Today

- Syntax Analysis: Grammars

Assignments

- HW-2, R-2 (out, due Thurs)
- HW-3, R-3 (out, due next Thurs)

Announcement

- Quiz on Thursday
• Example of “separation of concerns”
  – each stage does a specific task
  – makes compiler implementation easier to manage

• Our plan is to discuss the “front end” steps ... plus write an “interpreter”
  – lexer, parser, semantic analysis (e.g., type checking)
Syntax Analysis: The Plan …

1. Start with defining a language’s syntax using formal grammars

2. Describe how to check syntax using recursive descent parsing

3. Extend parsing by adding abstract syntax trees (parse trees)
Formal Grammars

Grammars define rules that specify a language’s syntax

- a “language” here means a set of allowable strings

Grammars can be used within

- Lexers (lexical analysis) e.g., numbers, strings, comments
- Parsers (syntax analysis) check if syntax is correct

We’ll look at context-free grammars

- won’t get into all the details
- typically studied in a compiler or theory (formal languages) class
Grammar Rules

Grammar rules define “productions” (“rewritings”)

\[ s \rightarrow a \]

- Here we say \( s \) “produces” (or “yields”) \( a \)
  - \( s \) is a non-terminal symbol (LHS of a rule) ... sometimes as \( \langle s \rangle \)
  - \( a \) is a terminal symbol
  - terminal and non-terminal symbols are disjoint
  - set of terminals is the “alphabet” of the language
  - often there is a distinguished “start” symbol

Concatenation

\[ s \rightarrow ab \]

- \( s \) yields the string \( a \) followed by the string \( b \)

\[ t \rightarrow uv \]
\[ u \rightarrow a \]
\[ v \rightarrow b \]

- \( t \) yields the same exact string as \( s \)
Alternation
\[ s \rightarrow a \mid b \]
- \( s \) yields the string \( a \) or \( b \)

- the same as:
  \[ s \rightarrow a \]
  \[ s \rightarrow b \]

Kleene Star (Closure)
\[ s \rightarrow a^{*} \]
- \( s \) yields the strings with zero or more \( a \)'s

Recursion
\[ s \rightarrow (s) \mid () \]
- \( s \) yields the strings of “well balanced” parentheses
- note that this is not possible to express using \( * \) (closure)
- the opposite is true through ... 
- \( * \) can be implemented using recursion (w/ the empty string ...)

Q: How can we represent \( s \rightarrow aa^{*} \) using recursion?
\[ s \rightarrow a \mid a.s \]
The empty string

\[ s \rightarrow a \cdot b \mid \epsilon \]

- \( s \) yields the strings \( a^i b^i \) for \( i \geq 0 \)
- here \( \epsilon \) denotes the special “empty” terminal

**Exercise**: Try to define some languages (exercise sheet)