— LECTURE NOTES —

On Generalized Left Corner Parsing

1 Preface: Predicted and Completed Categories

1 We have observed already certain technical ways in which top down and bottom up parsers are polar opposites of each other. Intuitively, of course, this is how it should be.

We have also observed that we can fold the left corner parsing strategy into this family by recognizing the connection between parsing and tree traversal (which is otherwise obscured by the necessity of making the strategy run efficiently).

In these notes we will explore some generalizations of these ideas and see how different parsing strategies can be seen as formally related specializations of one very general system.

2 The essential concepts underlying the family of parsing strategies that we have observed so far are the concepts of *predicted category* and *completed category*. It is in these terms that we can see how the left corner strategy combines elements of the top down and bottom up strategies: once the left corner category is completed, its sister categories (according to the selected grammar rule) are predicted. In this way a category is recognized (completed) by recognizing (completing) ever larger left corners.

3 Using these terms, we can see the *scan* rule of the top down strategy as taking a *predicted* preterminal category and *completing* it by verifying it against the input. In fact, if we were willing to tolerate a slight inefficiency, and allow proper grammar rules of the form `rule(det,[the]).`, then we could see the scan rule as *cancelling* or *checking* a predicted occurrence of `the` against a completed occurrence of `the` found in the input. Once the prediction is completed, it is removed from the stack. So we have the following fact about the top down strategy:

> Top Down: *The stack only ever contains predicted categories.*

4 As you would expect, the exact opposite could be said of the *shift* rule in the bottom up strategy: here we are starting with a completed category (again, we take words from the input to be the paradigm case of completed categories). We use completed categories found on the stack to trigger the application of a grammar rule which replaces a string of completed categories with another single completed category. This *reduce rule* is the exact polar opposite of the top down strategy's *expand rule*, which takes a single predicted category off the stack, and replaces it with a string of new predicted categories.

> Bottom Up: *The stack only ever contains completed categories.*

5 In our treatment of the left corner strategy, we avoided any mention of the stack or what was on it. The reason for this is rather complex, and you needn't digest it all right now: in order to make a left corner strategy run in an essentially top down environment (Prolog), we had to transform the recognizer in such a way that it became a top down recognizer simulating a left corner recognizer. That way we could use Prolog's stack (which always only holds predicted goals, right?) implicitly and not worry about these issues of predicted and completed categories.

Because, if we want to explicitly represent the stack that a left corner recognizer uses, it will have to contain *both* predicted and completed categories.

6 From now one we adopt the following notation. Given a category symbol A, we will represent a predicted occurrence of A as $\overline{A}$, and a completed occurrence of A simply as A.

2 A Little Bit of Proof Theory

7 We will see in this section how to express a generalized approach to parsing as a system of *inference rules*. The different parsing strategies we have seen, and many more, emerge from this general inference system according to the way in which we use grammar rules in doing this "grammatical inference".

8 Our background knowledge resources will be presented as a grammar, including lexical productions of the form *Det* $\longrightarrow$ *the*. However, we will enrich our grammars with a little notation which splits the right hand side of a rule in two pieces. Formally, given a rule $A \longrightarrow \alpha\beta$, we will write $A \longrightarrow \alpha \,{\scriptstyle\Delta}\, \beta$ to express the fact that in the strategy we are thinking about, we must have completed α before we will allow ourselves to predict β.

Of course, in all cases, when we have completed β, we have completed A as well. This is true no matter what strategy we choose.

We will allow either α or β to be empty.

9 As usual, the objects we will be interested in are pairs consisting of a *stack* and the unused portion of the *input string*.

Given a stack Γ and a string w remaining to be processed, we represent a configuration of the machine we are reasoning about as $(w \mid \Gamma)$.

We write the top of the stack on the left.

10 An *inference rule* is a schema, with some number of assumptions and a single conclusion separated by a horizontal line:

$$\frac{\textit{Assumption}\ 1 \quad \ldots \quad \textit{Assumption}\ n}{\textit{Conclusion}}$$

For example, we could express the *shift rule* as an inference rule like the following.

$$\frac{(aw \mid \Gamma)}{(w \mid a\Gamma)}\,\textit{shift}$$

(Note that we write the stack as a string, with the top of the stack on the left.)

Assumptions for our inference rules can be either states of the machine or (annotated) grammar rules.

11 Leftmost and Rightmost Derivations One technical point needs to be cleared up before we proceed. We have noted already that top down strategies simulate a leftmost derivation, while bottom up strategies simulate a rightmost derivation (in reverse).

Since we must handle the simulation of both leftmost and rightmost derivations with a single mechanism, we will run into problems with the order in which categories appear on the stack. We have seen this problem already: the order in which categories appear on the stack in bottom up processing is the reverse of the order in which they appear on the right hand side of a grammar rule. If we wrote the top of the stack on the right this problem would go away, but now we would have the same problem with top down strategies: the first category popped off the stack would be the rightmost element of the right hand side of some previously used grammar rule.

So we will use the notation α^r to express the reversal of string α. This will help us express the relation between the top of the stack and the right hand side of a grammar rule in a convenient way, but it should be emphasized that there is no really beautiful way to do this in a completely general way.

12 One possible system of inference rules (not the one we will end up using) is the following.

$$\frac{(aw \mid \Gamma)}{(w \mid a\Gamma)}\,\textit{shift} \qquad \frac{(w \mid A\overline{A}\Gamma)}{(w \mid \Gamma)}\,\textit{attach}$$

$$\frac{(w \mid \alpha\Gamma) \quad A \longrightarrow \alpha^r \,{\scriptstyle\vartriangle}\, \beta}{(w \mid \overline{\beta}A\Gamma)}\,\textit{glc}$$

This is a nice simple system, and we will refer to it from time to time as the *arc standard* system, but the system we will actually use "compiles away" the *attach* rule into alternative versions of the other two. This gives us a much better approximation of the top down strategy. (Observe that the *glc* rule always puts a completed category on the stack, but normally top down recognizers only put predicted categories on the stack.) Also the improvement in stack use is, in some strategies, overwhelming. We refer to this system as the *arc eager* system.

$$\frac{(aw \mid \Gamma)}{(w \mid a\Gamma)}\ shift \qquad \frac{A \longrightarrow \alpha^r \,_\vartriangle\, \beta \quad (w \mid \alpha\Gamma)}{(w \mid \overline{\beta} A\Gamma)}\ glc$$

$$\frac{(aw \mid \overline{a}\Gamma)}{(w \mid \Gamma)}\ shift{+}attach \qquad \frac{A \longrightarrow \alpha^r \,_\vartriangle\, \beta \quad (w \mid \alpha\overline{A}\Gamma)}{(w \mid \overline{\beta}\Gamma)}\ glc{+}attach$$

13 Strategies Different processing strategies are implemented simply by moving the $_\vartriangle$ in the annotated grammar rules.

14 It is often convenient to think of the position of the dot as having been specified by a function mapping grammar rules to dot positions. Let position 0 be the position at the left edge of the right hand side.

The *top down* strategy is implemented by the function F_{td} that places the $_\vartriangle$ in position 0, for every grammar rule.

The *bottom up* strategy is implemented by the function F_{bu} that places the $_\vartriangle$ at the right edge of the right hand side (position $|\alpha|$ for all rules $A \longrightarrow \alpha$).

The *left corner* strategy is implemented by the function F_{lc} that places the $_\vartriangle$ in position 1, for every grammar rule with a non-empty right hand side, and position 0 otherwise.

3 Some Examples

15 Example 1 Assume we have a grammar G that can generate the sentence $w =$ *the villain heard the hero sneeze* under the usual oversimplified analysis. Let $w_1 =$ *villain heard the hero sneeze*, $w_2 =$ *heard the hero sneeze*, etc. Apply mark-placement function F_{td} to G. So α and α^r in the rules *glc* and *glc+attach* will always be ϵ.

Assume for the moment that we can take as given the assumption $(w \mid \overline{S})$, that is, we can always start things off by predicting that the sentence is an S, and proceed to work out the consequences of that assumption.

We have the following proof, of which I sketch only the beginning.

$$\cfrac{Det \longrightarrow {}_\vartriangle the \qquad \cfrac{NP \longrightarrow {}_\vartriangle Det\ N \qquad \cfrac{S \longrightarrow {}_\vartriangle NP\ VP \quad (w \mid \overline{S})}{(w \mid \overline{NP}\,\overline{VP})}\ g+a}{(w \mid \overline{Det}\,\overline{N}\,\overline{VP})}\ g+a}{\cfrac{(w = the\, w_1 \mid \overline{the}\,\overline{N}\,\overline{VP})}{(w_1 \mid \overline{N}\,\overline{VP})}\ s+a}\ g+a$$

The final output of the proof will be the empty state $(\epsilon \mid \epsilon)$.

All this is to be expected, since predicted categories are essentially the same animals as (sub)goals in a Prolog proof; the empty state here plays exactly the same goal as the empty clause in a Prolog program, for example.

16 Example 2 Assume w and G as above. Now apply the left corner marking function F_{lc} to G. As before the initial state is $(w \mid \overline{S})$. Now however we cannot apply rule *glc+attach*, since α is not empty. If we are trying to use the rule $S \longrightarrow NP \vartriangle VP$, then $\alpha = NP$. In fact, the only rule that applies is *shift*.

$$\dfrac{NP \longrightarrow Det \vartriangle N \qquad \dfrac{Det \longrightarrow the \vartriangle \qquad \dfrac{(the\, w_1 \mid \overline{S})}{(w_1 \mid the\, \overline{S})}\, shift}{(w_1 \mid Det\, \overline{S})}\, glc}{(w_1 \mid \overline{N}\, NP\, \overline{S})}\, glc$$

Note the mix of predicted and completed categories on the stack in the last step. Note in particular their order. We won't be able to do anything with the "completed" *NP* until we get the predicted $\overline{N}$ out of the way. So in a way the stack expresses an implication: if we can complete the $\overline{N}$, then we have completed the *NP*, but not until then.

$$\dfrac{S \longrightarrow NP \vartriangle VP \qquad \dfrac{N \longrightarrow villain \vartriangle \qquad \dfrac{(villain\, w_2 \mid \overline{N}\, NP\, \overline{S})}{(w_2 \mid villain\, \overline{N}\, NP\, \overline{S})}\, shift}{(w_2 \mid NP\, \overline{S})}\, glc+attach}{(w_2 \mid \overline{VP})}\, glc+attach$$

And so on.

Continuing the proof is left as an exercise, but you may be interested in carrying it out. If you do, observe carefully how the stack grows when you use the left corner strategy. Or rather, fails to grow.

17 Example 3 To see how the bottom up strategy works in this setting note first that the *shift* rule is already part of our inference system. Note also that β will always be empty, since the $\vartriangle$ will always be all the way to the right. So the entire right hand side of the rule, reversed, will play the role of α in the *glc* rule. And, because β is empty, the *glc* rule will always act to replace the right hand side, reversed (that's α) on the top of the stack with the left hand side (that's A, a completed, not predicted, category).

18 Arc Standard Left Corner Recognition In this example, we go back to our left corner marking function F_{lc}, but we now use the *arc standard* package of rules. Observe how the memory requirements of the left corner strategy change.

The first part of the proof is the same as in Example 2, since we use only the *shift* and *glc* rules, which are common to both systems.

$$\dfrac{NP \longrightarrow Det \vartriangle N \qquad \dfrac{Det \longrightarrow the \vartriangle \qquad \dfrac{(the\, w_1 \mid \overline{S})}{(w_1 \mid the\, \overline{S})}\, shift}{(w_1 \mid Det\, \overline{S})}\, glc}{(w_1 \mid \overline{N}\, NP\, \overline{S})}\, glc$$

Now, where Example 2 used *glc+attach*, things start to diverge.

$$\dfrac{S \longrightarrow NP_{\vartriangle}\ VP \qquad \dfrac{\dfrac{N \longrightarrow \textit{villain}_{\vartriangle} \qquad \dfrac{(\textit{villain}\ w_2 \mid \overline{N}\, NP\, \overline{S})}{(w_2 \mid \textit{villain}\ \overline{N}\, NP\, \overline{S})}\ \textit{shift}}{(w_2 \mid N\, \overline{N}\, NP\, \overline{S})}\ \textit{glc}}{(w_2 \mid NP\, \overline{S})}\ \textit{attach}}{(w_2 \mid \overline{VP}\, S\, \overline{S})}\ \textit{glc}$$

As you can see, the pair $S\,\overline{S}$ is "protected" from the application of the *attach* rule by the predicted category $\overline{VP}$. We will not be able to apply *attach* until we have built and attached the VP, which could take a long time.

In fact, under the arc standard approach, the left corner strategy behaves no better than the bottom up strategy, at least in terms of its memory use.

19 Some Facts Assuming the use of the *arc eager* system:

1. On left branching structures, top down strategies use lots of stack space. In fact, if the grammar allows arbitrarily deep left branching constructions, then top down recognition with that grammar will cost arbitrarily much stack space.

2. On right branching structures, bottom up strategies use lots of stack space. In fact, if the grammar allows arbitrarily deep right branching constructions, then bottom up recognition with that grammar will cost arbitrarily much stack space.

3. Left corner strategies use bounded stack space for both left and right branching structures.

These facts are significant because natural languages tend to use both left and right branching structures, but they avoid center embedded structures. This makes parsers based on the left corner strategy very appealing candidates for natural language processing.

20 Another Fact Given a grammar with empty productions ($A \longrightarrow \epsilon$), both bottom up *and left corner* strategies have infinite search spaces.

In other words, the left corner strategy uses the stack much more efficiently than the bottom up strategy does, but they both share the same Achilles' heel.

This emphasizes that parsers *based on* the left corner strategy are a good choice for NLP. More needs to be done. One thing that suggests itself immediately is to use a top down *oracle* that can tell whether any given leaf (including an empty A) could possibly be a left corner of the current constituent.

Consider pursuing this for your Hausarbeit . . .