**Context-Free Languages**

* What features of C or Scheme programs cannot be verified by a DFA?
  + E.g. ∑ = {(, )}
    - L = {w ∈ L\* | w is a string of balanced parens}
    - E.g. ε ∈ L, () ∈ L, (()()) ∈ L etc.
    - Each new state allows one more level of nesting, but no finite # of states allows all levels of nesting – cannot be modelled by DFA
  + E.g. ∑ = {a, b}
    - L = {w: # of a’s in w = # of b’s in w}
    - Impossible recognize arbitrary #’s of a’s and b’s with DFA
* **Context-free languages**
  + Syntax – structure of code
    - Valid ways to combine valid tokens to form C++ statements
  + Semantics – meaning of code
    - What the program does
    - Two programs written in different languages can have the same semantics but different syntax
  + Context-free languages – languages that can be described by a context-free grammar
  + E.g. balanced parens problem:
    - S → ε a word in the language is either empty,
    - S → (S) surrounded by parens, or
    - S → SS the concatenation of 2 words in the language
    - Shorthand: S → ε | (S) | SS
  + E.g. S ⇒ SS ⇒ (S)S ⇒ (S)(S) ⇒ ((S))(S) ⇒ (())(S) ⇒ (())()
    - ⇒ means “derives” – second string can be obtained from first string by one application of a grammar rule
  + A context-free grammar consists of:
    - An alphabet ∑ of terminal symbols
      * Symbols that appear in the output
    - A finite, non-empty set N of non-terminal symbols
      * Abstract components that do not appear in the output
      * An element S ∈ N – start symbol
    - N ∩ ∑ = φ (they have no intersection)
    - V = N ∪ ∑ (“vocabulary”)
    - A finite set P of productions
      * Production has the form A → B, A ∈ N, B → V\*
      * Allows a non-terminal to expand into a repetition of terminals/non-terminals
    - A CFG can use recursion instead of repetition – more powerful
  + Conventions:
    - a, b, c … − elements of ∑ (characters)
    - w, x, y … − elements of ∑\* (words)
    - A, B, C … S … − elements of N (non-terminals)
    - S – start symbol
    - α, β … − elements of V\*
  + To derive a string:
    - Begin with S → replace single non-terminals using single rules → repeat until there are only terminals left
  + A CFG is defined as the collection of all valid strings that can be derived from the start symbol
    - Context-free – means no overlap between blocks; each block can be analyzed in isolation
  + Leftmost derivation – expand leftmost non-terminal first
  + Rightmost derivation – expand rightmost non-terminal first
* **Top-down parsing**
  + Start at S, expand according to rules, produce w
  + S ⇒ α1 ⇒ α2 ⇒ … ⇒ αn ⇒ w
  + A grammar is LL(1) if each cell in the predictor table has at most one entry
    - Left-to-right scan of input
    - Left-most derivations produced
    - 1-symbol look-ahead
* **Bottom-up parsing**
  + Start at w, apply rules in reverse, recover S