Chapter 4 -- Program Semantics

- Syntax -- form of the program (sequence of tokens)
- Semantics -- meaning of the program
  - Does the program "make sense", is it "valid"
  - Things that can not be defined by a CFG
    - call and definition of a function match
    - selecting the proper function from overloaded collection
    - type checking
    - proper declaration (if needed)
  - Runtime semantics -- not typically checked by compiler
  - Interpreter must implement semantics, compiler translates semantics
- Language design includes semantics
- static semantics -- can be enforced at compile time (semantic analysis)
- dynamic semantics -- runtime meaning
  - "dynamic languages" (python, javascript) have less static semantics, postpone checks to runtime
- static semantics -- early checking can lead to better performance

Abstract Syntax Trees

- Parse tree has a lot of "noise"
- Abstract syntax tree more closely describes the computation

  STMS -> STMTS STMT

  STMT -> ID ASSIGN E | READ RL | WRITE WL | WRITELN | lambda

  RL -> RL , ID | ID

  WL -> WL , E | WL , STR | E | STR

  E -> E + T | E - T

  T -> T * F | T / F | T % F

  F -> ID | CONST | (E)
- Parse tree for a := b + c * d ; write a , " ", b ; writeln
- bdcl lex, yacc, AST program
- Read book:
  - abstract grammar : formal definition of AST
  - AST and action rules (similar to yacc)
  - Recursive Descent action rules
  - top down actions

One-Pass compilers

- Some compilers run each phase until done
  - scanner -> parser -> semantic analysis -> optimization -> code generation
- Most do "syntax driven" "One Pass"
  - parser in control
  - calls scanner
  - generates AST, generation of AST does semantic checking
  - calls optimizer / code generater with AST
- Dyamic semantics -- semantics at run time
  - semantic analysis typically does static semantics
    - variables declared, initialized before use
    - types matching
    - return statment on every path (or runtime error)
    - and so forth
  - Dynamic semantics are what happens at run time
  - Book talks about formalization for describing dynamic semantics
  - Don’t have time for a deep dive

Chapter 5 -- Target Machine Architecture

- Book doesn’t have much on this.
- Very important for a code generator
- We may not make it there .... so we’ll ignore it for the time.
- If interested, book has a book companion with a 46 page PDF on chapter 5.