ECE251: Programming Languages & Translators Fall 2010

Lecture 8 — September 28, 2010

Patrick Lam *version 1*

Context-Free Grammars

Language syntax is always specified using context-free grammars. We saw one example in the assignment, as well as a context-free grammar for something else earlier in the course. (What was it?). Here are a couple more examples.

Arithmetic Expressions. Another classic example of a CFG is for arithmetic expressions.

$$\begin{array}{rcl} E & := & E \texttt{ '*' } E \mid E \texttt{ '/' } E \\ & \mid & E \texttt{ '+' } E \mid E \texttt{ '-' } E \\ & \mid & \texttt{'('} E \texttt{ ')'} \mid \mathsf{LIT} \end{array}$$

What are some examples of expressions?

Lists. Lists are easy to specify. However, for technical reasons, it is difficult to parse the following obvious list grammar.

$$\begin{array}{rcl} L & := & L \texttt{ ',' } \mathsf{LIT} \\ & \mid & \mathsf{LIT} \end{array}$$

Another example. You can also write bogus papers using context-free grammars.

`http://pdos.csail.mit.edu/scigen/`

Backus-Naur Form

A context-free grammar consists of:

- a set of *non-terminals*;
- a set of *terminals*;
- a set of *productions*, which map non-terminals to sequences of terminals and non-terminals; and
- a *start symbol*, drawn from the set of non-terminals.

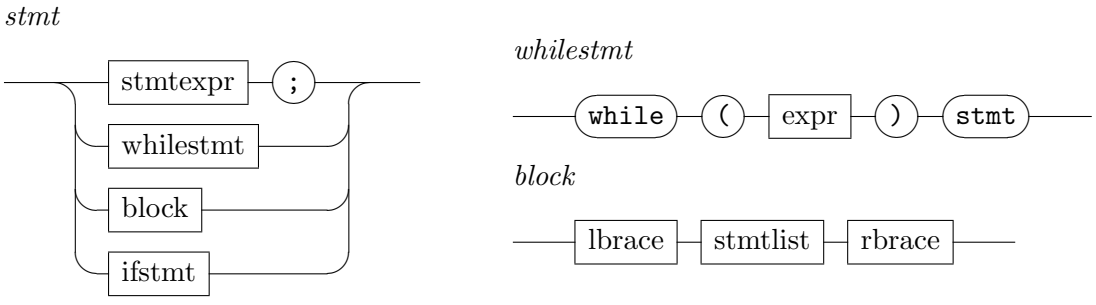
We've specified the above grammars in *Backus-Naur Form*, which is one formalism (or, domain-specific language) for specifying CFGs. In BNF, productions may only contain raw sequences of terminals and non-terminals as targets.

Extended Backus-Naur Form. We can use the notion of "syntactic sugar" to ease the task of writing grammars without adding overhead. In particular, EBNF adds:

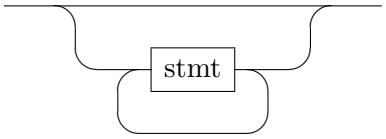
- *parentheses*, e.g.
$$T ::= (\texttt{'+'} \mid \texttt{'-'} \mid \texttt{'*'} \mid \texttt{'/'})\ \mathsf{LIT}.$$
- an *optional* construct, expressed in square brackets [,]; e.g.
$$\textit{constructor\_decl} \ ::= \ [\textit{visibility}]\ \textit{id}\ \texttt{'('}\ [\textit{argument\_list}]\texttt{')'}$$
- The *Kleene star*, as in regular expressions, for zero or more occurences of its target,e.g.
$$L := \mathsf{LIT}\ (\texttt{','}\ \mathsf{LIT})^*$$
Some authors use curly braces { } instead of the *.
- (Sometimes:) the + operator as in regular expressions, for one or more instances.

ANTLR accepts productions in EBNF form. You can mechanically translate productions in EBNF into BNF. How would you convert *constructor_decl* above into BNF?

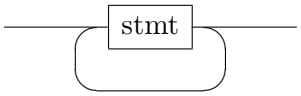
Railroad syntax diagrams. A third formalism for expressing grammars is the railroad diagram. If you search for SQL syntax on the Web, you'll likely encounter one of these.



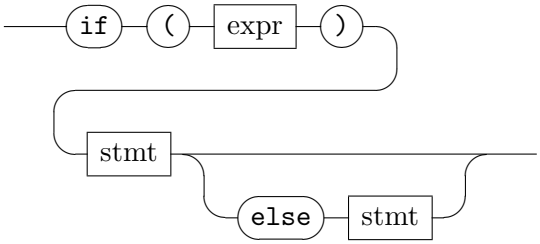
stmtlist0



stmtlist1



ifstmt



Parsing Examples

The primary task of the parser is to convert sequences of tokens into trees. Context-free grammars define the trees that the parser is allowed to construct. We'll see a couple of examples of parse trees; in the next few lectures, we'll discuss algorithms for parsing and how to use parser generators.

Expressions. $2 * 4 + 98/4$

Lists. $4, 1, 5, 9, 2$

Regexps versus (Context-(Free) Grammars)

The **most important point** I want you to learn in this class is that regular expressions can't do everything that you want. Informally:

> "Regular expressions can't count."

If you ever need to match parentheses, or HTML tags, or anything else which may arbitrarily nest (expressions!), regular expressions aren't the tool you're looking for.

(Of course, they may be fine for a one-time non-validating use. However, if you don't want to appear uneducated, don't use them in production for problems they can't solve!)

Proof sketch. Finite automata have a finite amount of memory (limited by the number of states). The FA therefore can't discriminate between different inputs which use more nested parentheses than the FA's memory.