ECE251: Programming Languages & Translators Fall 2010

Lecture 11 — October 6, 2010

Patrick Lam *version 1*

I found the algorithms for FIRST and FOLLOW in the book really hard to understand, so I found a better description for you on the Internet. In fact, you don't need to know FOLLOW sets.

`http://dragonbook.stanford.edu/lecture-notes/Stanford-CS143/07-Top-Down-Parsing.pdf`

These algorithms use the notion of "nullable". A *nullable* nonterminal is one that can derive ε, i.e. you don't have to consume any terminals to parse that nonterminal. We've seen ε in examples before; e.g. whenever you have an optional construct in the EBNF.

Constructing FIRST sets. The FIRST sets are easier to compute. Here are some rules for computing them. For each rule $A := X_1 X_2 \ldots X_n$, we add to FIRST(A) as follows:

1. If X_1 is a terminal, add X_1 to FIRST(A); done.
2. Otherwise, X_1 is a nonterminal. Recursively add FIRST(X_1) $- \varepsilon$ to FIRST(A).
3. Furthermore, if X_1 is nullable, then keep on going: add FIRST(X_2) $- \varepsilon$ to FIRST(A) also. If X_2 is nullable, add FIRST(X_3) $- \varepsilon$, and so on.
4. If A itself is nullable (i.e. all of the X_is are nullable), then add ε to FIRST(A).

The FIRST sets are the primary tools for constructing recursive-descent parsers, but they are not sufficient for creating *table-driven* parsers in the presence of nullable nonterminals. However, we are not going to talk about these parsers, so I don't need to explain FOLLOW sets.

Our manual parser generation technique only works when all the FIRST sets are disjoint. ANTLR gets around this limitation using more lookahead.

Worked Example. Here is a simple (contrived) grammar, for which we'll compute FIRST. (potential exam-style question!)

$$\begin{aligned} S &:= AB \\ A &:= Cz \mid \varepsilon \\ B &:= xB' \\ B' &:= zACB' \mid \varepsilon \\ C &:= y \mid \varepsilon \end{aligned}$$

Which nonterminals are nullable?

We can then compute FIRST sets.

$$\begin{aligned} \text{FIRST}(C) &= \\ \text{FIRST}(B') &= \\ \text{FIRST}(B) &= \\ \text{FIRST}(A) &= \\ \text{FIRST}(S) &= \end{aligned}$$

Using FIRST sets to build recursive-descent parsers

```
// A := A1 | A2 | ..., all Ai are not nullable
void nonterminal_A(void) {
    if (accept(FIRST(A!))) {
      // consume terminals and nonterminals for A1
           // terminals by calling getsym()
           // nonterminals by calling appropriate function
    } else if (accept(FIRST(A2))) {
      // same here
    } ... { // other elements of FIRST set
    } else {
      // syntax error
    }
}

// B := B1 | B2 | ..., with an epsilon or nullable
void nonterminal_B(void) {
    if (accept(FIRST(B1))) {
      // same as for A
    } else if (accept(FIRST(B2))) {
      // ...
    } else if (accept(FIRST(Bn))) {
      // consume terminals and nonterminals for Bn
      return;
    } else {
      // do nothing, to consume an epsilon
      return;
    }
}
```