A Chart Parsing implementation in Answer Set Programming

Ismael Sandoval Cervantes
Ingenieria en Sistemas Computacionales
ITESM, Campus Guadalajara
elcoraz@gmail.com

Rogelio Dávila Pérez
Departamento de Sistemas de Informacion
CUCEA, Universidad de Guadalajara
rdav90@gmail.com

Abstract. The present document presents an attempt to develop an Answer-Set Prolog program to implement a well-known strategy for natural language parsing, known as **chart parsing**. It is important to notice that the program presented is an end-term project for a course in natural language processing, even though we believe there are interesting insights that can be extracted from this effort.

1 Introduction

An important language for knowledge representation and programming, called Answer-Set Programming (ASP) [1] has been developed and offers interesting possibilities for natural language processing. With the aim of exploring the capabilities of ASP in this area, it was decided to build an experimental program for undertaking the parsing of a natural language sentence using a well-known technique, called **chart parsing**. The results are presented in this document. The program was built from some ideas taken form [4], and important insights extracted from the experience are included.

The rest of the paper is organized as follows: section 2 introduces the preliminaries of Chart Parsing; in section 3 the implementation of the program is discussed and in section 4 some conclusions are presented.

2 Chart Parsing

Natural languages are characterized by presenting a high degree of structural ambiguity, for example:
(a) Mary and John hear the report on the travel

The sentence (a) is ambigous since two different syntactic structures can be assigned to it, as it is shown in **Figure-1**.

In the representation *(b)* Mary and John hear the report while they were traveling. In the second one *(c)* Mary and John hear a report about a travel; two different meanings for the same sentence!

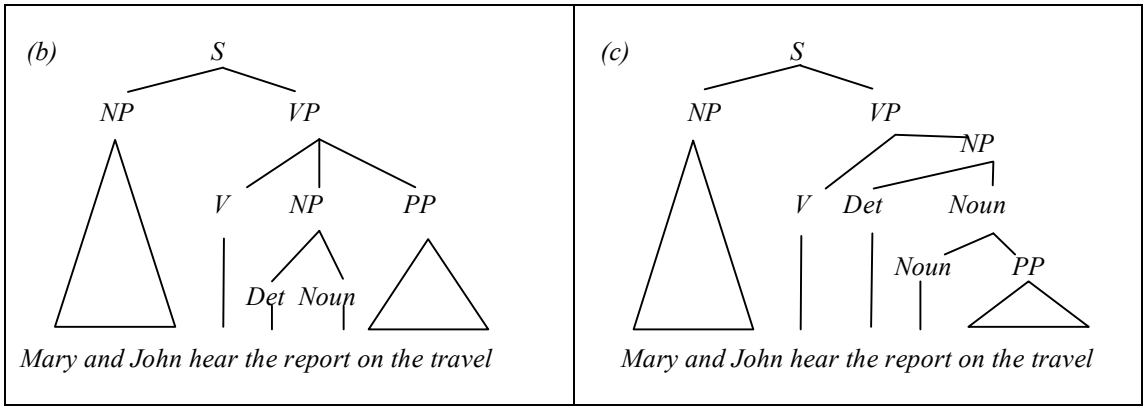


Figure 1: Parsing a sentence

Syntactic structures are generated by the following grammar defined by the *context free rules* that express the way in which different components of language are combined to build sentences:

(1) S → NP VP
(2) VP → V NP
(3) VP → VP NP PP
(4) NP → Det Noun
(5) NP → PropName
(6) NP → NP Conj NP
(7) Noun → Noun PP
(8) PP → Prep NP

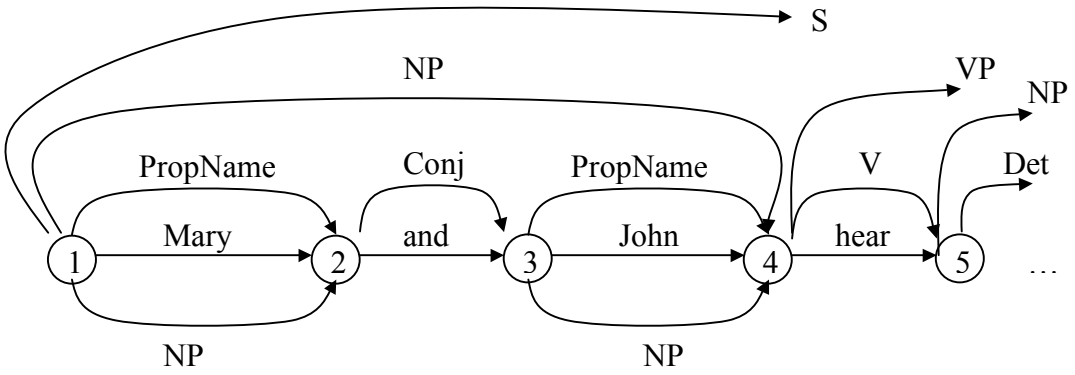
and the *lexicon rules* which attach to each word its corresponding category:

L1 Det → the
L2-3 Noun → report | travel
L4 Verb → hear
L4-5 PropName → Mary | John
L6 Conj → and
L7 Prep → on

The application of a parsing process to the sentence *(a)* using the grammar presented above conveys the syntactic structures shown in *(b)* and *(c)*.

The main desirable for a parsing algorithm is to avoid rebuilding substructures that have already been recognized during the parsing process. An important method for natural language parsing is called **Chart Parsing** [3,5]. It consists of a tabular based bottom-up parsing algorithm. The basic idea is to keep a table containing each substructure generated during the parsing process in different iterations until all possible structures for the sentence are obtained.
The input for the method consists of a sentence seen as a string of words and a grammar, espressed as a set of production rules.

An example of the (partial) structure generated after applying a chart parsing algorithm to our sample sentence *(a)* is shown below:



The point is that once we have recognized a substructure to say, a noun phrase (NP), we never come back an remake that analysis.

In the next section the actual implementation of the program is discussed.

3 Implementation

The implementation of the algorithm in AS Prolog is given in four steps:

Step 1. Create the initial table including an arc for each word in the sentence.

Step 2. Add a self_arc for each grammar rule whose first left-most element of the right-hand side is already in the table.

Step 3. Extend every self_arc into an arc of the same category where the full right-hand of the rule has been found.

Step 4. Iterate from Step 2 to 4 incrementing the time until the given limit.

Now we will show the way in which the following simple sentence is processed:

A dog bite the girl

There is a C interface which given the grammar and the sentence generates the following program.:

3.1 Step 1

Each word is mapped to a predicate called *word/3* whose arguments are the actual word, the initial and ending nodes.

```
word(a,0,1).
word(dog,1,2).
word(bite,2,3).
word(the,3,4).
word(girl,4,5).

init(0).
end(5).
time_goal(5).

range(0..5).
time(0..5).
```

ASP requires the initial and the last node to be expressed so we used the predicates ini*t/1* and *end/1*. Another parameter that needs to be considered is the limit for the iteration process. This limit is introduced by the *time_goal/1* predicate.

A rule has to be introduced for initiating the table *holds/2*, whose arguments are the *arc/3* and the *time/1*, starting to the value *0*.

```
holds(arc(X,A,B),0):-
       word(X,A,B), range(A), range(B).
```

Then the lexicon is introduced in the following way:

```
verb(bite).
dete(a).              % determiner
dete(the).
noun(girl).
noun(dog).
rule(v).              % verb
rule(np).             % noun phrase
rule(s).              % sentence
rule(vp).             % verb phrase
rule(noun).
rule(det).            % determiner
```

3.2 Step 2

Now the rules that have a chance to be applied are introduced. The decision on whether a rule will be applied or not depends on having detected that the left-most element of the right hand part of the rule has been recognized, as stated below:

```
holds(self_arc(sentence,A),T+1):- holds(arc(np,A,B),T),
                                  time(T),range(A),range(B).

holds(self_arc(np,A),T+1):- holds(arc(det,A,B),T),
                            time(T),range(A),range(B).

holds(self_arc(vp,A),T+1):- holds(arc(v,A,B),T),
                            time(T),range(A),range(B).

holds(self_arc(v,A),T+1):- holds(arc(X,A,B),T),word(X,A,B),v_t(X),
                           time(T),range(A),range(B).

holds(self_arc(det,A),T+1):- holds(arc(X,A,B),T),word(X,A,B),dete(X),
                             time(T),range(A),range(B).

holds(self_arc(noun,A),T+1):- holds(arc(X,A,B),T),word(X,A,B),noun(X),
                              time(T),range(A),range(B).
```

3.3 Step 3

The context free grammar rules are introduced for each of the syntactic categories. An arc labeled with this category will be introduced in the table whenever the constituents of the category are reached. These new arcs correspond to the expanded arcs in the model.

```
holds(arc(s,A,B),T+1):- holds(self_arc(s,A),T),
                         holds(arc(np,A,C),T),
                         holds(arc(vp,C,B),T),
                         range(A),range(B),range(C),time(T).

holds(arc(np,A,B),T+1):- holds(self_arc(np,A),T),
                         holds(arc(det,A,C),T),
                         holds(arc(noun,C,B),T),
                         range(A),range(B),range(C),time(T).
```

.

3.4 Step 4

The result of the process is given by collecting an arc with category ″S″ sentence) which goes from the initial to the final node. The presence of this arc means that a sentence has been recognized to belong to the language generated by the provided grammar. If more than one arc like this one are found, it means that different syntactic structures where found for the same sentence and it is structurally ambiguos.

If it happens that none of the arcs satisfied the previous conditions then the sentence was not recognized.

We define two rules, called the ***inertia rules*** [4], they basically state that ″actions normally do not affect fluents″ or ″things remain unchanged unless something change them. ″ Theses rules preserve any arc which exists in time T at the new time T+1.

```
holds(arc(X,A,B),T+1):-
      holds(arc(X,A,B),T),
      not -holds(arc(X,A,B),T+1),
      range(A),range(B),
      time(T),rule(X).
```

```
holds(arc(X,A,B),T+1):-
      holds(arc(X,A,B),T),
      not -holds(arc(X,A,B),T+1),
      range(A),range(B),
      time(T),word(X,A,B).
```

These rules prevent an arc for been inserted if it is already inside.

3.5 Sample Output (formatted)

```
//////////   tiempo 0

holds(arc(a,0,1),0)                %
holds(arc(dog,1,2),0)             % The inicial arcs are introduced
holds(arc(bite,2,3),0)          %
holds(arc(the,3,4),0)               %
holds(arc(girl,4,5),0)             %
```

```
//////////   tiempo 1

holds(arc(a,0,1),1)             % these arcs are introduced by the
holds(arc(dog,1,2),1)           % inertial rule
...

holds(self_arc(v,2),1)          % A self_arc is an arc that points
holds(self_arc(det,0),1)        % to the same node
...
```

And it goes on until the final table generated in time 6, shown below:

```
//////////   tiempo 6

holds(arc(a,0,1),6)             % block found in time 0
holds(arc(dog,1,2),6)           %
...

holds(self_arc(v,2),6)          % block found in time 1
holds(self_arc(det,0),6)        %
...

holds(arc(v,2,3),6)                % block found in time 2
holds(arc(det,0,1),6)
...

holds(self_arc(np,0),6)         % block found in time 3
holds(self_arc(np,3),6)
holds(self_arc(vp,2),6)

holds(arc(np,0,2),6)
holds(arc(np,3,5),6)              % block found in time 4

holds(self_arc(s,0),6)
holds(self_arc(s,3),6)            % block found in time 5
holds(arc(vp,2,5),6)

holds(arc(s,0,5),6)        %  The sentence arc is generated
                                %  from node 0 to node 5
```

As shown in the final table the solution for the sentence was finally found. The process should keep running until no more arcs are generated as another solution may came out.

4 Conclusions

A program build under the ASP model to accomplish a chart parsing of sentences of natural language has been developed. Even when the application was developed as the final project for a course in natural language processing,

Some positive conclusions from the experience:

a) It became to be a challenging and interesting project that provided us of a deeper understanding of the ASP model.
b) Once you have realized the way in which the knowledge has to be represented in ASP it is more intuitive to model the problem.
c) An interesting feature of the software is that it keeps track of all events that had happen during the developing of the solution which is very helpful to debugging.

Some reflections on the resulting program and the AS Programming model are:

a) At each step of the iteration every arc from the previous one, T, is copied into the new one, T+1. This fact implies that much information is redundant.
b) In this first attempt the grammar is encoded inside the program and not just a different input to it. The most diserable configuration would be for the grammar to be independent of the program.
c) It was not easy to find out the way in which the algorithm had to be developed in ASP, there are interesting material [1,2] but they were not written in order to introduce the newcomers into this programming area.

References

[1] M. Gelfond and V. lifschitz. The stable model semantics for logic programming. In Robert Kowalski and Kenneth Bowen, editors, *Proceedings of International Logic Programming Confernce and Symposium*, pp 1070-1080, 1988.

[2] M. Gelfond and V. lifschitz. Classial Negation in Logic Programs and Disjunctive Databases. *New Generation Computing*, vol. 9. pp 365-385, 1991.

[3] Gerald Gazdar and Chris Mellish, *Natural Language Processing in Prolog: An Introduction to Computational Linguistics*, Addison-Wesley Publishers, 1989.

[4] Mónica Nogueira, *Building Knowledge Systems in A-Prolog*. Ph. D. Thesis, Computer Science Department, University of Texas at El Paso, 2003.

[5] Christer Samuelsson and Mats Wirén, Parsing Thechnics, in *Handbook on Natural Language Processing*, Marcel Dekker Publishers, NY, 2000.