Chapter 7 - Syntax-Directed Translation

Syntax directed translation

- Analysis -- scanner, parser, semantic checking
- Synthesis -- semantic generation, optimizer, codegen

Representation of the parse

- Use the syntax tree representation of parse
  - concrete syntax tree vs abstract syntax tree
  - attributed abstract syntax tree
  - parser builds AST
- Use tree for semantic checks
  - declared variables
  - type checking
  - other restrictions (for example)
    - value parameters may not be assigned
    - overloaded procedure declarations
Concrete vs Abstract syntax trees

S -> ID ASSIGN E
E -> E + T | E - T | T
T -> T * P | T / P | P
P -> CONST | ID | ( E )

Draw trees for:

value <-- 37 * ( T3 + 6 )

Add attributes
Semantic Checks?
Code generation from tree?
Syntax Directed Translation

- Use grammar rules to build AST
- **Bottom-Up (LR)**
  - Build tree from bottom
  - Parse stack contains a forest of partial trees
  - At "reduce rule A -> alpha" time we may:
    - Construct a new node using trees under alpha
    - Just pass a tree up the parse stack

- **Top Down (LL, Recursive descent)**
  - Build tree from top down
  - Build a Node at production prediction time
  - Fill in details as more productions add information
Abstract syntax tree

How does one represent an AST?

- single node type?
- variable part? (union?)

C:
```c
struct node {
    nodekind whichone;
    union attribute {
        ....
    }
    } a;
};
```

C++:

- base class node
- new derived class for each node type?

Example syntax tree: public/cs450/atl0.yacc

Read Chapter 7, we will cover 7.4 later, but read it now.