Kragujevac J. Math. 24 (2002) 179–192.

A PROLOG PROGRAM FOR ANALYSIS AND SYNTHESIS OF TYPICAL SENTENCES TAKEN FROM ADVERTISEMENTS

Mirjana M. Lazić

Department of Mathematics, Faculty of Science, University of Kragujevac, P. O. Box 60, 34000 Kragujevac, Yugoslavia.

(Received March 8, 2002)

Abstract. This paper is about a PROLOG program for analysis and synthesis of typical sentences taken from advertisements. That program checks grammatical accuracy of a sentence, i.e. it is used as a syntactic analyzer of a chosen fragment of natural language. On the other hand, the same program is used as a generator of the fragment of language.

1. INTRODUCTION

Having in mind that some demands of grammar can be simply expressed by means of arguments of the structures which appear in them, it is clear that PROLOG is a very convenient program for the development of syntactic analyzers, as well as syntactic generators of natural language (see [2], [3], [4]). The realization of the program must be preceded by the formalization of grammar [1], which is a very complex and

opened problem. The chosen advertisements are written in a form of complete and grammatically correct sentences. After their detailed grammatical analysis, according to the groups they belong to, the formal grammar is made for a chosen fragment of language (see [5]).

2. DESCRIPTION OF PROGRAM

The main characteristics of the program, and the definitions of the most important predicates are described. In order to be more comprehensible, the program can be divided into four parts. The "translation" of both the grammatical rules and the dictionary that defines the fragment of natural language in the PROLOG program is given in the first two parts .

The third part refers to semantics, i.e. to the semantic accuracy of the sentences of the chosen fragment, and in the last part of the program some auxiliary predicates are defined.

2.1 RULES OF GRAMMAR

The initial part of the program represents "the translation" of the grammatical rules, i.e. it consists of the definitions of the predicates *sentence, predp, vp, advp, np, cnp, modp*, which are used for the formation of some complex categories (sentence, predicate phrase, verb phrase, adverbial phrase, noun phrase, common noun phrase and modifier phrase).

Predicate *sentence* is assigned in the following way:

sentence(X,Y,sentence(I,G)): - np(nom,3,Num,X,Z,I),predp(nom,3,Num,Z,Y,G).
sentence(X,Y,sentence(I,G)): - np(nom,Gender,_,X,Z,I),predp(nom,3,pl,Z,Y,G).
sentence(X,Y,sentence(G)): - predp(nom,1,Num,X,Y,G).
sentence(X,Y,sentence(I,G)): - np(dat,_,_,X,Z,I),predp(dat,Z,Y,G).
sentence(X,Y,sentence(G)): - predp(dat,X,Y,G).

Regarding the rules for the description of this predicate, the line of words representing the difference between the list X and the list Y forms a sentence if the beginning of that line is a noun phrase (*np*) and the rest of it a predicate phrase (*predp*), while the noun phrase can be omitted, because it is possible to omit subject in sentences. The demands for the congruence of noun and predicate phrases in case, person and number are realized by means of equality or congruence of appropriate arguments of the predicates.

In these sentences the verb is transitive, so the verb phrase (*vp*) is not sufficient for the formation of the predicate phrase (*predp*), but its obligatory part is a direct object and very often there is also an indirect object. This means that a verb phrase (*vp*), and one or two noun phrases (*np*) in appropriate cases and in certain places, take part in the formation of the predicate phrase. This predicate has four or six arguments. The first one indicates the case of a noun phrase which stands (or should stand) next to the predicate phrase, which is being discussed, the last three are usual and the remaining two, which appear in case of need, represent the mark for person (the first or the third) or number (singular or plural) of the subject which does the action of the verb.

Here are the rules for the description of the predicate *vp* (verb phrase), which also has four or six arguments. When there are six of them, the first two are used for marking person and number of the subject, which stands next to this phrase. The important argument, which exists in all predicates, *vp* is a list of cases of the nouns, which they connect. A verb has to take part in the formation of verb phrase, and an adverbial phrase (*advp*) can stand in front of or behind it. When using passive voice, for example "is needed", there is also a past participle "needed" (*adjp*) beside the verb form "is".

The predicate *advp* (adverbial phrase) is described by two rules. According to the first rule, an adverbial phrase consists of one adverb (*adv*) only, and according to the second one it consists of conjunction of two different adverbs.

For the description of the predicate *np* several rules can be used. If there is only a common noun phrase, it is defined in the following way:

np(K,A,B,X,Y,np(I)): - cnp(K,A,B,X,Y,I), elem(K,[nom,dat,acc,za_acc]).

If there is a common noun preceded by a mark of number, than the following rule is used:

np(K,Gender,_,X,Y,np(D,I)): - det(K,Gender,X,Z,D), cn(gen,Gender,Z,Y,I),
elem(K,[nom,za_acc]), elem(Gender,[m,f]).
np(dat,Gender,_,X,Y,np(D,I)): - det(dat,Gender,X,Z,D), cn(dat,Gender,Z,Y,I),
elem(Gender,[m,f]).

And finally, if there is a common noun or a new noun phrase preceded by a preposition, the following rules are introduced:

np(K,X,Y,np(P,I)): - prep(K,X,Z,P), cn(K,_,_,Z,Y,I), not (K = za_acc).
np(za_acc,X,Y,np(P,I)): - prep(acc,X,Z,P),np(za_acc,A,B,Z,Y,I).

Regarding the predicate, there are four obligatory arguments. The first one is the one that marks the case and the last three have in common all the predicates of the first two parts of program. The remaining two are sometimes, if it is necessary, used for marking both person and number, and sometimes the first one marks gender and the second one is anonymous.

There are also the rules, in the program, used to describe the predicate *cnp* (common noun phrase), which can either be a common noun (*cn*) alone, or a common noun with one or two modifier phrases (*modp*) either in front of or behind the noun. It is possible, of course, to add more modifier phrases, but it would considerably expand the part of the program concerning semantics. The arguments of this predicate are the same as those of the preceding one. The first one is the mark of the case, while the second and the third one are used if necessary, and the last three are the same as those of the other predicates of this part of the program.

Modp (modifier phrase) is a predicate described at the end of this part of program:

modp(K,A,B,X,Y,modp(M)): - adj(K,A,B,X,Y,M).
modp(K,_,_,X,Y,modp(M)): - np(L,X,Y,M), elem(K,[nom,acc]),not L=za_acc.

That is either an adjective (*adj*) or the description of the location of object realized by means of noun phrase (*np*), where a common noun is preceded by a preposition (*prep*). In this case, it is important that every preposition stands next to the nouns in the particular case. All those demands are realized by using the arguments. Here, it is possible to expand the program, because if something is "on the coast" it can be on the coast of a lake, river or sea or if something is "in the center", that is, for example, in the center of a city, etc. Then, it is possible to specify the name of the street, city, lake, etc., which is very often present in chosen sentences.

2.2 DICTIONARY

For the second part of the program it can be said that it is the "translation" of the dictionary into PROLOG program, because here are definitions of the predicates *verb, adv, cn, adj, det, prep, cnj, adj_p* by means of which groups of words, which belong to the appropriate lexical categories (verb, adverb, common noun, adjective number, preposition, conjunction and past participle) are assigned. The auxiliary predicate *elem* has been used everywhere.

The rules for the description of the predicate verb are given at the beginning, and depending on them, the verbs are assigned. The verbs in the present tense, the first or the third person singular or plural have been used. The arguments of this predicate are the same as those of the corresponding predicate phrase, since the verb is its integral part. The important characteristic is the argument, which represents the list of cases of the nouns, which the verb links, because some verbs can have an indirect object, and some cannot. Therefore, the verbs *kupiti (to buy)* and *prodati (to sell)* appear without an indirect object in the first person singular and plural:

```
verb(1,sng,[nom,acc],[LX],X,verb(L)): -
                    elem(L,[izdajem,iznajmljujem,kupujem,prodajem,tražim]).
verb(1,pl,[nom,acc],[LX],X,verb(L)):-
                    elem(L,[izdajemo,iznajmljujemo,kupujemo,prodajemo,tražimo]).
```

in contrast to verbs *izdavati (to rent out)* and *iznajmljivati (to let out)* which can be used with or without an indirect object (in singular and plural):

verb(1,sng,[nom,acc,dat],[LX],X,verb(L)): - elem(L,[izdajem,iznajmljujem]).
verb(1,pl,[nom,acc,dat],[LX],X,verb(L)): - elem(L,[izdajemo,iznajmljujemo]).
verb(1,sng,[nom,acc,p_acc],[LX],X,verb(L)): - elem(L,[izdajem,iznajmljujem,tražim]).
verb(1,pl,[nom,acc,p_acc],[LX],X,verb(L)): - elem(L,[izdajemo,iznajmljujemo,tražimo]).

Then, the rule for the description of the predicate *adv* is given, and the set of adverbs is assigned.

Common nouns are assigned by predicate *cn* and divided into groups according to the cases they appear in, and, where it is necessary, according to the gender, i.e., person and number. In the cases where they appear with a preposition, it is also stated before the case so that they could distinguish from those which appear alone in the same case (*gen* and *pored_gen*, *acc* and za_acc), i.e., with different prepositions (*u_loc, na_loc*). So, for example, we have:

cn(acc,_,_,[LX],X,cn(L)): - elem(L,[. . .]). i
cn(za_acc,_,_,[LX],X,cn(L)): - elem(L,[. . .]).
i.e.:
cn(u_loc,_,_,[LX],X,cn(L)): - elem(L,[. . .]). i
cn(na_loc,_,_,[LX],X,cn(L)): - elem(L,[. . .]).

Then, the rules for the description of the predicate *adj* (adjective) are given. In that sense, adjectives are assigned and divided into groups according to cases, i.e. either according to gender or to person and number, if necessary. Since they appear next to nouns, their arguments are the same.

Besides the singular, plural number is also introduced (two is the most probable to be met in the chosen sentences). Predicate *det* with the four rules defining it, gives particular forms of that number in three cases *(nom, dat, za_acc)* and two genders (*m* and *f*). Here, as well as in the previous part of the program, an expansion on the new cases (numbers) is possible.

In the chosen fragment of language there are several prepositions, which are assigned by predicate *prep*. The first argument of this predicate is the case of the nouns they stand by. In most cases the preposition is also cited for the reasons mentioned above.

The predicate *conj* introduces only one conjunction, which appears in these sentences, and which could be introduced into some other places.

And, finally, at the end of this part of the program, there are rules for the description of the predicate *adj_p*, by means of which the three genders of the past participle *needed* are assigned.

2.3 SEMANTICS

The following, third part of the program, as it has already been said, is made to control semantical accuracy of the sentences. It is not related to the problems of grammatical nature, but to the meaning of the words that appear in a chosen dictionary and to their relations. Namely, there are sentences that are grammatically correct, but they are clumsy, they are not it the nature of language, etc. In this part of the program only two predicates appear,*sen* and *sentence 2*. While for the description of the first one a great number of rules are used, the second one is assigned by only one rule.

For the list X applies *sen(X)*, if it consists of the elements that cannot stand together. Thus, first we set rule that next to a noun in a particular case only one of the adjectives from the given group in the same case can stand. In other words, a sentence presented as a list of words is not correct if the adjective related to a given noun does not belong to the given group. For example, the noun *garsonjera (a studio)* can appear with the following adjectives: *prazna (empty)*, *nameštena (furnished)* and *polunameštena (semi-furnished)*. In the program it has been solved in the following way:

sen(X): - je_adj(L,garsonjera,X),not elem(L,[prazna,nameštena,polunameštena].

For the nouns that can be used separately, i.e. without an adjective, that set is empty, so the appropriate rule for them is:

sen(X): - je_adj(L,K,X),not elem(L,[]),
elem(K,[stranac,stranci,učenik,učenica,učenici,učenice, nepušač,nepušači]).

Regarding the possibility that a noun can have two adjectives, not only do the adjectives have to be different, but there are those which can not stand together (for example:*prazna, nameštena soba (empty, furnished room)*, or *dvosoban, trosoban stan (two room, three room flat)*. Those cases are eliminated from the set of correct sentences by several rules for defining the predicate *sen*. One of them is:

sen(X): - elem(prazna,X),elem(K,X),elem(K,[nameštena,polunameštena]).

Beside the adjectives, nouns can appear with modifier phrases in order to be more comprehensible. They also have to have some limitations because, for example, *plac (a building site)* cannot have an attic or it cannot be in a house. In other words, *apartman (a suite)* already has electricity and water so it does not have to be emphasized. So, there are the rules for the defining the predicate *sen* which prevent those combinations from appearing. For example:

sen(X): - elem(plac,X),elem(L,X), elem(L,[grejanjem,telefonom,
kupatilom,kuhinjom, inventarom,placem,potkrovljem,kući]).

And, finally, we should notice that students, single men, etc. do not need a building site, stand or cultivated field but a room, flat, house, etc. A two-bed room is for two persons so it is not necessary to repeat that. The elimination of those semantically incorrect sentences is done by several rules for the defining of the predicate *Sen*.

The predicate *sentence2* is used for the formation, i.e. identification of the syntactically and semantically correct sentences of the fragment. Namely, a sentence written in the form of a list of words (X) is syntactically and semantically correct (*sentence2(X,[],_)*) if it does not consist of the elements (words) which cannot stand together (*not sen(X)*).

Simply, using this predicate, from the group of syntactically correct sentences only these which are also semantically correct are isolated.

2.4 AUXILIARY PREDICATES

Finally, in the last part of the program, the definitions of the auxiliary predicates have been given. The first group of them helps the realization of the grammar demands by using the PROLOG language (*elem, je_lista, razlglava, je_adj*). The second group is used for the transformation of a sentence into a list of words (*form_list, formiraj, rec, promeni*), and the third group is used for writing the parse tree (*ispis*).

The predicate *elem* checks if an element is in the list and, in the other hand, it generates the elements from the list one by one. The first one is used in the analysis, and the second one is used in the synthesis of sentences. It has been given recursively, so that something is an element of the list if it is a "head" of the list, or if it is an element of the list's "tail".

The predicate *je_lista* has also been given in a recursive way. A list is *je_lista* if it has different elements. So, *je_lista* is an empty list, and *je_lista* is a list where a head is not an element of a "tail" *je_lista*. Due to the specificity of the problem, a list where the same prepositions appear several times next to different nouns is also *je_lista*.

In some cases, it was necessary for the lists to have different "heads", and that was simply solved by means of a predicate *razlglava*.

The congruence of an adjective L and a noun K, next to which the adjective stands in the list (in the sentence), in case, gender and number, enables the predicate*je_adj(L,K,X)*.

Different predicates in this part of the program are used in the analysis of sentences. Namely, the program, which acts as a generator of the natural language's fragment, generates the sentences as lists of words separated by commas. Regarding the program which deals with a syntactical analysis, there is a problem concerning an appropriate structure for the presentation of the natural language's sentence, as

well as the possibility of its input into system, because the sentences of the natural language are not the terms of the PROLOG language.

The first way of the transformation is to write a sentence as a list of its words, where the words with initial capitals should be quoted with single quotation marks, so that they could become constants. Otherwise, those words would be variables and their meaning would be lost from the natural language consequently. The demand for manual transformation of a sentence into a list of words, before uploading the data, is a basic disadvantage of this method.

The second way of the sentence transformation into a term of PROLOG language, would be to quote it with a double quotation marks. Thus, it is transformed into the codes list of the signs, that make it, so that it is, then, the structure of PROLOG. This solution is simpler than the previous one, but this way cannot enable the access to a single word of a sentence, and that is exactly what a syntactical analysis is based on.

The procedure *form_list* enter firstly the natural language's sentence without double quotation marks. Since that sentence is not the term of the PROLOG language, its entering by the predicate *read* is not possible. So, the entering is realized by the predicate *get0*, by means of which the sentence is uploaded sign by sign (exactly, code by code) until the sign for the end has been found (i.e. sign "." whose code is 46). The use of the predicate *get0* makes this part of the program specific.

The procedures *formiraj* and *rec* are always called with an entering sign, which should be processed. At first, the processing of a sign with a chosen constituent is tried, and if it does not work, the processing of the same sign with the next constituent is tried. So the failure of a chosen constituent does not mean the loss of that sign. However, if the next sign was read in a body of a chosen constituent the failure of the processing and selection of the next constituent would simultaneously mean the loss of an entered sign. Actually, returning by the target *get0* means the loss of the entered sign.

During the formation of a word list for a given sentence, it is convenient (because of using the dictionary) that the capitals from the entrance sentence should be replaced

by corresponding small letters. Since the ASCII codes of the capital letters range from 65 to 90 and the codes of small letters from 97 to 122 the replacement of the capital letters' codes by the codes of the corresponding small letters can be simply realized using the procedure *promeni*. The replacement of the codes is realized in the procedure *rec*.

The transformation of the codes list into the list of words is done by the procedure *formiraj*. Let's see the third constituent of that procedure. From the given list of codes, the list of codes for a single word is formed by the procedure *rec*. Than, the formed list of codes is transformed into a word by the systematic predicate *name*. Finally, from the rest of the given list of the codes, the list of the remaining words is formed recursively. The transformation is finished when the sign for the termination has been found (i.e. the sign "." with code 46). There is a cut in the first constituent of the procedure that proves that it is the only result of the transformation. The second constituent of the procedure *formiraj* executes the omission of the blanks (code 32) which appear within some words.

The formation of the codes list for a particular word is done in the procedure *rec*. The third argument of this procedure is used for returning the unused part of a given list of codes into a procedure *formiraj*, where the recursive formation of the remaining words is being continued. The list of codes for a single word is formed when the full stop (code 46) or the blank (code 32) has been found and this is emphasized by the cut in the first and the second constituent.

During the analysis of a sentence, yes/no answers i.e., the sentence is/is not syntactically correct, are not informative enough. In order to correct that, the third argument on which the generation of the parse tree is based, has been introduced into all non-terminal symbols of the grammar. Let's see the first constituent of the procedure *sentence*. The third argument of that constituent's head is structure *sentence (I, G)*. The first argument of that structure gets the value by the fulfillment of the target *predp*. The way those values are got can be seen from the arguments of the procedures they belong to, as well as from their past procedures, according to the dictionary. The omission of some optional parts of a sentence is marked by a

constant '**'. For example:

?-sentence([devojci,potrebna,soba],[],S).
S = sentence(np(cnp(cn(devojci))),predp(vp(verb(**),adj_p(potrebna)),
np(cnp(cn(soba))))) →;
S = sentence(np(cnp(cn(devojci))),predp(adj_p(potrebna),np(cnp(cn(soba))))) →;
No

The sentences similar to the previous one have two parse trees, because the grammar rules concerning the formation of the verb phrase (*pred*) say that the passive voice is formed either by a verb and past participle or only by past participle.

The parse tree derived in this way is very clumsy for longer sentences. Its more simple realization is defined by the procedure *ispis*.

The last predicate in this part of the program is *analiziraj*. It uses the procedure for entry of a sentence (*form_list*), its analysis (*sentence2*), and the generation of the parse tree and its writing (*ispis*). Let's illustrate that, using the target:

?-analiziraj.

Student hitno traži prazan stan u centru.
(A student needs an empty flat in the centre urgently)

sentence
np
cnp
cn: **student**
predp
vp
advp
adv: **hitno**
verb: **traži**
np
cnp
modp

adj: **prazan**

cn: **stan**

modp

prep: **u**

cn: **centru**

yes

The second argument of the target *sentence2* from the body of the procedure *analiziraj* is an empty list. Namely, if a line of words, which, in the list Lwords formed a given entry, really represents a sentence of a language then all the words from that list have to be used during the fulfillment of that target. So, a sentence is formed by a line of words, which represents a difference between the first list Lwords, and the empty list. The second argument of the target is an empty list even in the cases when PROLOG program works as a generator of syntactically correct sentences of a given language. Then, the target is set in this way:

?-sentence(A,[],_),write(A),fail.

if we want to get all the "fruits" of the grammar. If we want to get both syntactically and semantically correct sentences then we set the target:

?-sentence2(A,[],_),write(A),fail.

Setting the target:

?-sentence2(A,[],_).

we can also get all the sentences but one by one, using the sign ";" (and then using the sign for the end of a line [Return]). Thus, the system has been asked to find a new way of fulfillment of the same target and also to find an answer to a question which has been asked by that target. Then, the system "forests" the previous answer and tries to fulfill the target differently, so the second answer can be found.

By putting "S" instead of anonymous third argument in the targets mentioned above, we would get words separated by commas but we would also get a parse tree for that sentence (in the clumsy form).

This PROLOG program can be applied for:

- Control of the belonging of sentence to a chosen fragment
- Analysis of the chosen fragment's sentence
- Synthesis of a chosen fragment's sentences

References

[1] M. Prešić, *Zasnivanje prirodnih jezika – formalne gramatike,* unpublished monograph.

[2] M. Prešić, *A PROLOG Program for Generating a Fragment of Serbo-Croation,* The Prague Bulletin of Mathematical Linguistics 54, 1990.

[3] S. Prešić, *PROLOG, relacijcki jezik,* Matematički fakultet, Beograd, Nauka, Beograd, 1996.

[4] M. Radovan, *Programiranje u Prologu,* Informator, Zagreb, 1990.

[5] M. Lazić, *Formal grammar for a fragment of Serbian language*, Kragujevac J. Math. **24** (2002), 147–178.