Programming Assignment 2
- due Friday, February 25

Last Time
- Makefiles
- ambiguous grammars
- grammars for expressions
  - precedence
  - associativity

Today
- wrap up CFGs
- list grammars
- syntax-directed translation

Next Time
- continue syntax-directed translation
- ASTs

Recall our expression grammar

Write an unambiguous grammar for integer expressions involving only addition, multiplication, and parentheses that correctly handles precedence and associativity.

```
expr → expr PLUS term
   |   term

term → term TIMES factor
    |   factor

factor → INTLIT
   |   LPAREN expr RPAREN
```

Extend this grammar to add exponentiation (POW)
Add exponentiation (POW) to this grammar, with the correct precedence and associativity.
List grammars

Example a list with no separators, e.g., A B C D E F G

Another ambiguous example

stmt → IF cond THEN stmt
| IF cond THEN stmt ELSE stmt
| . . .

Given this word in this grammar: if a then if b then s1 else s2
How would you derive it?
Overview of CFGs

CFGs for language definition
- the CFGs we've discussed can generate/define languages of valid strings
  - start by building parse tree & end with some valid string

CFGs for language recognition
- start with string w & end with yes/no answer depending on whether w ∈ L(G)
  - generally use AST instead of parse tree
  - need to translate sequence of tokens (w)

CFGs for parsing
- start with string w & end with parse tree for w if w ∈ L(G)

Syntax-directed translation (SDT)
- translating from a sequence of tokens into a sequence of actions/other form, based on underlying syntax
  - could be: AST, value, type, etc.

To define a syntax-directed translation
- augment CFG with translation rules (can have 1 per production)
  - define translation of LHS non-terminal as a function of
    - constants
    - translations of RHS non-terminals
    - values of tokens (terminals) on RHS

To translate a sequence of tokens using SDT
- build parse tree
- use translation rules to compute translation of each non-term in the parse tree (bottom up) ≤ 2 handle children of node before node
- translation of seq of tokens is the translation of parse tree's root, non-term (i.e., start symbol)

Type of translation can be anything: numeric, string, set, tree, ...
Example: grammar for language of binary numbers

CFG

<table>
<thead>
<tr>
<th>Production</th>
<th>Terminals</th>
</tr>
</thead>
<tbody>
<tr>
<td>$b \rightarrow 0$</td>
<td></td>
</tr>
<tr>
<td>$b \rightarrow 1$</td>
<td></td>
</tr>
<tr>
<td>$b \rightarrow b\ 0$</td>
<td></td>
</tr>
<tr>
<td>$b \rightarrow b\ 1$</td>
<td></td>
</tr>
</tbody>
</table>

Translation rules

<table>
<thead>
<tr>
<th>Production</th>
<th>Terminals</th>
</tr>
</thead>
<tbody>
<tr>
<td>$b_{\text{trans}} = 0$</td>
<td></td>
</tr>
<tr>
<td>$b_{\text{trans}} = 1$</td>
<td></td>
</tr>
<tr>
<td>$b_{1,\text{trans}} = b_{2,\text{trans}} \times 2$</td>
<td></td>
</tr>
<tr>
<td>$b_{1,\text{trans}} = b_{2,\text{trans}} \times 2 + 1$</td>
<td></td>
</tr>
</tbody>
</table>

Example: input string is 10110

Translation is 22

Syntax-directed translation to compute the decimal equivalent of a binary number.
Example: grammar for language of variable declarations

CFG

dclList → ε
  |  decl dclList

decl → type ID ;

type → INT
  |  BOOL

Translation rules

dclList.trans = ""

dclList₁.trans = decl₁.trans + " " + dclList₂.trans

dcl₁.trans = ID.value

Translation (of sequence of tokens) is string containing IDs that have been declared

Example input

int xx;
bool yy;
int zz;

Output

"xx yy zz"
(in any order)

Parse tree
Example: grammar for language of variable declarations

```
CFG
declList \rightarrow \epsilon
| decl declList
decl \rightarrow type ID ;
type \rightarrow INT
| BOOL
```

Translation rules

```
declList.trans = ""
declList.trans = decl.trans + " " + declList2.trans
if (type.trans) decl.trans = ID.value
else decl.trans = ""
typetrans = true
else typetrans = false
```

Modify translation so it only contains IDs declared as INTs

Note:
- different non-terms can have different types as their translation
- translation rules can be conditional