },
    goal_x(noun,[T2],X1,X2),
    goal_x(s_rel,[T3],x(np,[np(t)],X2),X3),
    np(Goal,[[np,T1,T2,T3]],Info,X0,X3,XR).
vt(Goal,[T1],Info,X0,X1,XR) --->
    { vp(Goal) },
    goal_x(np,[T2],X1,X2),
    vp(Goal,[[vp,T1,T2]],Info,X0,X2,XR).
rel_pron(Goal,[T1],Info,X0,X1,XR) --->
    { s_rel(Goal) },
    goal_x(s,[T2],X1,X2),
    s_rel(Goal,[[s_rel,T1,T2]],Info,X0,X2,XR).
```

Figure 8: BUP–XG clauses translated from figure 6

```
v(info(get)) ---> [get].
v(ref(get,[[vf|ed]])) ---> [got].
v(ref(get,[[vf|en]])) ---> [gotten].
v(info(get_up)) ---> [get, up].
v(info(get_on)) ---> [get, on].
```

Figure 9: Sample dictionary including idioms

3 TRIE structured dictionary

This chapter explains the TRIE structured dictionary, another extension to the BUP–XG system and the BUP system. The TRIE structured dictionary requires less memory, provides faster dictionary reference and flexible idiom handling.

3.1 TRIE structure

The name "TRIE" is taken from "reTRIEval" [1] and it means a kind of tree structure. A dictionary written in DCG is translated into a TRIE structured dictionary by the TRIE dictionary translator. The TRIE structure is a tupple which has three elements, that is "word", "information for word(s)" and "its child TRIE structure".

For example, the dictionary written in DCG shown in figure 9 would be translated to the TRIE structured dictionary shown in figure 10.

To look up a TRIE structured dictionary, the dictionary reference program searches through the tree matching the input string with the first element of the TRIE structure and, information for the string of input is retrieved only after the last word of the input string is matched. Actually, the translator bundles up the dictionary entries which has the same first word into a clause (see how the entries "get", "get on" and "get up" are translated in figure 10). By using this structure for the dictionary, the system can avoid the

```
dicta(get,
        [[v,[info(get)]]],
        [ [on,
            [[v,[info(get_on)]]],
            []],
          [np,
            [[v,[info(get_np)]]],
            []] ] ).
dicta(got,
        [[v,[ref(get,[[vf|ed]])]]],
        [] ).
dicta(gotten,
        [[v,[ref(get,[[vf|en]])]]],
        [] ).
```

Figure 10: TRIE structure translated from figure 9

backtracking at clause level in dictionary reference, and at the same time can save memory.

In figure 9, the argument of the head is the information of this entry. The argument "info(*)" means the information of the entry "*". The argument of the entry "got" and "gotten" is a structure "ref" which denotes a pointer to the entry denoted by the first argument of "ref" (In this case, a pointer to the entry "get"). Dictionary entries the information of which only differs from each other partially, e.g. the root form and the conjugated form of an irregular verb, can be written in this manner.

The second argument of the structure "ref" is the differential information between this entry ("got" or "gotten") and the entry pointed to by the first argument of "ref". In this example, feature "vf" means "verb form" and its value "ed" and "en" means "past" and "past participle" respectively. With such a description, users do not have to write additional idiom entries which include the conjugated form of irregular verbs. In the case of regular verbs, since conjugated forms are processed by the morphological analysis program built in the dictionary reference program, idiom entries which include the conjugated form are not necessary. For example, users do not have to write the idiom entry "kicked the bucket", if the entry "kick the bucket" is written.

3.2 Idiom handling with TRIE structured dictionary

The TRIE structured dictionary can include Prolog programs to check some constraints and syntactic categories in its "word" position (first element of tupple). This feature makes it possible to handle the idioms including non-frozen elements such as "not only ~ but also ~". In the BUP system and the BUP-XG system, the system regards such idioms as a two-element word, that is a prefix terminal part and a following nonterminal part. The former part is included in the dictionary and the latter part is included in the grammar rules. In LangLAB, the TRIE structured dictionary is able to handle all such idioms as the dictionary entries.

The idiom entries which include non-frozen elements such as shown "not only ~ but also ~" can be written as figure 11.

```
adj([],[]) --> [not,only],adj(_,_),
               [but,also],adj(_,_).
np(Np,[]) --> [not,only],np(Np1,_),
              [but,also],np(Np2,_),
              {join(Np1,Np2,Np)}.
```

Figure 11: Sample dictionary with nonterminal symbols and programs in the rule body

```
dicta(not,[],[
  [only,[],[
    [[adj,_,_],[],[
      [but,[],[
        [also,[],[
          [[adj,_,_],
            [[adj,[[],[]]],[]]]]]]]]],
    [[np,Np1,_],[],[
      [but,[],[
        [also,[],[
          [[np,Np2,_],[],[
            [(join(Np1,Np2,Np)),
              [[np,[Np,[]]]],[]]]]]]]]]]]]]).
```

Figure 12: TRIE structure translated from figure 11

And figure 12 is the result of the translation.

In the case of DCG, as the idiom entry such as figure 11 is usually handled as a grammar rule, the number of grammar rules increases and inefficiency of analysis process results. It is preferable to handle grammar rules and dictionary entries separately.

As shown in figure 12, the translator converts the Prolog programs in the dictionary entry "{join(Np1,Np2,Np)}" into the form "(join(Np1,Np2,Np))". The dictionary reference program calls the program enclosed by parenthesis when it encounters such a form. In the same way, the syntactic category in the dictionary entries such as "np(Np1,_)" are converted into the list the first element of which is a category name and the rest of which are arguments of the category ([np,Np1,_]). The dictionary reference program calls the predicate **goal** (goal(np,[Np1,_],X,Y)) for such a form.

The TRIE structured dictionary enables the LangLAB system to handle idioms with versatility [4].

4 Performance Considerations

We conducted experiments to verify the effect of optimization of BUP–XG clauses. We measured the time for syntactic analysis of ten sample sentences. The experiment environment is as follows:

- Machine : Sun3/260 Workstation
- Prolog : Quintus-Prolog Release 1.6
- Grammar : 163 rules in XGS

In the experiment, we measured the time required to obtain all parse tree before and after the optimization for each

Table 1: Analysis time using interpretive code

No.	Number of Words	Number of Trees	Analysis Time [msec] (1) BUP–XG	(2) LangLAB	Ratio (1)/(2)
1	14	9	80,415	8,552	9.40
2	4	12	18,868	2,700	6.99
3	3	7	46,700	4,983	9.37
4	1	10	30,900	3,600	8.58
5	3	11	39,634	4,050	9.79
6	4	18	95,933	9,550	10.05
7	9	21	323,167	26,183	12.34
8	2	19	87,550	9,349	9.36
9	4	17	180,300	15,816	11.40
10	1	25	116,284	12,083	9.62
				average	9.69

Table 2: Analysis time using compiled code

No.	Number of Words	Number of Trees	Analysis Time [msec] (1) BUP–XG	(2) LangLAB	Ratio (1)/(2)
1	14	9	20,485	4,134	4.96
2	4	12	2,467	1,299	1.90
3	3	7	4,783	2,284	2.09
4	1	10	2,884	1,566	1.84
5	3	11	4,383	1,917	2.29
6	4	18	18,768	4,500	4.17
7	9	21	127,400	14,000	9.10
8	2	19	13,450	4,450	3.02
9	4	17	59,468	8,216	7.24
10	1	25	23,650	5,801	4.08
				average	4.07

sample sentence. This analysis does not include morphological analysis. Table 1 is the result of the experiment in the interpretive mode and table 2 is the one in the compiled mode. The fourth and the fifth column of the table is the time to analyze the sentence in the original BUP–XG system and in the LangLAB system respectively. Time is shown in millisecond.

Results showed that in comparison to the original BUP–XG system, the analysis sped up 10 times in the interpretive mode and 4 times in the compiled mode. The optimization is less effective in the compiled mode than in the interpretive mode. However, this optimization is practical because debugging is usually done in the interpretive mode. We believe that LangLAB has the capacity for practical use.

There is a related work SAX [6] by Matsumoto. SAX is also a parsing system based on logic programming, but its parsing strategy is bottom-up and breadth-first. Okunishi of ICOT reports that LangLAB is 6 ~ 10 times faster than SAX in the interpretive mode. However, in the compiled mode, SAX is 6 ~ 16 times faster than LangLAB [11]. SAX has still yet to be modified to handle idioms. If this modification is introduced, debugging can be done on LangLAB in the interpretive mode and the debugged grammar can be executed on SAX in the compiled mode.

5 Conclusion

We have made the following modification to the original BUP–XG :

- Optimized and enhanced translated code
- Adopted TRIE structured dictionary

With these modifications, the analysis sped up in comparison to the original BUP–XG system and flexible idiom handling became possible. We believe that LangLAB has become a more powerful and practical tool for natural language processing. We plan to develop a natural language processing system which includes semantic analysis, based on LangLAB.

References

[1] A. V. Aho, J. E. Hopcroft, and J. D. Ullman. *Data Structures and Algorithms.* Addison-Wesley, 1983.

[2] A. Colmerauer Metamorphosis grammar. In *Natural Language Communication with Computers*, pages 133–190, Springer-Verlag, 1978.

[3] G. Gazdar and A. F. Pullum. *Generalized Phrase Structure Grammar:A Theoretical Synopsis.* Indiana University Linguistics Club, 1982.

[4] M. Gross. Lexicon–grammar: the representation of compound words. In *COLING '86*, pages 1–6, 1986.

[5] S. Konno and H. Tanaka. Processing left–extraposition in bottom up parsing system. *Computer Software*, 3(2):115–125, 1986. (in Japanese).

[6] Y. Matsumoto and R. Sugimura. A parsing system based on logic programming. In *IJCAI '87*, pages 671–674, 1987.

[7] M. McCord. Natural language processing in prolog. In Adrian Walker, editor, *Knowledge Systems and Prolog*, chapter 5, pages 291–402, Addison-Wesley, 1987.

[8] F. Pereira. Extraposition grammar. *American Journal of Computational Linguistics*, 7(4):243–256, 1981.

[9] F. Pereira and D. Warren. Definite clause grammar for language analysis– a survey of the formalism and a comparison with augmented transition networks. *Artificial Intelligence*, 13(3):231–278, 1980.

[10] J.R. Ross. Constraints on variables in syntax. In *On Noam Chomsky: Critical Essays*, Anchor Books, 1974.

[11] T. Okunishi, et.al. Comparison of logic programming based natural language parsing systems. In *2nd International Workshop on Natural Language Understanding and Logic Programming*, pages 90–102, 1987.

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

[13] W.A. Woods. Experimental parsing system for transition network grammar. In *Natural Language Processing*, Algorithmic Press, 1971.

[14] Y. Matsumoto, et.al. Bup:a bottom–up parser embedded in Prolog. *New Generation Computing*, 1(2):145–158, 1983.