A META TRANSLATOR OF A LOGIC GRAMMAR THAT HANDLES MISSING CONSTITUENTS

N. N. KARANIKOLAS, Dept. of Informatics,
Athens University of Economics,
76 Patission St., Athens 104 34, Greece

ABSTRACT

The work reported in the present paper concerns a metatranslator that translates into Prolog code a grammar, written in a notation similar to Definite Clause Grammars, augmented with two operations called HOLD and HELD. In order to implement these operations the Prolog code produced by the metatranslator maintains a push-down Hold List. The operations are tested in the resolution of bound and unbound missing constituents of natural language sentences.

1. INTRODUCTION

Metaprograms have been used to analyze, transform and simulate other programs. In particular a *metainterpreter* for a language is an interpreter for that language written in the language itself [Sterling and Shapiro]. A specific example, which concerns the formal description of an *attribute grammar intepreter*, that can be interpreted by an almost identical interpreter, is described in [Kontos and Papakonstantinou].

A *metatranslator* of a grammar is a program that translates the grammar to a program in the same language that the metatranslator is written. This enables the writer of the translator to augment the grammar with features of the target language. Such a metatranslator can be used to alter the execution flow of the target program, as in the *BUP* and *LangLAB* systems, for example [Okunishi et al.], from top-down to bottom-up. Metatranslators can be useful in translating grammars that construct *logic structures* as in McCord's *Modular Logic Grammar* [McCord]. The work reported in this paper concerns a metatranslator which augments a logic grammar with two operations, called **HOLD** and **HELD**. In particular, the metatranslator presented translates a Grammar written in a *Definite Clause Grammar (DCG)* like notation to Prolog code in order to implement these operations. The Prolog code produced maintains a push-down Hold List. The operations **HOLD** and **HELD** are tested for the resolution of *bound* and *unbound missing constituent* (a case of bound and unbound *movement*, respectively [c.f. Allen]) into natural language sentences. It will be shown, in particular, how the **HOLD** operation, in conjunction with the **HELD** operation solve bound missing constituent and how the

S. G. Tzafestas (ed.), Engineering Systems with Intelligence, 51–58.

HOLD operation alone resolves the unbound missing constituent.

Section 2 gives a brief introduction to the missing constituent problem in natural language sentences. The notation and the operations used by the logic grammar proposed is the subject of section 3. The methodology of the metaparser, that takes as input such a logic grammar and produces Prolog code that parses the language sentences, is the subject of section 4.

2. MISSING CONSTITUENTS

We will present two cases of missing constituent. The first case is presented in sentences that take an *infinitive* subsentence as a *verb complement*. As an example consider the following sentence (1):

Mary wants to meet John. (1)

In sentences such as (1) the first noun phrase of the infinitive subsentence is systematically omitted. The reason for this omission is that the omitted part is repeated somewhere in the main sentence. The place where this constituent is repeated is either the noun phrase which lies before the verb of the main sentence, or the noun phrase which lies after the verb of the main sentence but before the infinitive subsentence. We would thus like a *syntactic interpreter* to fill the omitted constituent, before constructing the syntactic tree and analyze the sentence (1) as if it was sentence (2):

Mary wants [Mary to meet John]. (2)

Other examples of such sentences are:

Mary advised Bill to read the book. (3)

Mary wants to be met by John. (4)

We expect the syntactic interpreter to analyze such sentences as if they were:

Mary advised Bill [Bill to read the book]. (5)

Mary wants [Mary to be met by John]. (6)

As it is known beforehand where the omitted part can be found, this kind of ellipsis is called *bound ellipsis* or *bound missing constituent.*

The second case of ellipsis is presented in cases where a *relative clause* is used to further specify a noun phrase. Consider the following examples:

A policeman caught the man who stole the old woman. (7)

The man who hit Mary with a book has disappeared. (8)

The man who(m) Mary hit with a book has disappeared. (9)

We would like a syntactic analyzer to fill the omitted constituent and handle the above sentences as if they were:

A policeman caught the man.
[The man stole the old woman]. (10)

The man has disappeared.
[The man hit Mary with a book]. (11)

The man has disappeared.
[Mary hit the man with a book]. (12)

Sentences (10,11,12) show that the noun phrase which is further specified by the relative clause sentence, can be used to fill an omitted constituent in that sentence. Unfortunately, it is not known beforehand *which* constituent in the relative clause sentence can be filled by the noun phrase. This kind of ellipsis is called *unbound ellipsis* or *unbound missing constituent.*

3. THE LOGIC GRAMMAR

Definite Clause Grammars, DCGs for short [Pereira and Warren], can be used for both the description of a language and the analysis of sentences of this language. Because DCGs can be translated automatically into Prolog, they are characterized as *executable grammars*. Another advantage of DCGs is their ability to describe the *analysis structures* that are used to produce the *derivation tree* of a sentence. The above and other advantages of DCGs convinced us to use the DCG notation to describe our *Logic Grammar* which handles bound and unbound missing constituent. The following two rewrite rules are used to understand active voice sentences that take an infinitive (toinf) as a verb complement.

```
sact(sact(XX,Y,Z))-->
        np(X):vact(Y):HOLD(np,X)<toinf(Z):>:XX=subj(X):.          (13)
sact(sact(XX,Y,ZZ,W))-->
        np(X):vact(Y):np(Z):HOLD(np,Z)<toinf(W):>:
        XX=subj(X):ZZ=obj(Z):.                                    (14)
```

The operation **HOLD** is implemented by pushing its two arguments - an object and its type - down the push-down Hold List that is automatically maintained by the target Logic Grammar (Prolog code). The angle brackets that follow the operation **HOLD** mean that the rule (or *one* of the rules if there are more than one rule) inside them will use the top element of the Hold List, instead of finding an element of the same type from the list of the sentence's words. In other words, the Hold List after the application of the rule (or the rules) inside the angle brackets will have the same contents as those it contained before the application of the **HOLD** operation. Rules such (13) and (14) also use secondary variables (e.g. XX in (13) and (14) and ZZ in (14)) that build intermediate substructures. These substructures are used later to evaluate the arguments of the coresponding rule head, in order to build the syntactic structure returned by the rule in question. The following four rewrite rules are used for understanding subsentences in infinitive form. The first three deal with infinitive subsentences in active voice, whereas the last one is required for infinitive subsentences in passive voice.

```
toinf(toinf(W))-->
        HELD(np,X):to(NULL):vactinf(Y):pps(Z):
        XX=subj(X):W=sact(XX,Y,Z):.                               (15)
toinf(toinf(R))-->
        HELD(np,X):to(NULL):vactinf(Y):np(Z):pps(W):
        XX=subj(X):ZZ=obj(Z):R=sact(XX,Y,ZZ,W):.                  (16)
toinf(toinf(R))-->
        HELD(np,X):to(NULL):vactinf(Y):np(Z):np(W):
        pps(V):XX=subj(X):ZZ=iobj(Z):WW=obj(W):
        R=sact(XX,Y,ZZ,W,V):.                                     (17)
toinf(toinf(R))-->
        HELD(np,X):to(NULL):vpasinf(Y):pps(W):
        XX=obj(X):R=spas(Y,XX,W):.                                (18)
```

The **HELD** operation is implemented as follows. It is first tested whether the element at the top of the Hold List is of the same type as the operation's first argument. If it is, then the next (second) element of the Hold List is returned to its second argument and the top two elements of the list are removed from it. Rule (13) in conjuction with rule

(16) parse sentence (1), rule (13) with (18) parse sentence (4), while rules (14) with (16) parse sentence (3). Thus, the **HOLD** operation in conjunction with the **HELD** operation can be used for the resolution of bound missing constituent.

The following two rewrite rules can be used for parsing the sentences. These will be used later in the Appendix in tracing the resolution of unbound missing constituent:

```
s(SVAR)-->sact(SVAR):.                                          (19)
s(SVAR)-->spas(SVAR):.                                          (20)
```

The following four rules are used to parse noun phrases:

```
np(np(N))-->name(N):.                                           (21)
np(np(D,A,N))-->det(D):adj(A):noun(N):.                         (22)
np(np(P))-->pronoun(P):.                                        (23)
np(np(D,A,N,R))-->det(D):adj(A):noun(N):relpro(DUMMY):
        X=np(D,A,N):HOLD(np,X)<s(SVAR):>:R=rels(SVAR):.         (24)
```

Rule (24) is in particular used to parse noun phrases that take as a specifier of the noun a relative clause sentence. Another rule that parses active sentences and which will be used in following, is:

```
sact(sact(XX,Y,ZZ,W))-->
        np(X):vact(Y):np(Z):pps(W):XX=subj(X):ZZ=obj(Z):.       (25)
```

Finally, the system supplies automatically rule (26):

```
np(X)-->HELD(np,X):.                                            (26)
```

Rule (24) in conjunction with a rule such as (26) can be used for the resolution of unbound missing constituent in a relative clause (which further specifies another noun). A trace of the cooperation of these rules in the syntactic analysis of the following phrase of sentence (7) is presented the Appendix.

the man who stole the old woman. (27)

Rules such as (26) are automatically inserted by the Logic Grammar metatranslator (see next section). This means that the user defines only rules such as (24) and expects the constituent pushed to be used at a deeper stage, but before control is returned to the rule that follows the right angle bracket. Thus, from the user point of view, the **HOLD** operation alone is responsible for the resolution of unbound missing constituent.

4. METATRANSLATOR METHODOLOGY

DCGs use a *difference list* for the words in a sentence. A metatranslator that translates a DCG rule into Prolog has to insert two additional arguments that maintain the difference list of words. A rule such as (19) could be translated by a DCG metatranslator as:

```
s(SVAR,DLIn,DLOut) :- sact(SVAR,DLIn,DLOut).                    (28)
```

where DLIn is the list of words before the application of the rule and DLOut is the list of the words remaining after this application. In the previous section we mentioned that our Logic Grammar additionaly handles automatically a Hold List. Thus, a metatranslator that translates our grammar into Prolog could produce something like the following Prolog clause (29):

```
s(SVAR,HLIn,HLOut,DLIn,DLOut) :-
        sact(SVAR,HLIn,HLOut,DLIn,DLOut).                       (29)
```

where HLIn and HLOut are the contents of the Hold List before and after the application of the rule. Because our metatranslator does not know beforehand how many rules occur

in the right hand side of a rule, rule (19) is translated as rule (30), while rule (22) is translated as rule (31).

```
s(SVAR,HLA,HLOut,DLA,DLOut) :-
        sact(SVAR,HLA,HLB,DLA,DLB),HLOut=HLB,DLOut=DLB.          (30)
np(np(D,A,N),HLA,HLOut,DLA,DLOut) :-
        det(D,HLA,HLB,DLA,DLB),adj(A,HLB,HLC,DLB,DLC),
        noun(N,HLC,HLD,DLC,DLD),HLOut=HLD,DLOut=DLD.             (31)
```

Our metatranslator carries out two more things when a **HOLD** operation is encountered in the body of a Logic Grammar's rule. First, it inserts two more arguments in the **hold** Prolog rule produced for the **HOLD** operation. These refer to the Hold List before and after the pushing. Second, it inserts another rule, in the body of the translated Logic Grammar's rule, which tests if the Hold List before the pushing is equal (**isEQ**) to the Hold List after the application of the rules inside the angle brackets. For example, rule (13) is translated into (32):

```
sact(sact(XX,Y,Z),HLA,HLOut,DLA,DLOut) :-
        np(X,HLA,HLB,DLA,DLB),vact(Y,HLB,HLC,DLB,DLC),
        hold(np,X,HLC,HLD),toinf(Z,HLD,HLE,DLC,DLD),
        isEQ(HLC,HLE),XX=subj(X),HLOut=HLE,DLOut=DLD.            (32)
```

where isEQ is the rule that tests the equality of the two Hold Lists as explained above. As this example shows the construction of substructures into secondary variables does not need to be translated as these constructions do not change when they are translated into Prolog.

The **HELD** operation is translated into a **held** Prolog rule that has two additional arguments. These refer to the Hold List before and after the popping. For example rule (15) is translated into rule (33):

```
toinf(toinf(W),HLA,HLOut,DLA,DLOut) :-
        held(np,X,HLA,HLB),to(NULL,HLB,HLC,DLA,DLB),
        vactinf(Y,HLC,HLD,DLB,DLC),pps(Z,HLD,HLE,DLC,DLD),
        XX=subj(X),W=sact(XX,Y,Z).                               (33)
```

The metatranslator reads the rules from a source file, it translates them using the methodology presented in this section and it appends the Prolog clauses produced in a target file. When the translation is completed, the metatranslator creates and appends to the target file Prolog clauses similar to rule (26). Some examples of such rules are:

```
s(X,HLIn,HLOut,DL,DL) :- held(s,X,HLin,HLOut).                   (34)
np(X,HLIn,HLOut,DL,DL) :- held(np,X,HLin,HLOut).                 (35)
```

The reason for defining such rules has been explained in the previous section. The following five rules are also appended to the target file:

```
hold(Type,OBJ,HoldIn,[Type,OBJ|HoldIn]).                         (36)
held(Type,OBJ,[Type,OBJ|HoldOut],HoldOut).                       (37)
isEQ([X|Xs],[X|Ys]) :- isEQ(Xs,Ys).                              (38)
isEQ([],[]).                                                     (39)
parse(ListOfWords) :-
        s(S,[],[],ListOfWords,[]),write(S).                      (40)
```

Rule (40) takes a list of words and checks if those words compose a complete sentence. It initialises the Hold List to empty, it expects to be empty at the end of the parsing (c.f. the second and third argument of the s rule) and it also expects to use all the words in this list (c.f. the last argument of the s rule). After all these steps are carried out by the

metatranslator, the target file consists of a Prolog program that can parse sentences and produce syntactic trees.

In the notation used in our logic grammar terminal symbols are represented either as themselves or as variable that contains them. Rule (41), for example, shows that if the word "good" is the next word in the sentence processed, then it succeeds and returns adj(good).

```
adj(adj(good))-->good:.                                          (41)
```

This rule is translated into the Prolog clause (42):

```
adj(adj(good),HLA,HLOut,DLA,DLOut) :-
        DLA=[good|DLB],HLOut=HLA,DLOut=DLB.                      (42)
```

The logic grammar can call immediately Prolog rules. For example, if the Prolog clause n(N,P) succeeds, and, furthermore, if the symbol that variable N unifies, by the prolog rule, is the next word of the sentence, then rule (43) succeeds and returns noun(N,sn). This rule is translated to the Prolog clause (44):

```
noun(noun(N,sn))-->PROLOG(n(N,P)):N:.                            (43)
noun(noun(N,sn),HLA,HLOut,DLA,DLOut) :-
        n(N,P),DLA=[N|DLB],HLOut=HLA,DLOut=DLB.                  (44)
```

5. CASES WITHOUT MISSING CONSTITUENTS

So far we have only presented examples of infinitival complements of main sentences which have their first noun phrase ommited (e.g. examples (1), (3), (4)). However, there are cases where there is no missing constituent in the infinitival complement. For example consider the following sentence (45):

Mary wants Bill to wash the dishes. (45)

In this case, we would like our syntactic interpreter to analyze such a sentence as (46).

Mary wants [Bill to wash the dishes]. (46)

To handle such cases we introduce another one rule (47) that analyzes sentences and four more rules that mirror rules (15), (16), (17) and (18) in the cases of infinitival complements without omission. Rule (48) mirrors rule (16), while the other three rules are obvious and have been omitted for saving space.

```
sact(sact(XX,Y,Z))-->
        np(X):vact(Y):toinfnomiss(Z):XX=subj(X):.                (47)
toinfnomiss(toinf(R))-->
        np(X):to(NULL):vactinf(Y):np(Z):pps(W):
        XX=subj(X):ZZ=obj(Z):R=sact(XX,Y,ZZ,W):.                 (48)
```

Rule (14) in conjuction with rule (16) produces the simplified syntactic tree of the form (49), while rule (47) in conjuction with (48) produces the simplified syntactic tree of the form (50) for both sentences (3) and (45). In (49) and (50) *V1* stands for the verb *advise* in (3) and *want* in (45), *V2* stands for the verb *read* in (3) and *wash* in (45), and *N1* stands for the noun *book* in (3) and *dish* in (45).

```
sact(   subj(mary),
        vt(V1),
        obj(bill),
        toinf(sact(     subj(bill),
                        vt(V2),
                        obj(N1) )))                              (49)
```

```
sact(   subj(mary),
        vt(V1),
        toinf(sact(     subj(bill),
                        vt(V2),
                        obj(N1) )))                    (50)
```

The semantic interpretation module takes both trees (49) and (50) for each sentence. It decides tree (49) for sentence (3) and (50) for sentence (45), depending on the *deep cases* and their *selectional restrictions* [Bruce] that the main verb in the sentence imposes. However, further consideration of semantic interpretation is beyond the scope of this paper.

CONCLUSIONS

The Logic Grammar suggested was designed to solve the cases of bound and unbound missing constituents. In the first case the user defines grammar rules that save a constituent - with the use of the **HOLD** operation - and grammar rules that determine where the saved constituent can be used - with the use of the **HELD** operation. In the second case the user defines grammar rules that save a constituent - with the use of the **HOLD** operation -and expects the saved constituent to be used wherever a constituent of the same type is omitted.

ACKNOWLEDGMENTS

My thanks are due to Professors John Cavouras and John Kontos of the Athens University of Economics and Business and Dr. Edgar Whitley of the London School of Economics, for reading this paper and for making many useful suggestions.

REFERENCES

Allen, J. Natural Language Understanding. The Benjamin, Cummings Publishing Company, 1987.

Bruce, B. Case systems for natural language. Artificial Intelligence 6 (1975), 327-360.

Kontos, J., and **Papakonstantinou**, G.K. The Interpretation of MetaGrammars Describing Syntax-Directed Interpreters Using an Attribute Grammar Interpreter. IEEE Trans. on Software Engineering. SE-8, 4(1982), 435-436.

McCord, M. Natural Language Processing in Prolog. In Walker, A. Ed., Knowledge Systems and Prolog. Addison-Wesley, Reading, MA., 1987.

Okunishi, T., Sugimura, R., Matsumoto, Y., Tamura, N., Kamiwaki, T., and Tanaka, H. Comparison of Logic Programming Based Natural Language Parsing Systems. In Dahl, V., and Saint-Dizier, P., Eds., Natural Language Understanding and Logic Programming, II. North-Holland, 1988, pp. 1-14.

Pereira, F., and **Warren**, D. Definite Clause Grammars for Natural Language Analysis - A Survey of the Formalism and a Comparison with Augmented Transition Networks. Artificial Intelligence 13 (1980), 231-278.

Sterling, L., and **Shapiro**, E. The Art of Prolog, MIT Press Series in Logic Programming, 1986.

APPENDIX: A trace of " ... *the man who stole the old woman.*"

```
Step    Rule and Explanation
1       Rule (24) applies
2               Rule (det) applies and returns det(the)
3               Rule (adj) applies and returns adj(null)
4               Rule (noun) applies and returns noun(man,sn)
5               Rule (relpro) applies
6               X becomes np(det(the),adj(null),noun(man,sn))
7               Rule (HOLD) pushes np and
                np(det(the),adj(null),noun(man,sn))
                to the Hold List
8               Rule (19) applies
9                       Rule (25) applies
10                              Rule (26) applies
11                                      Rule (HELD) applies, pops the objects np
                                        and np(det(the),adj(null),noun(man,sn))
                                        from the Hold List and returns the second
                                        object
12                              Rule (26) returns
                                np(det(the),adj(null),noun(man,sn))
13                              Rule (vact) applies and returns
                                vt(steel,actpast,null)
14                              Rule (22) applies and returns
                                np(det(the),adj(old),noun(woman,sn))
15                              Rule (pps) applies and returns null
16                              XX becomes
                                subj(np(det(the),adj(null),noun(man,sn)))
17                              ZZ becomes
                                obj(np(det(the),adj(old),noun(woman,sn)))
18                      Rule (25) returns
                        sact(   subj(np(det(the),adj(null),noun(man,sn)))
                                vt(steel,actpast,null)
                                obj(np(det(the),adj(old),noun(woman,sn))))
19              Rule (19) returns
                sact(   subj(np(det(the),adj(null),noun(man,sn)))
                        vt(steel,actpast,null)
                        obj(np(det(the),adj(old),noun(woman,sn))))
20              The Hold List is tested and is found equal to what it
                was in step 6 because the objects pushed in step 7 are
                popped later in step 11
21              R becomes
                rels(   sact(   subj(np(det(the),adj(null),noun(man,sn)))
                                vt(steel,actpast,null)
                                obj(np(det(the),adj(old),noun(woman,sn)))))
22      Rule (24) returns
        np(     det(the)
                adj(null)
                noun(man,sn)
                rels(   sact(   subj(np(det(the),adj(null),noun(man,sn)))
                                vt(steel,actpast,null)
                                obj(np(det(the),adj(old),noun(woman,sn))))))
```