CSE6390 3.0 Special Topics in AI & Interactive Systems II
Introduction to Computational Linguistics
Insructor: Nick Cercone – 3050 CSEB – nick@cse.yorku.ca
Tuesdays,Thursdays 10:00-11:30 – South Ross 104
Fall Semester, 2010

Left Corner Parsing

We will now look at examples of top-down and bottom-up processing which show what kind of information we use to make decisions in these two different processing strategies, what kind of information we ignore with these strategies, and how it can happen that we go wrong, because of that.

Going Wrong with Top-down Parsing

Assume that we have the following grammar fragment

S	→	NP VP
NP	→	Det N | PN
VP	→	IV
Det	→	the
N	→	robber
PN	→	Vincent
IV	→	died

and that we want to use it to top-down recognize the string *vincent died*. Proceeding in a top-down manner, we would first expand S to NP VP. Next we would check what we can do with the NP and find the rule NP→Det N. We would therefore expand NP to Det N. Then we either have to find a lexical rule to relate *vincent* to the category Det, or we have to find a phrase structure rule to expand Det. Neither is possible, so we would backtrack checking whether there are any alternative decisions somewhere.

So, when recognizing in a top-down manner, we totally ignore what the actual input string looks like. We start from some non-terminal symbol and then use rules to rewrite that symbol. Only when we can't apply any rules anymore, we check whether what we have produced matches with the input string.

Here is part of the trace that we will get when trying to recognize *vincent died* with a simple top-down recognizer in Prolog. You can see how Prolog first tries to use the first rule to expand the noun phrase. And only when Prolog realizes that Det leads into a dead-end does it try the next NP rule NP→Det N.

```
Call: (7) recognize_topdown(s, [vincent, died], []) ?
Call: (8) matches([np, vp], [vincent, died], []) ?
Call: (9) recognize_topdown(np, [vincent, died], _G579) ?
Call: (11) recognize_topdown(det, [vincent, died], _G585) ?
Fail: (11) recognize_topdown(det, [vincent, died], _G585) ?
Call: (11) recognize_topdown(pn, [vincent, died], _G582) ?
Exit: (11) recognize_topdown(pn, [vincent, died], [died]) ?
Exit: (9) recognize_topdown(np, [vincent, died], [died]) ?
Call: (10) recognize_topdown(vp, [died], _G582) ?
.
.
.
```

Going Wrong with Bottom-up Parsing

Top-down parsing starts with some goal category that it wants to recognize and ignores what the input looks like. In bottom-up parsing, we essentially take the opposite approach: we start from the input string and try to combine words to constituents and constituents to bigger constituents using the grammar rules from right to left. In doing so, any consituents that can be built are built; no matter whether they fit into the constituent that we are working on a the moment or not. No *top-down* information of the kind ``we are at the moment trying to built a sentence'' or ``we are at the moment trying to built a noun phrase'' is taken into account. Let's have a look at an example.

Say, we have the following grammar fragment:

S	→	NP VP
NP	→	Det N
VP	→	IV | TV NP
TV	→	plant
IV	→	died
Det	→	the
N	→	robber

Note, how *plant* is ambiguous in this grammar: it can be used as a common noun or as a transitive verb. If we now try to bottom-up recognize *the plant died*, we would first find that *the* is a determiner, so that we could rewrite our string to *Det plant died*. Then we would find that *plant* can be a transitive verb giving us *Det TV died*. *Det* and *TV* cannot be combined by any rule. So, *died* would be rewritten next, yielding *Det TV IV* and then *Det TV VP*. Here, it would finally become clear that we took a wrong decision somewhere: nothing can be done anymore and we have to backtrack. Doing so, we would find that *plant* can also be a noun, so that *Det plant died* could also be rewritten as *Det N died*, which will eventually lead us to success.

9.1.3 Combining Top-down and Bottom-up Information

As the previous two examples have shown, using a pure top-down approach, we are missing some important information provided by the words of the input string which would help us to guid our decisions. However, similarly, using a pure bottom-up approach, we can sometimes end up in dead ends that could have been avoided had we used some bits of top-down information about the category that we are trying to build.

The key idea of left-corner parsing is to combine top-down processing with bottom-up processing in order to avoid going wrong in the ways that we are prone to go wrong with pure top-down and pure bottom-up techniques. Before we look at how this is done, you have to know what is the *left corner* of a rule. The left corner of a rule is the first symbol on the right hand side. For example, NP is the left corner of the rule S→NP VP, and IV is the left corner of the rule VP→IV. Similarly, we can say that Vincent is the left corner of the lexical rule PN→Vincent.

A left-corner parser alternates steps of bottom-up processing with top-down predictions. The *bottom-up* processing steps work as follows. Assuming that the parser has just recognized a noun phrase, it will in the next step look for a rule that has an NP as its left corner. Let's say it finds S→NP VP. To be able to use this rule, it has to recognize a VP as the next thing in the input string. This imposes the *top-down* constraint that what follows in the input string has to be a verb phrase. The left-corner parser will continue alternating bottom-up steps as described above and top-down steps until it has managed to recognize this verb phrase, thereby completing the sentence.

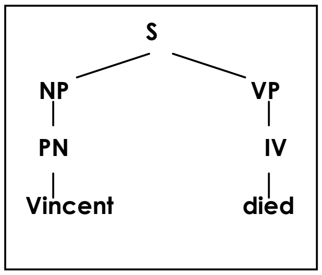
A left-corner parser starts with a top-down prediction fixing the category that is to be recognized, like for example S. Next, it takes a bottom-up step and then alternates bottom-up and top-down steps until it has reached an S.

To illustrate how left-corner parsers work, let's go through an example. Assume that we again have the following grammar:

S	→	NP VP
NP	→	Det N | PN
VP	→	IV
Det	→	the
N	→	robber
PN	→	Vincent
IV	→	died

Now, let's look at how a left-corner recognizer would proceed to recognize *vincent died.*

1.) Input: *vincent died*. Recognize an S. (Top-down prediction.)
2.) The category of the first word of the input is PN. (Bottom-up step using a lexical rule.)
3.) Select a rule that has PN at its left corner: NP→PN. (Bottom-up step using a phrase structure rule.)
4.) Select a rule that has NP at its left corner: S→NP VP. (Bottom-up step.)
5.) Match! The left hand side of the rule matches with S, the category we are trying to recognize.
6.) Input: *died*. Recognize a VP. (Top-down prediction.)
7.) The category of the first word of the input is IV. (Bottom-up step.)
8.) Select a rule that has IV at its left corner: VP→IV. (Bottom-up step.)
9.) Match! The left hand side of the rule matches with VP, the category we are trying to recognize.



Make sure that you see how the steps of bottom-up rule application alternate with top-down predictions in this example. Also note that this example can be used to illustrate how top-down parsers can go wrong and that, in contrast to the p-down parser, the left-corner parser does not have to backtrack with this example.