Capabilities of context free grammars

Regular expressions vs. context free grammars

- Context free grammars are useful in describing most of the syntax of programming languages but not all
- Regular expressions are capable of describing the syntax of tokens
- Any syntactic construct that can be described using regular expression can also be describe using context free grammar
  - However there ‘s a good reason why regular grammars is still used along with context free grammars to describe the syntax of programming languages

Advantages of using regular expressions

1. The lexical rules usually quite simple and does not require a notation as powerful as context free grammars.
   - Regular expression notation are easier to understand
2. It is efficient to construct recognizers from regular expressions than from context free grammars
3. Separating the syntactic structure of a language into lexical and nonlexical parts provide a convenient way of modularizing the front end of a compiler into two manageable-sized components

- Regular expressions are most useful for describing the structure of lexical constructs
  - Identifiers, constants, keywords,…
- Context free grammars on the other hand are most useful in describing nested structures
  - Balanced parentheses, begin-end's,if-then-else,do-while,…

A grammar for generating conditional statements

- Consider the string

if c1 then s1

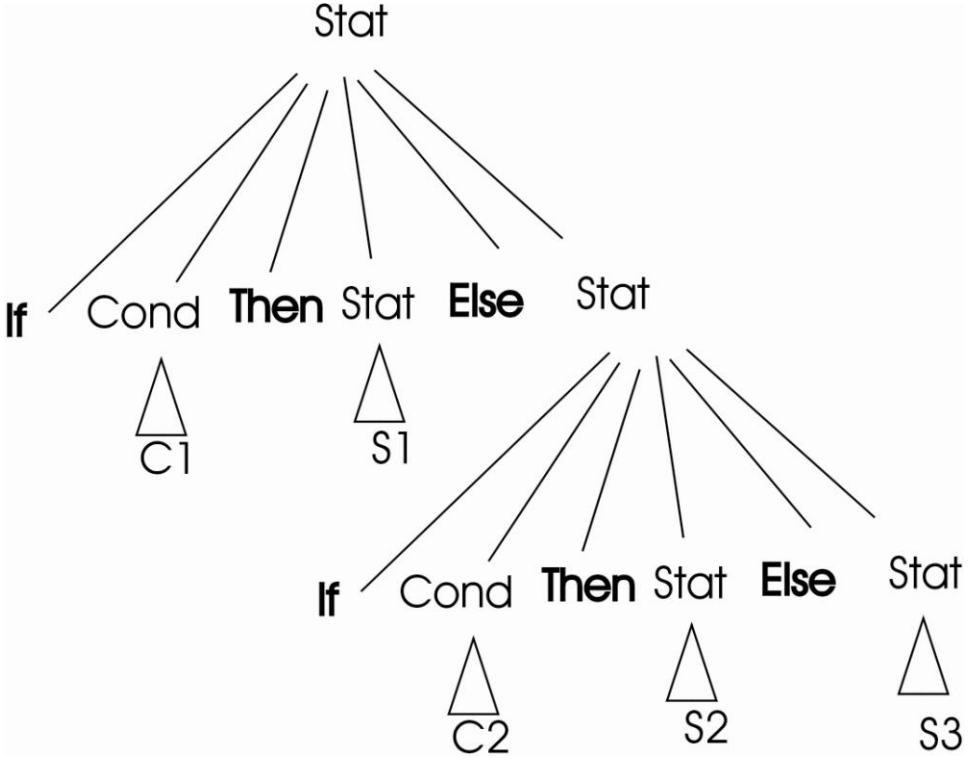
else If c2 then s2 else s3

- And the grammar

Stat→if cond then stat | if cond then stat else stat | other-stat

Parse tree

Parse tree



- Create parse tree(s) for the string

If c1 then if c2 then s1 else s2

Eliminating ambiguity from the if then else grammar

stat→ matched-stat|unmatched-stat

matched-stat→ if cond then matched-stat else matched-stat|other-stat

unmatched-stat→ if cond then stat | if cond then matched-stat else unmatched-stat

- Create a context free grammar for declaring valid identifiers in C language