Programming Assignment 2
  • due Tuesday, February 20

Last Time
  • Makefiles
  • ambiguous grammars
  • grammars for expressions
    • precedence
    • associativity
  • grammars for lists

Today
  • syntax-directed translation
  • intro to abstract syntax trees

Next Time
  • implementing ASTs

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

\[
\begin{align*}
  \text{expr} & \rightarrow \text{expr} \text{ PLUS} \text{ term} \\
  & \mid \text{ term} \\
  \text{term} & \rightarrow \text{term} \text{ TIMES} \text{ factor} \\
  & \mid \text{ factor} \\
  \text{factor} & \rightarrow \text{INTLIT} \\
  & \mid \text{LPAREN} \text{ expr} \text{ RPAREN}
\end{align*}
\]

Extend this grammar to add exponentiation (POW)
Add exponentiation (POW) to this grammar, with the correct precedence and associativity.
Overview of CFGs

CFGs for language definition
- the CFGs we've discussed can generate/define languages of valid strings

CFGs for language recognition

CFGs for parsing
Syntax-directed translation
= translating from a sequence of tokens into a sequence of actions/other form, based on underlying syntax

To define a syntax-directed translation
Augment CFG with translation rules
• define translation of LHS non-terminal as a function of
  •
  •
  •

To translate a sequence of tokens using SDT
•
  • use translation rules to compute translation of
  • translation of sequence of tokens is

The type of the translation can be anything:

Note:
Example: grammar for language of binary numbers

<table>
<thead>
<tr>
<th>CFG</th>
<th>translation rules</th>
</tr>
</thead>
<tbody>
<tr>
<td>(b \rightarrow 0)</td>
<td>(b_{\text{trans}} = 0)</td>
</tr>
<tr>
<td>(</td>
<td>1)</td>
</tr>
<tr>
<td>(</td>
<td>b 0)</td>
</tr>
<tr>
<td>(</td>
<td>b 1)</td>
</tr>
</tbody>
</table>
Example: grammar for language of variable declarations

CFG

Translation rules

declList → ε
| decl declList

decl → type ID ;

type → INT
| BOOL

Write a syntax-directed translation for the CFG given above so that the translation of a sequence of tokens is a string containing the ID's that have been declared.
Example: grammar for language of variable declarations

CFG            Translation rules

decList → ε
  | decl decList

decl → type ID ;

type → INT
  | BOOL

Modify the previous syntax-directed translation so that only declarations of type int are added to the output string.
SDT for parsing

Previous examples showed SDT process assigning different types to the translation
- translate tokenized stream to an integer value
- translate tokenized stream to a string

For parsing, we'll need to translate a tokenized stream to an **abstract-syntax tree (AST)**

Abstract syntax trees

**AST** = condensed form of parse tree
- 
- 
- 
- 
-
AST Example

CFG

expr → expr PLUS term
     |  term

term → term TIMES factor
     |  factor

factor → INTLIT
     |  LPAREN expr RPAREN