ECE251: Programming Languages & Translators Fall 2010

Lecture 12 — October 7, 2010

Patrick Lam *version 1*

Fixing Grammars

In practice, the two key obstacles which will prevent you from generating top-down parsers are ambiguity and left-recursion. Let's talk about working around these. (see pp. 82–86 of text)

Ambiguity part 1: expressions. We saw how to get rid of ambiguity for expressions by stratifying that grammar into expressions, terms and factors. That is a fairly common fix. There are other fixes (e.g. `bison` supports explicit precedence declarations, or you can provide custom code to do precedence in your ANTLR grammar), but they are beyond the scope of this class.

Ambiguity part 2: dangling else. The other common source of ambiguity is the "dangling else" problem. What are the two possible parse trees for this input?

$$\texttt{if } (e_1) \texttt{ then if } (e_2) \texttt{ then } S_1 \texttt{ else } S_2$$

I have to tell you about the standard grammar-rewriting fix. But no one actually uses it.

$$\begin{array}{rcl}
\textit{stmt} & := & \textit{balanced\_stmt} \mid \textit{unbalanced\_stmt} \\
\textit{balanced\_stmt} & := & \texttt{if } \textit{cond} \texttt{ then } \textit{balanced\_stmt} \texttt{ else } \textit{balanced\_stmt} \\
 & \mid & \textit{other\_stmt} \\
\textit{unbalanced\_stmt} & := & \texttt{if } \textit{cond} \texttt{ then } \textit{stmt} \\
 & \mid & \texttt{if } \textit{cond} \texttt{ then } \textit{balanced\_stmt} \texttt{ else } \textit{unbalanced\_stmt}
\end{array}$$

You still can't parse this grammar top-down with finite lookahead because of the common prefix between the productions (but ANTLR can!). And it makes the grammar harder to understand and maintain.

Real Fixes. Perhaps the best fix is to avoid including such ambiguity in your grammar; for instance, you can require an explicit `end` after every `if`.

If you don't get that choice, then you can feed the ambiguous grammar to your favourite parser generator. It'll give a warning, but then it'll do the right thing (match the closest `if`). `bison` will give a shift-reduce warning, while ANTLR says the grammar is ambiguous.

You can allegedly suppress the warning in ANTLR by giving an option:

```
stmt : 'if' expr 'then' stmt
       (options {greedy=true;} : 'else' stmt)?
     | ... ;
```

or you can tell it how to resolve the conflict with its look ahead in this case:

```
stmt : 'if' expr 'then stmt
       (('else') => 'else' stmt)?
     | ... ;
```

but I won't really explain these.

Left-recursion. The second big problem is left-recursion, which is never good for top-down parsers. For instance,

$$\begin{aligned} \textit{id\_list} &:= \textit{id\_list\_prefix} \\ \textit{id\_list\_prefix} &:= \textit{id\_list\_prefix}\ \texttt{','}\ \texttt{id} \mid \texttt{id} \end{aligned}$$

is left-recursive because the *id_list_prefix* may itself lead to an *id_list_prefix*. A recursive-descent implementation would loop infinitely.

Indirect left-recursion is also possible, e.g. $A := B; B := A$.

We can remove left-recursion by rewriting the productions. Unlike for ambiguous grammars, we often rewrite to eliminate left-recursion in practice. This equivalent grammar is not left-recursive:

$$\begin{aligned} \textit{id\_list} &:= \texttt{id}\ \textit{id\_list\_suffix} \\ \textit{id\_list\_suffix} &:= \texttt{','}\ \texttt{id}\ \textit{id\_list\_suffix} \mid \varepsilon \end{aligned}$$

Common prefixes. A non-problem with ANTLR is common prefixes. We've manually implemented parsers with one token of lookahead—LL(1). However, the following grammar requires LL(2). That is, two tokens of lookahead suffice to top-down parse the following grammar.

$$\begin{aligned} \textit{stmt} &:= \texttt{id}\ \texttt{'='}\ \textit{expr} \\ &\mid\ \ \texttt{id}\ \texttt{'('}\ \textit{argument\_list}\ \texttt{')'} \end{aligned}$$

However, no fixed amount of lookahead suffices for this (artificial) grammar:

$$\begin{aligned} \textit{stmt} &:= \texttt{id}*\ \texttt{'='}\ \textit{expr} \\ &\mid\ \ \texttt{id}*\ \text{'('}\ \textit{argument\_list}\ \texttt{')'} \end{aligned}$$

ANTLR allows infinite look-ahead (LL(*)) using a DFA.