2nd. Workshop
Logic and Natural Language Understanding

Comparison of Logic Programming Based Natural Language Parsing Systems

Toshiyuki Okunishi, Ryoichi Sugimura, Yuji Matsumoto
ICOT Research Center
Institute for New Generation Computer Technology
1-4-28, Mita, Minato-ku, Tokyo, 108, Japan

Naoyoshi Tamura, Tadashi Kamiwaki, Hozumi Tanaka
Tokyo Institute of Technology
Ookayama, Meguro-ku, Tokyo, 152, Japan

ABSTRACT

This paper compares practical natural language parsing systems based on logic programming. The systems selected are BUP, LangLAB (a successor to BUP) and SAX.

There are two major aspects of comparison, system facilities and environment, and parse time and memory space needed to analyse sample English sentences using a middle scale English grammar written in DCG.

LangLAB has the most advanced facilities and environment for developing natural language grammars. It also offers grammatical formalism that can express left extrapositions. The debugging facilities of SAX are not yet well developed.

LangLAB is 5 to 11 times faster than BUP when interpreted. SAX is 6 to 16 times as efficient as LangLAB when compiled, while LangLAB is 6 to 10 times as efficient as SAX when interpreted.

Data in this paper shows that SAX in compiled mode has its upper bound in the BUP family that adopts the bottom-up algorithm with top-down prediction.

From these results, it is proposed that, at present, it is best to use both systems jointly.

1. INTRODUCTION

This paper reports a comparison of practical natural language parsing systems based on logic programming. Since the framework of logic programming is congenial to natural language processing, research on logic based natural language processing is active, and various syntactic and semantic analysis tools have been proposed and developed.

The common formalism for this research is based on *context free grammars* (CFG), and many tools for syntactic analysis have been proposed. Although they have the same base, each system has a different purpose and facilities, making it difficult to evaluate and compare each system objectively. The research summarized in this paper is a starting point for obtaining objective evaluations.

The research reported in this paper started with an investigation and comparison of three existing natural language parsing systems based on logic programming, BUP [Matsumoto 83], LangLAB (a successor to BUP) [Tanaka 86] and SAX [Matsumoto 87a] [Sugimura 86]. There are two major aspects of comparison, system facilities and environment, and parse time and memory space needed to analyse sample English sentences using a middle scale English grammar written in DCG with about 400 grammar rules and 600 dictionary entries.

Chapter 2 describes the basic mechanisms and the environment of each parsing system.

Chapter 3 gives a qualitative comparison of the three systems.

Chapter 4 gives a quantitative comparison of the three systems.

Chapter 5 describes the results of the comparisons and discusses future research.

2. OVERVIEW OF COMPARED SYSTEMS

2.1 Parsing Systems Based on Logic Programming

Definite clause grammars (DCG), proposed by Warren and Pereira [Pereira 80], can be seen as an extended formalism of CFG in the following two points.

1) Prolog programs can be written on the left hand side of rules as *extra-conditions*.
2) Grammar categories, like Prolog predicates, may have arguments for carrying information.

DCG formalism provides a representative parser embedded in Prolog. A grammar in the form of a DCG is directly transformed into a Prolog program and is executed either by the Prolog interpreter or by being compiled into machine code, without any auxiliary programs. However, DCGs have the following problems because of the pure *top-down* and *depth-first* search.

1) Handling *left-recursive rules*
2) Efficiency for large scale grammars

To overcome these problems, a parsing method called BUP (bottom up parser embedded in Prolog) has been developed by Matsumoto et al. A grammar written in a DCG is translated to a Prolog program called the *BUP program*, which parses a sentence *bottom-up* and depth-first. Several improvements on the original algorithm have given BUP the similar order of computational complexity to Earley's [Earley 70] and Pratt's [Pratt 75] algorithms.

LangLAB, developed at the Tokyo Institute of Technology, is a comprehensive natural language system based on the BUP algorithm. LangLAB has facilities and tools for helping users to develop grammars. One is an extension of grammar formalism for expressing left extrapositions, such as *extraposition grammars* (XG) [Pereira 81]. Another is an interactive tracer. Several improvements have also been made in the efficiency of both the speed and the memory space.

In another recent trend, a parallel parsing method called AX (analyser for natural language syntax) has been proposed by Matsumoto [Matsumoto 86]. AX is able to parse sentences using grammar in DCG form with a similar bottom-up strategy to BUP, but in a parallel environment, i.e., in a *breadth-first* search. AX can be seen as the parallel bottom-up version of Kay's chart-parser [Kay 80]. The target environment of AX is parallel logic programming languages such as *Guarded Horn Clauses* (GHC) [Ueda 85] and Parlog [Clark 84].

The basic idea is to translate a (restricted) DCG into a program of Prolog or parallel logic programming languages. The translated code results in an efficient code suitable for existing compiling techniques of sequential logic programming languages, such as *indexing* and *tail-recursion optimisation* (TRO). These are described in Section 2.4. Therefore, relatively efficient parsing can be expected even in sequential implementation. The AX system in a parallel language is called PAX, and that in a sequential language is called SAX.

2.2 BUP

The BUP program, which is generated from a DCG by the BUP system, is a set of *Horn clauses*, each of which corresponds to a rule in the original grammar. The BUP program parses a sentence bottom-up and depth-first. This gives rise to the following problems, which are also found in top-down and depth-first parsing of DCGs.

1) Generation of useless *partial parse trees*
2) Repetition of computations

Both problems are overcome by extending the original algorithm of BUP. Generation of useless parse trees can be reduced by combining a *top-down prediction* with bottom-up parsing. The BUP system represents this information as the predicate 'link' that works in the same way as Pratt's *'oracle'*.

Depth-first parsing has a problem in efficiency, where preceding computations are forgotten at every *backtrack*. This problem is generally solved by introducing a *tabular method*, where the partial parsing results are saved and re-used.

In the BUP algorithm, both success and failure of predicate 'goal' are registered by *side-effects*. These two improvements on the original algorithm give BUP a similar order of computational complexity to Earley's and Pratt's algorithms.

The BUP algorithm is also the basis of LangLAB. Since a detailed parsing algorithm of BUP appeared in [Matsumoto 83], only a brief explanation and example are given here. Suppose there is a DCG grammar shown in (1). The BUP system generates Prolog clauses shown in (2) for this grammar. (3) is the definition of the predicate 'goal' which is invoked from (2).

```
(1) sentence(sentence(NP,VP)) --> np(NP), vp(VP).
    np(np(you)) --> [you].
    vp(vp(walk)) --> [walk].

(2) link(X,X).
    link(np,s).
    np(np,A,B,B,A).
    vp(vp,A,B,B,A).
    sentence(sentence,A,B,B,A).

    np(Goal,[NP],S1,S,Arg) :-
        link(sentence,Goal),
        goal(vp,[VP],S1,S2),
        sentence(Goal,[sentence(NP,VP)],S2,S,Arg).
    dict(np,[np(you)],[you|S0],S0).
    dict(vp,[vp(walk)],[walk|S0],S0).

(3) goal(CurGoal,Arg,S0,S) :-
        (wf_goal(CurGoal,_,S0,_), !, wf_goal(CurGoal,Arg,S0,S) ;
        fail_goal(CurGoal,S0),!,fail).
    goal(CurGoal,Arg,S0,S) :-
        dict(Nt,Arg1,S0,S1),
        link(Nt,CurGoal),
        Pred =.. [Nt,CurGoal,Arg1,S1,S,Arg],
        call(Pred),
        assertz(wf_goal(CurGoal,Arg,S0,S)).
    goal(CurGoal,Arg,S0,S) :-
        (wf_goal(CurGoal,_,S0,_) ;
        assertz(fail_goal(CurGoal,S0)) ), !, fail.
```

The sentence 'you walk' is parsed using this program. Parsing a sentence means that a category of the input string is recognised as 'sentence'. This corresponds to the success of the procedure 'goal(sentence,Tree,[you,walk],[])'. First, a check is made to determine whether the predicate 'goal' has already been executed. Since it has not yet been executed, 'goal' tries to consult the dictionary. The predicate 'dict' recognizes that 'you' is a word with category 'np'. A check is made to determine whether the category 'np' can be the left-most descendant of 'sentence' using the predicate 'link', a top-down prediction, so as not to generate useless partial trees. When the predicate 'link' succeeds, a partial parse tree 'np', the daughter of which is 'you', has been formed. Next, the program tries to compose a larger parse tree with 'np' as its leftmost category. This corresponds to invoking the procedure 'np(sentence,Arg1,[walk],S,Arg)'. The predicate 'np' checks whether the category that will be composed by the success of this rule (in this case, 'sentence') is linked to the current goal 'Goal' (it is also instantiated to 'sentence' in this case). Next, the predicate 'np' tries to obtain 'vp'. This corresponds to invoking the procedure 'goal(vp,[VP],[walk],S2)'.

The rest of the parsing is done in the same way. Every result of 'goal' is saved by the Prolog predicate 'assertz', preventing repetition of computations caused by the backtrack mechanism of Prolog.

2.3 LangLAB

LangLAB is a tool for building natural language processing systems based on Prolog. Since LangLAB follows the method used in BUP, the problem of left-recursive rules does not arise. LangLAB has two major extensions to the original BUP, which enable the user to handle *left extrapositions* and idioms.

Left extraposition is the *movement* of a phrase that typically appears in constructs such as relative clauses and yes-no questions. If the grammar is written in the form of *extraposition grammar with slash category* (XGS) [Konno 86], left extrapositions are processed automatically in LangLAB. Grammar written in XGS is translated to a Prolog program which is called the BUP-XG program. This translator can insert variables showing the parse tree into the BUP-XG program automatically. (1) is an example of an XGS rule.

```
(1) np(Np_syn, Np_sem) -->
        det(Det_syn, Det_sem),
        noun(Noun_syn, Noun_sem),
        relative_sentence(Srel_syn, Srel_sem) ../ np(Noun_syn, Noun_sem).
    relative_sentence(Srel_syn, Srel_sem) -->
        relative_pronoun(Relpro_syn, Relpro_sem),
        declare_sentence(Sdec_syn, Sdec_sem).
```

In the above rules, 'relative_sentence(...)../np(...)' is the *slash category* which indicates the existence of a *trace* 'np' within the mother category of 'relative_sentence'.

The notation of XGS constributes to decrease the number of grammar rules The key mechanism for processing XGS is a trace operation. In XG [Pereira 81], the trace is searched for using the stack operation. The same mechanism has also been adopted in LangLAB. In the case of bottom-up parsing, the timing of the stack operation causes inefficiency. LangLAB has overcome the problem by effective use of top-down prediction and by a slight extension of the goal clause [Konno 86].

To process both *fused* and *separable* idioms, LangLAB translates each related group of idioms into a special kind of a tree called TRIE [Kamiwaki 85]. Common words in the dictionary share the structure, saving memory space in the dictionary and increasing parsing speed. The original DCG rules (2) for idiom description are translated into the TRIE structure dictionary (3).

```
(2) v(1) --> [get,up], !.
    v(2) --> [get,on].
    v(3) --> [get].
    np(4) --> [not,only], np(NP1,_), [but,also], np(NP2,_), {check(NP1,NP2)}.
    adj(5) -- >[not,only], adj(AJ1,_), [but,also], adj(AJ2,_).
```

```
(3) dicta(get,[[v,[3]]],[
        [on,[[v,[2]]]],
        [up,[],
            [!,[[v,[1]]]]]]).
    dicta(not,[],[
        [only,[],
            [[adj,AJ1,_],[],
                [but,[],
                    [also,[],
                        [[adj,AJ2,_],[[adj,[5]]]]]]]],
            [[np,NP1,_],[],
                [but,[],
                    [also,[],
                        [[np,NP2,_],[],
                            [(check(NP1,NP2)),[[np,[4]]]]]]]]]]]).
```

In addition to the above extensions, several optimisations, including the following three, have been applied to the LangLAB system [Tanaka 86].

1) Indexed search for predicate 'link'
2) Direct consultation of dictionary, not via predicate 'goal'
3) Indexed search for asserted goals ('wf_goals' and 'fail_goals')

2.4 SAX

As described in Section 2.1, SAX is a sequential, efficient implementation of AX parsing method. This section briefly describes the organisation of SAX. Suppose there is a context free grammar shown in (1). On the right side of the grammar rules, the symbol 'id' followed by a number is not a grammatical symbol, but stands for an *identifier* that indicates a particular position in a particular grammar rule. The key mechanism of SAX is passing of identifiers.

```
(1) sentence --> np, id1 noun.
    np --> det, id2 noun.
    np --> np, id3 coconj, id4 np.
    noun --> noun, id5 rel_clause.
    noun --> noun, id6 pp.
    rel_clause --> [that], id7 vp.
```

```
pp --> prep, id8 np.
vp --> verb.
vp --> verb, id9 np.
vp --> vp, id10 coconj, id11 vp.
```

The parsing process operates from left to right and from bottom to top. Suppose a noun phrase has just been found. There are two kinds of processes that must be performed according to the grammar rules. The first is to start parsing using new grammar rules. The other is to augment already constructed incomplete tree structures to form more complete ones. In the system, the discovery of a noun phrase corresponds to a call of the definition of 'np'. Since the parsing process proceeds from left to right and bottom to top, a call of 'np' produces identifiers 'id1' and 'id3', which indicate that the parsing process has successfully proceeded up to these points in the grammar rules. It is defined as Prolog clause (2).

```
(2)  np1(X,[id1(X),id3(X)|Yt], Yt).
```

The second and third arguments of the clause represent the set of these two identifiers by difference lists. The first argument of this clause is a list of identifiers that are produced by the words or grammatical symbols immediately preceding the noun phrase in the given input sentence. The clause (2) produces the identifiers without regard to the contents of the first argument. It is, however, modified when top-down prediction is used. (This process is not described here.) This clause corresponds to the first job for a noun phrase. The second job for the noun phrase is to build up more complete tree structures by modifying partially constructed trees. This job is defined by the Prolog clauses shown in (3).

```
(3)  np2([],X,X).
     np2([id4(X)|Xt],Y,Yt) :-
        np(X,Y,Y1), !, np2(Xt,Y1,Yt).
     np2([id8(X)|Xt],Y,Yt) :-
        pp(X,Y,Y1), !, np2(Xt,Y1,Yt).
     np2([id9(X)|Xt],Y,Yt) :-
        vp(X,Y,Y1), !, np2(Xt,Y1,Yt).
     np2([_|Xt],Y,Yt) :- np2(Xt,Y,Yt).
```

The second clause of (3) states that a noun phrase can be constructed if it receives 'id4', which is only produced by a coordinating conjunction that has already received a noun phrase. The third and fourth clauses correspond to other occurrences of 'np' in the grammar rules. The first clause defines that it produces an empty difference list when it receives an empty list. The last clause is necessary to discard the identifiers that are irrelevant to a noun phrase.

Definition (2) is for the occurrences of 'np' as the leftmost element on the right side of grammar rules. Definition (3) is for the other occurrences of 'np' on the right side of grammar rules. They are called *type one* occurrences and *type two* occurrences. The complete definition of a noun phrase is a union of these definitions, as shown in (4).

```
(4)  np(X,Y,Yt) :-
        np1(X,Y,Y1), np2(X,Y1,Yt).
```

Although only analysis of 'np' was described here, the rest of the parsing also proceeds by receiving identifiers and producing new processes and new identifiers in the same way.

There is always a single clause for the definition of type one occurrence, and *cut* symbols are embedded in each clause in (3), guaranteeing that the parsing process is deterministic and never backtracks. This means that SAX is very efficient for the following two reasons.

1) Tail-recursion optimisation can be applied in compiled mode.
2) No side-effect is used for the *well-formed substring table*, while the same effect is achieved by the tail-recursive definition of type two clauses.

3. QUALITATIVE COMPARISON

This chapter compares the following three aspects of qualitative facilities.

1) Grammar descriptive power
2) Idiom handling and morphological analysis
3) Environment for grammar development

3.1 Grammar Descriptive Power

Current SAX system imposes some restrictions on DCG [Matsumoto 87a], because of its parsing mechanism.

Since the parsing process of SAX runs deterministically, as explained in Section 2.4, an extra-condition that contains ambiguous interpretations is evaluated only once during the parsing process, no matter what the grammar writer's intention is. This means that only the first successful substitution of variables is computed. Furthermore, if the grammar itself is ambiguous, the ambiguities spawn as many processes. Variables in spawned processes must be copied or renamed to ensure that they work correctly. To cope with these problems, SAX applies a restriction to DCG by modifying its formalism as follows.

```
c_0 --> c_1, {extra_1}, ..., c_n, {extra_n} & {delayed_extra}.
```

The extra-conditions to the left of '&' are evaluated only once. 'Delayed_extra' to the right of '&' is evaluated after the termination of the parsing process and may have ambiguities.

In parsing systems based on DCG with bottom-up parsing, such as BUP and LangLAB, the grammar developer need not be concerned with left-recursive rules.

As stated in Section 2.3, LangLAB also provides XGS formalism. XGS is effective for expressing structures such as relative clause and yes-no questions. In this experiment, a DCG with about 400 rules required only about 300 rules in XGS.

The current grammar formalism offered by SAX is a restricted DCG, as described above. However, SAX is also designed to provide parsing [Matsumoto 87b] for *gapping grammars* (GGs) [Dahl 84a] [Dahl 84b]. GGs are a very powerful grammar formalism that enables grammar writers to specify a rule concentrating on constituents in a sentence that are not necessarily adjacent.

3.2 Idiom Handling and Morphological Analysis

A parsing system must have flexible facilities to analyse various language phenomena. The separation of dictionary and grammar rules in BUP and LangLAB simplifies the introduction of morphological analysis and idiom handling by modifying the predicate 'dict' of the BUP program.

In BUP, an idiom dictionary distinct from the word dictionary is checked every

Table 3-1 Qualitative comparison

	BUP	LangLAB	SAX
Parsing method	Bottom-up and depth-first	Bottom-up and depth-first	Bottom-up and breadth-first
Side-effect	Used	Used	Not used
Copy of environment	Not required	Not required	Required
Parallel implementation	Impossible	Impossible	PAX
Grammar description	DCG	XGS (extension of DCG)	Restricted DCG
Extra-condition	Any Prolog program	Any Prolog program	Deterministic Prolog programs and *delayed evaluation* of nondeterministic Prolog programs
Idiom handling	Idiom dictionary	TRIE structure	(Being designed)
Morphological analysis	Longest matching	TRIE structure	Parallel analyser (under development)
Debugging tools	None	Interactive tracer	(Being designed)
Retranslation after correction of grammar	Whole	Corrected part only	Whole

time 'dict' is consulted [Matsumoto 84]. LangLAB handles idioms with a special structure called TRIE, and is reported to be superior to BUP for the following reasons [Kamiwaki 85].

1) Idioms can be described within the word dictionary.
2) Idioms with inflection can be handled.
3) Efficiency in both memory usage and analysis speed

In BUP, the processes of automatic segmentation and morphological analysis are combined. The segmentation algorithm isolates a word from the beginning of the input string based on the longest successful matching. When the word has an inflection, the type of inflection and suffix are examined [Matsumoto 84].

The morphological analysis system for SAX is under development [Sugimura 87]. The key mechanism is parallel search, like the AX parser.

3.3 Environment for Grammar Development

Grammar debugging tools are indispensable for large natural language parsing systems. This section compares the debugging environments of three systems.

LangLAB provides an interactive tracer for debugging grammars. This tracer is an interpreter that reads DCG grammar rules and executes them step by step according to the BUP algorithm. The LangLAB tracer displays various parameters such as partial parse trees, and changes states by commands from the user.

In LangLAB, grammars can be dynamically or partially corrected in the course of the development of large scale grammars.

SAX has not yet provided satisfactory debugging tools. This will cause various difficulties for the user since it is hard to follow the parallel parsing process, which is the key feature of SAX. Development of debugging tools is one of the most urgent problems for SAX.

4. QUANTITATIVE COMPARISON

4.1 Environment, Grammar and Dictionary for the Experiment

Systems

BUP ([Matsumoto 84] version)
LangLAB
SAX

Aspects

Parse time
a) Interpreted
b) Compiled
Memory space needed for parsing

Computer and environment

Quintus Prolog version 1.1 on VAX 11/785 VMS version 4.3
(about 10 KLIPS in compiled mode)

Grammar and dictionary

English grammar

BUP and SAX	398 DCG rules
LangLAB	295 XGS rules

(These cover almost the same range of English sentences.)

Dictionary	86 entries

Sample sentences

9 sentences (see APPENDIX)

Restriction

Morphological analysis is excluded because the morphological analysis system for SAX is under development at present.

4.2 Comparison of Parse Time

Tables 4-1 a) and b) show the results of the experiment. Some observations follow.

A. The efficiency of SAX and BUP is comparable in interpretive mode.
B. SAX is 7 to 60 times faster than BUP and 6 to 16 times faster than LangLAB in compiled mode.
C. LangLAB is 5 to 11 times faster than BUP and 6 to 10 times faster than SAX in interpretive mode.
D. SAX in compiled mode is about 150 to 230 times faster than in interpretive mode.

Result A indicates that SAX, BUP and LangLAB have the same order of computational complexity, because of top-down prediction and re-use of partial results. Results B, C and D describe the following key features, which contribute to the efficiency of SAX and LangLAB.

Table 4-1 Parse time*) of three systems (msec)
*) The time taken by garbage collection and stack shift is excluded.

a) Interpretive mode

Sentence number	Number of words	Number of parse trees		Interpreted		
		BUP SAX (DCG)	LangLAB (XGS)	BUP	LangLAB	SAX
1	4	1	1	21,330	3,740	35,800
2	5	1	1	16,730	2,050	16,730
3	7	2	2	45,740	7,850	52,200
4	10	1	1	57,320	10,830	60,310
5	11	2	2	70,590	13,310	114,770
6	18	1	2	103,830	20,510	140,550
7	21	5	5	343,590	56,170	414,610
8	19	1	1	128,600	26,520	146,770
9	20	6	2	243,420	22,170	230,070

b) Compiled mode

Sentence number	Number of words	Number of parse trees		Compiled		
		BUP SAX (DCG)	LangLAB (XGS)	BUP	LangLAB	SAX
1	4	1	1	1,290	1,240	190
2	5	1	1	690	800	110
3	7	2	2	3,130	2,660	320
4	10	1	1	4,970	4,130	330
5	11	2	2	11,860	5,630	530
6	18	1	2	18,210	8,920	630
7	21	5	5	112,500	27,070	1,750
8	19	1	1	23,490	11,610	770
9	20	6	2	61,490	11,090	1,010

1) The optimisation methods introduced in LangLAB are effective, particularly in interpretive mode.
2) The Prolog program that SAX generates is suitable for compiling techniques of Prolog, such as indexing of predicate and tail-recursion optimisation.
3) SAX does not use side-effects.

E. In LangLAB and BUP, the difference between the efficients when compiled and when interpreted is not as great as in SAX.

Result E shows that, in LangLAB and BUP, the interpretive search for asserted goals takes the greater part of parse time and that this is the primary reason for the slight improvement in parse time in the two systems when used in compiled mode.

F. The longer the sentence is and the more parse trees there are, the greater the difference is in parse time between BUP and SAX and between BUP and LangLAB when used in compiled mode.

This result means that SAX and LangLAB have the characteristic, G, desirable for large scale grammar in practical use.

G. Parse time increases approximately in proportion to the length of the sentence in SAX and LangLAB when compiled.

4.3 Comparison of Memory Space Needed for Parsing

Table 4-2 shows the size of the grammar used for evaluation and the programs translated by three systems. Table 4-3 shows the size of the *heap memory* that is used during the parsing process. Since SAX does not use side-effects, there is no data in Table 4-3.

Table 4-2 proves that the translated program of SAX is more than twice as large as those of BUP and LangLAB. However, the gross size of used memory space (i.e., the sum of the program size and the size of the heap memory used in parsing) of BUP becomes close to that of SAX for long sentences, depending on the experimental results.

Table 4-3 shows that the optimisations introduced in LangLAB are effective not only for parse time but also for memory space.

Table 4-2 Program size (bytes)

DCG (Grammar and dictionary)	BUP	LangLAB	SAX
52,896	103,000	98,648	227,084

5. DISCUSSION

This paper compared three natural language parsing systems based on logic programming (BUP, LangLAB and SAX) in both the qualitative and quantitative aspects. In qualitative comparisons, SAX gives the fastest parsing method for the DCG formalism (currently, restricted DCG) when used in compiled mode. SAX is 6 to 16 times faster than LangLAB, and LangLAB is 1 to 6 times faster than BUP. These results show that SAX has its upper bound in the BUP family that adopts the bottom-up algorithm. The speed of SAX is fast enough for practical uses. If a higher speed for parsing is required in the future, it will be realized in a parallel environment. Since SAX originates from the AX algorithm, the basis of which is parallel parsing, it will satisfy future requirements (in the form of PAX).

As discussed above, SAX (including AX) is one of the most promising parsing methods. However, it has not yet provided satisfactory environments such as debugging tools. In addition, it is important to investigate how the restriction given to

Table 4-3 Size of used heap memory (bytes)

Sentence number	Number of words	Number of parse trees		BUP	LangLAB
		BUP (DCG)	LangLAB (XGS)		
1	4	1	1	2,572	1,884
2	5	1	1	2,528	1,692
3	7	2	2	6,300	4,940
4	10	1	1	9,260	7,372
5	11	2	2	14,380	8,528
6	18	1	2	23,744	11,588
7	21	5	5	53,616	29,944
8	19	1	1	25,040	13,244
9	20	6	2	30,132	10,300

DCG formalism is relaxed while keeping it amenable to the AX algorithm. At present, LangLAB has the most advanced facilities and useful environment for developing natural language grammars. It also offers grammatical formalism that expresses left extrapositions.

A debugging model for SAX is expected to be very difficult. Since the grammar formalism of both LangLAB and SAX is DCG, it is proposed to use DCG as the interface of both systems and to use them jointly for the time being. Since it is the fastest among the three systems in interpretive mode, LangLAB will be used during the development of grammar in DCG form. After debugging the grammar, SAX will execute it efficiently, although SAX will need flexible facilities, such as tools for idiom handling and morphological analysis, and the full treatment of DCG formalism. Development of utilities is one of the most urgent areas of research for SAX.

ACKNOWLEDGMENTS

We would like to thank Dr. Yokoi (Deputy Director of ICOT) and Dr. Uchida (Chief of the Second Research Laboratory at ICOT) for their advice and encouragement. We also extend our thanks to the members of Prof. Tanaka's Laboratory at the Tokyo Institute of Technology and the Second Research Laboratory at ICOT for many useful discussions and comments.

REFERENCES

[Clark 84] K. L. Clark and S. Gregory, "PARLOG: Parallel Programming in Logic", Research Report DOC 84/4, Imperial College, April 1984

[Dahl 84a] V. Dahl and H. Abramson, "On Gapping Grammars", Proc. 2nd International Conference on Logic Programming, Uppsala, Sweden, pp.77-88, 1984

[Dahl 84b] V. Dahl, "More on Gapping Grammars", Proc. of the International Conference on Fifth Generation Computer Systems, Tokyo, pp.669-677, 1984

[Earley 70] J. Earley, "An Efficient Context-Free Parsing Algorithm", C.ACM, 13, 1970

[Kamiwaki 85] T. Kamiwaki and H. Tanaka, "TRIE Dictionary and Idiom Processing", Proc. of the Logic Programming Conference, Tokyo, 1985 (in Japanese)

[Kay 80] M. Kay, "Algorithm Schemata and Data Structures in Syntactic Processing", Technical Report CSL-80-12, Xerox PARC, 1980

[Konno 86] S. Konno and H. Tanaka, "Processing Left-extraposition in Bottom-up Parsing System", Computer Software Vol.3, No.2, pp.19-29, Tokyo, 1986 (in Japanese)

[Matsumoto 83] Y. Matsumoto, et al., "BUP: A Bottom-Up Parser Embedded in Prolog", New Generation Computing, Vol.1, No.2, pp.145-158, 1983

[Matsumoto 84] Y. Matsumoto, M. Kiyono and H. Tanaka, "Facilities of the BUP Parsing System", Proc. of 1st International Workshop on Natural Language Understanding and Logic Programming", Rennes, France, 1984

[Matsumoto 86] Y. Matsumoto, "A Parallel Parsing System for Natural Language Analysis", Proc. of 3rd International Conference on Logic Programming, London, 1986

[Matsumoto 87a] Y. Matsumoto and R. Sugimura, "A Parsing System Based on Logic Programming", to appear in Proc. 10th IJCAI, Milan, Italy, Aug. 1987

[Matsumoto 87b] Y. Matsumoto, "Parsing Gapping Grammars in Parallel", to be published as an ICOT Technical Report, Tokyo, 1987

[Pereira 80] F. Pereira and D. Warren, "Definite Clause Grammars for Language Analysis -- A Survey of the Formalism and a Comparison with Augmented Transition Networks", Journal of Artificial Intelligence, 13, pp. 231-278, 1980

[Pereira 81] F. Pereira, "Extraposition Grammars", American Journal of Computational Linguistics, 7, 4, pp. 243-256, 1981

[Pratt 75] V. R. Pratt, "LINGOL -- A Progress Report", Proc. of 4th IJCAI, pp. 422-428, 1975

[Sugimura 86] R. Sugimura and Y. Matsumoto, "Implementation of SAX on CIL", Proceedings of IPSJ, 1986, pp. 1799-1800 (in Japanese)

[Sugimura 87] R. Sugimura and Y. Matsumoto, "Parallel Lexical Analysis of Japanese Sentences", to be published as an ICOT Technical Report, Tokyo, 1987

[Tanaka 86] H. Tanaka et al., "Software System LangLAB for Natural Language Processing", Proc. of the Logic Programming Conference, Tokyo, 1986 (in Japanese)

[Ueda 85] K. Ueda, "Guarded Horn Clauses", ICOT Technical Report TR-103, Tokyo, 1985

APPENDIX Sample sentences (number of words)

1. I open the window. (4 words)
2. Diagram is an augmented grammar. (5 words)
3. The structural relations are holding among constituents. (7 words)
4. It is not tied to a particular domain of applications. (10 words)
5. Diagram analyzes all of the basic kinds of phrases and sentences. (11 words)
6. This paper presents an explanatory overview of a large and complex grammar that is used in a sentence. (18 words)
7. The annotations provide important information for other parts of the system that interpret the expression in the context of a dialogue. (21 words)
8. For every expression it analyzes, diagram provides an annotated description of the structural relations holding among its constituents. (19 words)
9. Procedures can also assign scores to an analysis, rating some applications of a rule as probable or as unlikely. (20 words)