University of Malta
BSc(Hons.) IT Year III

CSA305: Natural Language Processing using Logic Programming

Lecture 11

Parsing Techniques

- Empty Categories

Department of Computer Science and AI
1999/2000

Empty Categories

Sometimes we need to introduce *empty categories*, i.e categories that are realised as the null string.

These are necessary, for example, if we assume that all NPs have determiners, and then discover that there are sentences like

birds fly.

These cases can be handled using the null string on the RHS of productions, which we write "0".

S	$\rightarrow$	NP VP
NP	$\rightarrow$	D N
D	$\rightarrow$	0
D	$\rightarrow$	...

Handling Empty Categories

Empty categories can cause certain problems for parsers, since in principle, they can appear anywhere in a parse tree without being associated with any visible lexical material.

This is no problem when parsing *top down*, since the empty category is only introduced when the grammar says so.

checkpoint: How would you write such a rule for a DCG and what would its translation be?

However, it is a problem when parsing bottom-up. The existence of a rule like

$\phi \rightarrow 0$

means that the subtree ϕ can be invoked *everywhere*. Of course, most of these subtrees will be redundant, just as the subsentence *the cow walked to market* is redundant in the sentence *the cow walked to market was sold*.

Empty Categories Left Corner Parsing

The left-corner parser is only partially able to deal with rules that introduce such null consituents.

If the null constituent is encountered when parsing *top down*, there is no problem, since the parser works like an ordinary top-down parser. This is the situation with rules of the form

A	$\rightarrow$	B C
C	$\rightarrow$	0

i.e where the empty category appears *in the tail* of the rhs.

Modifying the Left Corner Parser

The only modification is to add an extra clause to parse/3

parse(+C,+P1,+P):*Parse a constitutent of category C between text indices P1 and P.*

```
parse(C,[W|Rest],P) :-
 word(K,W),
 complete(K,C,Rest,P).

parse(C,S2,S) :-
 rule(X,[ ]),
 complete(X,C,S2,S).
```

In other words, if X is a possibly empty category, it is permissible for it to “complete” at any time by doing nothing.

Unfortunately, this will not handle the empty determiner rule, where the empty category appears as the leftmost element of the rewrite.

With such rules the parser loops.

Looping with Empty Categories

S	→	NP VP
VP	→	V NP
NP	→	D N
D	→	0

The problem arises whenever it tries a parse that does not succeed.

Suppose the parser is looking for a VP which isn't there. This means that it failed to accept the first word. Instead, it accepts a null determiner and hypothesises the structure

S(VP(NP(0,...

It now tries to complete the NP by parsing the N top down. Now suppose the N isn't there either.

S(VP(NP(0,NP(0,...

etc.

Introducing Links

The above problem occurs because of this clause

```
parse(C,S2,S) :-
 rule(X,[ ]),
 complete(X,C,S2,S).
```

which allows the parser to accept an empty category anywhere. One solution is to constrain the parser by adding a table of links which specifies what kind of constituents can appear at the beginning of other constituents. For instance, from the rules

S	→	NP VP
NP	→	D N
VP	→	V NP

we can infer the links

```
link(np,s). link(d,np).
link(v,vp). link(X,X).
```

The last clause says that any constituent can begin with itself.

Modifications to parse/3

We modify parse/3 so that there is a link between C, the constituent it is trying to parse and K, the category of the word it is trying to accept:

```
parse(C,[W|Rest],P) :-
 word(K,W),
 link(K,C),
 complete(K,C,Rest,P).

parse(C,S2,S) :-
 rule(K,[ ]),
 link(K,C),
 complete(K,C,S2,S).
```

Inherent Non-Termination

Note that this modification is not an infallible guarantee against looping. A left-corner parser can still loop given:

$$
\begin{array}{lll}
\text{NP} & \rightarrow & \text{det NP} \\
\text{det} & \rightarrow & 0
\end{array}
$$

Rules such as these do indeed specify an infinite number of parses for each string, so infinite backtracking is the appropriate behaviour for a grammar of this sort.