Natural Language Understanding and Logic Programming
V. Dahl and P. Saint-Dizier (Editors)

Facilities of the BUP Parsing System

Yuji MATSUMOTO,
Electrotechnical Laboratory
Ibaraki, 305, Japan

Masaki KIYONO,
Matsushita Electric Industrial Co., Ltd.
Tokyo, 105, Japan

and

Hozumi TANAKA
Tokyo Institute of Technology
Tokyo, 152, Japan

ABSTRACT

The BUP Parsing System [1] is a bottom-up parsing system written in Prolog. It is intended for natural language analysis. Grammar rules and dictionary entries are written in an "epsilon-free" DCG formalism (a DCG formalism [2] that does not contain empty production rules). The BUP translator transforms the grammar rules and the dictionary entries into Prolog clauses. The resulting clauses together with some additional Prolog clauses combine to form a bottom-up parsing program. This can be viewed as a compilation of context-free grammar rules into a Prolog program.

This paper introduces the basic concept of the BUP system and some of the utility programs that are currently available for supporting the development of parsing programs. In addition to the BUP translator, the system includes utilities for morphological analysis, an idiom handler, and a tool for automatic segmentation of Japanese sentences. These facilities are neatly embedded in the framework of our system.

1. Introduction

The aim of this paper is to introduce the current facilities of the BUP parsing system [1]. The BUP system, is a general parsing system for natural languages, and is currently used for analysis of English and Japanese text.

DCGs (Definite Clause Grammars) of Pereira and Warren [2] give a clear grammatical formalism for natural languages. A DCG is directly transformed into a Prolog program which comprises a top-down backtracking parser for context-free grammars. Although the formalism is very clear, the derived parsing system has problems due to its reliance on a top-down backtracking algorithm. The parser cannot deal with left recursive rules, for they may cause an infinite loop; it cannot be an efficient parser because of its naive parsing strategy; and it is not easy to know when to consult the dictionary, since the dictionary is put on the same level as the grammar rules.

The basic component of the BUP system is a set of Prolog clauses called BUP clauses, which are obtained through a uniform transformation from grammar rules written in DCGs. We can say, therefore, that the BUP system gives another procedural semantics to DCGs.

A parsing system must have many flexible facilities to analyze a broad area of a language. Two such facilities are the automatic segmentation and idiom handlers. For example, text in some languages like Japanese do not include word separators, such as spaces. Parsing systems for such languages must take word inflection into account while partitioning sentences into their components. Another special problem is unusual word patterns (idioms) which often occur in sentences in many languages.

This paper introduces the basic concept of the BUP system and presents two utility programs that cope with the automatic segmentation and idiom handling problems mentioned in the previous paragraph.

2. A Brief Introduction to the BUP Parsing System

In the BUP system, users first describe grammar rules and dictionary entries using DCGs. Empty production rules are prohibited in our system, but most such rules can be included in the grammar rules with some modifications [1]. Next context-free grammar rules written in DCG formalism are then transformed into Prolog clauses (called BUP clauses) by the BUP translator [3]. For example, the following context-free grammar rules:

```
1)  sentence(s(NP,VP)) -->
          noun_phrase(NP),verb_phrase(VP).
2)  noun_phrase(np(john)) --> [john].
```

are transformed by the BUP translator to become:

```
1') noun_phrase(Goal,[NP],Info) --> {link(sentence,Goal)},
        goal(verb_phrase,[VP]),
        sentence(Goal,[s(NP,VP)],Info).

2') dict(john,noun_phrase,[np(john)]) --> [john].
```

We will not describe the system in detail here. Dictionary entries in the system are DCG clauses where the first element of the body is a list (i.e. a terminal symbol). All other clauses are treated as grammar rules. The informal meaning of BUP clause 1') is stated as follows: When a noun_phrase is found, first we check whether the non-terminal symbol 'sentence' can link to the current goal 'Goal' as a descendant. This process checks whether it is worth using this grammar rule in this context or not. If this check is successful the next thing to do is to find a verb_phrase. And we would get a sentence, if all of these are successful. The predicate 'link' is computed and all of the instances are asserted during the transformation. This predicate works as the top-down expectation just like the 'oracle' of LINGOL [4] and Extended LINGOL [5]. BUP clauses like 2') are dictionary entries. In this case, it says that 'john' is

a noun_phrase.

We must define the predicate 'goal'. Additional clauses are necessary to terminate the procedure:

```
3) goal(Goal,Args,[Word|X],Z) :-
        dict(Word,Cat,Args1,[Word|X],Y),link(Cat,Goal),
        P=..[Cat,Goal,Args1,Args,Y,Z],call(P).

4) cat(cat,I,I,X,X).      (for every non-terminal symbol 'cat')
```

The predicate 'goal' first consults the dictionary and obtains a non-terminal symbol ('Cat'). Next 'goal' checks whether the obtained non-terminal symbol can link up to the current goal, and makes a call whose predicate name is that non-terminal symbol (Thus we have now identified a phrase in the text which belongs to that non-terminal). Clause 4) says that the process terminates immediately if the current goal is equal to what has just been found.

All of the above Prolog clauses combine to form a parsing program. This is a basic view of the BUP parsing system.

A slight modification of 'goal' improves the efficiency of the parsing algorithm a great deal. The idea is to avoid useless repetitions by saving partial successes and failures. The predicate 'goal' is rewritten as follows:

```
goal(Goal,Arg,X,Y) :-
   wf_goal(Goal,_,X,_),!,wf_goal(Goal,Arg,X,Y);
   fail_goal(Goal,X),!,fail.
goal(Goal,Arg,X,Z) :-
   dictionary(Cat,Arg1,X,Y),link(Cat,Goal),
   P=..[Cat,Goal,Arg1,Arg,Y,Z],call(P),
   assertz(wf_goal(Goal,Arg,X,Z)).
goal(Goal,Arg,X,Z) :-
 ( wf_goal(Goal,_,X,_);
   assertz(fail_goal(Goal,X)) ),!,fail.

dictionary(Cat,Arg,X,Y) :-
   wf_dict(_,_,X,_),!,wf_dict(Cat,Arg,X,Y).
dictionary(Cat,Arg,[Word|X],Y) :-
 ( dict(Word,Cat,Arg,[Word|X],Y) ;
   morpheme(Cat,Arg,[Word|X],Y) ),
   assertz(wf_dict(Cat,Arg,[Word|X],Y)),
   fail.
dictionary(Cat,Arg,X,Y) :-
   wf_dict(Cat,Arg,X,Y).
```

In the above clauses, 'wf_goal', 'fail_goal', and 'wf_dict' are predicates used for asserting partial information, which correspond to the partial successes, partial failures, and the successful dictionary entries. The predicate 'morpheme' performs the morphological analysis and returns the original form of the word and the information about the inflection.

3. Facilities of the System

This section describes two major facilities of the BUP system, the automatic segmentation and the idiom handler.

3.1 Combining Automatic Segmentation with Morphological Analysis

Preliminary analysis of Japanese sentences poses an especially difficult problem: the division of sentences into words. Although it seems better to segment input sentences beforehand, it is difficult even for a Japanese to segment a sentence correctly. We have combined the processes of automatic segmentation and morphological analysis. The segmentation algorithm isolates a word from the beginning of the input based on the longest successful matching. When the word has an inflection, the type of inflection and the suffix are examined using the following 'morpheme' predicate:

```
morpheme(Cat,N_Arg,[Bunso|Bun],Bun1) :-
    name(Bunso,B_List),
    reverse(B_List,B_List_R),
    wakachi(B_List_R,[],R_List,Word,Cat,Arg),
    gobi_shori(Word,Cat,Arg,R_List,Bun,N_Arg,Bun1).

wakachi([Char|W_List_R],Rest,[Char|Rest],Word,Cat,Arg) :-
    reverse(W_List_R,W_List),
    name(Word,W_List),
    dict(Word,Cat,Arg,[Word],[]).
wakachi([Char|W_List_R],Rest,[Char|Rest],Word,doshi,Arg) :-
    onbin_shori(W_List_R,Arg).
wakachi([Char|W_List_R],Rest,R_List,Word,Cat,Arg) :-
    wakachi(W_List_R,[Char|Rest],R_List,Word,Cat,Arg).
```

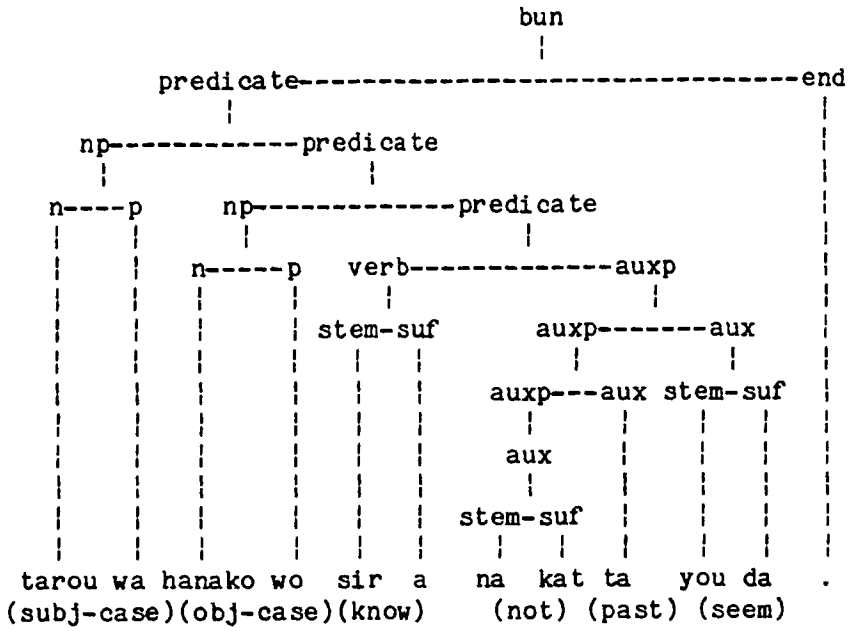
The predicate 'wakachi' takes one character from the given word and consults the dictionary. If the consultation fails, 'wakachi' then examines the possibility of euphonic changes (using the predicate, 'onbin_shori'), causing unusual formation of verb inflections. This process is executed repeatedly. Dictionary entries of inflectional words are indexed by their stems. Thus the inflectional analysis ('gobi_shori') is performed once a stem of a word has been isolated by 'wakachi'. Fig. 1 shows a sample analysis of a Japanese sentence. As is seen by the program, a partially segmented sentence is analyzed more efficiently than a sentence not segmented. A partially segmented input sentence, "tarou wa hanako wo siranakatta youda", takes a total of 938 msec. to execute.

3.2 Idiom Handling

Natural languages have many idiomatic expressions (idioms). By an idiom we mean a particular sequence of words and/or phrases, not necessarily consecutive, with a peculiar semantic interpretation. Although such a sequence may fit some of the grammar rules, we call it an idiom because of its peculiar interpretation. We choose not to represent idiomatic expressions using grammar rules, as the number of grammar rules directly affects the efficiency of the parsing system. Thus we construct an idiom dictionary, whose every element is invoked by a particular word. So, idioms are registered and referred to only by their head words. A typical idiom is expressed as follows (we

```
Input a sentence.
|: tarouwahanakowosiranakattayouda.
  (Tarou seemed not to know Hanako.)


937  msec.
No. 1
```




```
Total Time = 1576  msec.

number of wfgoal was :      28.
number of failgoal was :    11.
number of wfdict was :      22.
```

Fig. 1 A Sample Analysis Tree of a Japanese Sentence

(Words in parentheses are not the system output)

refer to this clause as an 'idiom clause'):

```
idiom(word,category,receive_var,return_var) --> body.
```

The first argument, 'word', is the key word which invokes this idiom. The morphological analysis ensures that even when idiomatic words in a given sentence are inflected, the proper idiom clauses are identified. The second argument, 'category', stands for the name of the grammar category that the idiomatic expression belongs to. The third argument, 'receive_var', represents the information of the key. The fourth variable, 'return_var', gives the complete information for the idiom when the call to the idiom clause has succeeded. All the idiom clauses comprise the idiom dictionary.

We now show some kinds of idiomatic expressions and how they are represented in the idiom dictionary.

(1) idioms made up of words only:

The body of such idiom clauses consists only of terminal symbols. However, we provide a predicate 'word' to handle words that may be inflected. Examples are:

```
a) idiom(as,postmod,_,[Syn,Sem,[postmod,'as well']]) -->
       [well].
b) idiom(computer,n,_,[Syn,Sem,[n,[n,computer],Tree]]) -->
       word(system,_,n,[Syn1,Sem1,Tree]).
```

The first example a) includes the non-inflected words, 'as' and 'well'. In the second example b), 'system' can appear in the plural form. The first argument of the predicate 'word' is the base form of the word and the third is the name of its grammar category. The second is for returning the information of suffix, which is not used here.

(2) idioms including phrases:

When an idiom includes not only simple words but some phrases, it can use the 'goal' predicate and any Prolog predicate to indicate the properties of the phrases.

```
a) idiom(by,adv,_,[Syn,Sem,[adv,[prep,by],Tree]]) -->
       goal(pron,[Syn1,Sem1,Tree]),
       { type_of(Syn1,reflexive) }.
b) idiom(look,v,Arg1,Arg) -->
     ( goal(np,Arg2) ; [] ),[up],
     { look_up(Arg1,Arg2,Arg) }.
```

The first example treats a phrase consisting of 'by' followed by a reflexive pronoun. The second example shows an idiom 'look up', that can have a noun phrase between its components.

(3) idioms including phrases not belonging to specific categories

Let us consider the following example:

```
idiom(not,Goal,_,[Syn,Sem,[Goal,[not,not],[only,only],Tree1,
            [but,but],[also,also],Tree2]]) -->
      [only],
      goal(Goal,[Syn1,Sem1,Tree1]),[but,also],
      goal(Goal,[Syn2,Sem2,Tree2]).
```

Two phrases (indicated by 'goal') that appear in this example are not specified by a certain grammar category. Instead, this idiom clause indicates that they belong to the same grammar category (The variable 'Goal' ensures this property) as does the whole sequence.

If we choose to restrict these phrases to one of the grammar categories, noun phrase, adjective phrase or prepositional phrase, this clause may be rewritten as below:

```
idiom(not,Goal,_,[Syn,Sem,[Goal,[not,not],[only,only],Tree1,
            [but,but],[also,also],Tree2]]) -->
      [only],{ member(Goal,[np,adjp,pp]) },
      goal(Goal,[Syn1,Sem1,Tree1]),[but,also],
      goal(Goal,[Syn2,Sem2,Tree2]).
```

Partial success results can be saved during the idiomatic analysis to avoid repetitive computation as in the previous cases of dictionary look-up and morphological analysis. Failure to find idioms is noted to avoid useless repetition. These optimizing memorizations are taken into our system just like in the case of 'goal'. The predicate 'dictionary' is modified as follows:

```
dictionary(Cat,Arg,X,Y) :-
   wf_idiom(_,_,X,_),!,
      wf_idiom(Cat,Arg,X,Y);
   fail_idiom(X),!,
      word(_,_,Cat,Arg,X,Y).
dictionary(Cat,Arg,[Word|Y],Z) :-
   idiom(Word,Cat,_,Arg,Y,Z),
      assertz(wf_idiom(Cat,Arg,[Word|Y],Z)),
      fail;
   word(Rword,_,Cat1,Arg1,[Word|Y],_),
      Rword\==Word,
      idiom(Rword,Cat,Arg1,Arg,Y,Z),
      assertz(wf_idiom(Cat,Arg,[Word|Y],Z)),
      fail;
   wf_idiom(_,_,[Word|Y],_),!,
      wf_idiom(Cat,Arg,[Word|Y],Z);
   assertz(fail_idiom([Word|Y])),
      fail.
dictionary(Cat,Arg,[Word|Y],Y) :-
   wf_dict(Word,Cat,Arg,_,_).
```

This 'dictionary' predicate differs from the previous one (in Section 2) in that it refers to the idiom handling. Thus this new 'dictionary' uses a 'word' predicate that actually acts the same as the previous 'dictionary' predicate. (Note: 'word' has already appeared in the definition of idioms). 'Word' has two arguments in addition to the arguments of the previous 'dictionary'. These additional arguments, the first and the second, unifies with the original form and with the suffix of the first word of the current

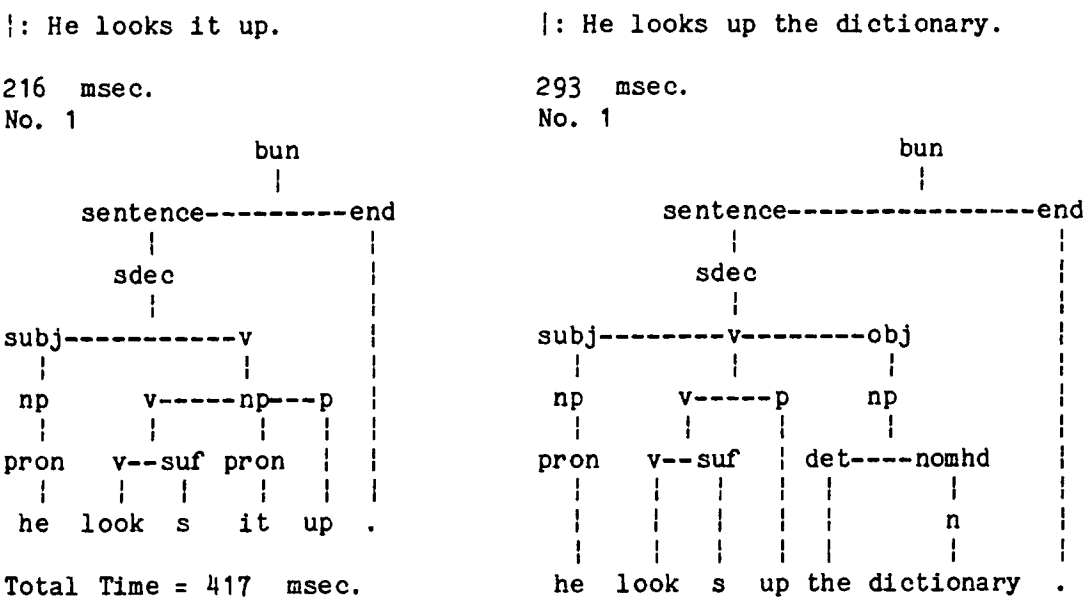


Fig. 2 Sample Parsing Trees

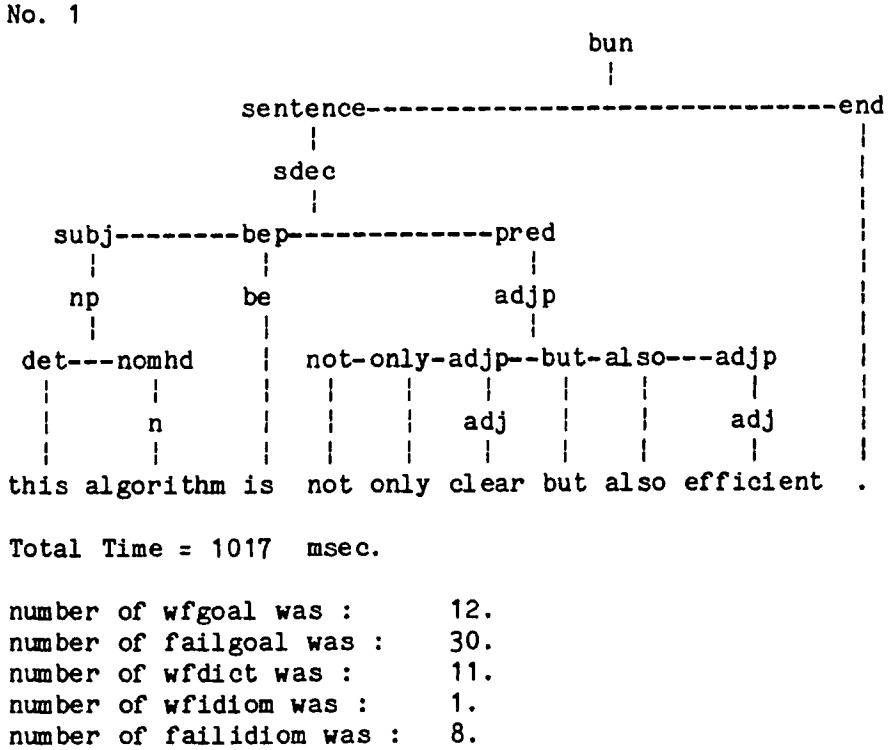


Fig. 3 A Sample Parsing Tree

input (the first element of the list given as the value of the fifth argument).

Note that the last clause for 'dictionary' only calls 'wf_dict'. This is because 'word' in the second clause of 'dictionary' has already searched for every variation of the first word of the current input by backtracking, and all of them have already been asserted. Some sample analyses of idioms are shown in Figs. 2 and 3.

4. Discussions and Conclusions

We have presented some of the facilities of the BUP parsing system. This bottom-up parsing system is more powerful than top-down realization of DCGs in that it is as efficient as Earley's [6] and Pratt's [4] algorithms, and that it does not fall into infinite loops even when left-recursive rules are included. The facilities described here have been achieved by slight modifications to the auxiliary predicates, 'goal' and 'dictionary'. The majority of the programs and the basic parsing strategy can be used with widely differing grammatical structures. Thus the system is a highly versatile facility for parsing many different natural languages, most notably Japanese and English.

Weak points in the system are that empty production rules are prohibited, and that cycles of grammar rules (sets of grammar rules which make a self recursive loop) may cause an infinite loop. The problem of cycles, which we have not yet discussed, is actually a special case of left recursive rules. Cycles can be rewritten without greatly changing the size of grammar rules and thus without greatly affecting the efficiency.

Future research on syntactic problems will consider undefined words, ellipses, parenthetical expressions, and movements.

In languages such as English and other Indo-European languages, undefined words can be easily identified after morphological analysis, even though no dictionary entry occurs. However, in a language like Japanese, where sentences usually do not delineate words with spaces or other separators, the identification of undefined words is very troublesome. Even if a sentence includes an undefined word, a part of the word may be recognized as a different word because of a wrong segmentation. Although the identification of undefined words might be performed by an exhaustive search, we need more sophisticated methods for the identification so as not to lose the efficiency.

Pereira's analysis of extraposition grammars [7] is inspiring. Most of his idea on left extrapositions, a major problem in the phenomenon of movements, can be incorporated into the original BUP system. We hope to adjust the memorization feature of the revised version so as to implement left extraposition with this optimization.

Bottom-up strategies are better than top-down strategies for the problems of ellipses and parenthetical expressions (For a discussion of these problems, see [8]). We plan to attack such phenomena using the BUP system.

The entire system is implemented both in DEC-10 Prolog [9] ,[10] on DEC 2060 and in C-Prolog [11] on VAX-11/780. The execution time shown in the examples is the CPU time of the compiled version of DEC-10 Prolog on the DEC 2060. The interpreted versions on the DEC 2060 and VAX-11/780 require respectively about five times and seven times as much execution time as the compiled version on the DEC 2060.

Acknowledgments

The authors wish to express their thanks to Mr. Kazuhiro Fuchi, Director of the ICOT Research Center for his encouragement. We thank the members of Dr. Tanaka's Lab at the Tokyo Institute of Technology, the natural language processing group at ICOT Research Center, and the Machine Inference Section of ETL for their discussions and comments. Thanks are also due to Monica Strauss for her various comments on this manuscript.

References

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

[2] Pereira, F.C.N. and Warren, D.H.D., Definite Clause Grammar for Language Analysis--A Survey of the Formalism and a Comparison with Augmented Transition Networks, Artificial Intelligence, 13, pp.231-278, 1980.

[3] Matsumoto, Y., Kiyono, M. and Tanaka, H., BUP Translator (in Japanese), Bulletin of Electrotechnical Laboratory, vol.47, no.8, pp.67-85, 1983.

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

[5] Tanaka, H., Sato, T. and Motoyoshi, H., Extended LINGOL --A Programming System for Natural Language Processing (in Japanese), IECE Japan, J60-D, Dec. 1977.

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

[7] Pereira, F.C.N., Extraposition Grammars, AJCL, vol.7, no.4, pp.243-256, October-December, 1981.

[8] Hayes, P.J., Flexible Parsing, AJCL, vol.7, no.4, pp.232-242, October-December, 1981.

[9] Pereira, L.M., Pereira, F.C.N. and Warren, D.H.D., User's Guide to DECsystem-10 Prolog, Edinburgh, 1978.

[10] Bowen, D.L. (eds.), DECsystem-10 Prolog User's Manual, DAI Occasional Paper no.27, Edinburgh, 1982.

[11] Pereira, F., C-Prolog User's Manual version 1,2a, EdCAAD, Edinburgh, 1983.