Lecture topics:
I Syntax definition
II Syntax-directed translation
III Recursive-descent parsers
I Syntax definition

Example grammar G₁:
E → E+E | E-E | E*E | E/E
1 0 1 1 ... 1 9

Grammar concepts:
- nonterminals (e.g., E)
- syntactic variables
- defined in syntax-analyse grammar
- terminals (e.g., +, -, *, /, 0, 1, ..., 9)
- (defined in lexical-analyse grammar
- rule (head → body)
- define nonterminal
- e.g., "production"
- sequence (e.g., E+E)
- alternation (e.g., 0|1|1)
- empty string (E = epsilon)
- start symbol (e.g., E)
- not in syntax grammar:
  +, *, ?, (.), [ ... ]

Parse trees:

ambiguous grammar:
- multiple different parse trees
- for same input

Precedence = binding strength
- *, / have higher precedence than +,
- *, / bind stronger than +,
- "layered-grammar" technique

G₂: E → E+E | E-E | E*E | E/E
T → T*T | T/T | T
F → 0|1|1 ... 1 9

Still ambiguous!
Associativity = grouping direction
+,-,*,/ are all left-associative
(group to the left)

"asymmetric layered grammar" technique
G₃: E → E+T | E-T | T
T → T*T | T/T | T
F → 0|1|1 ... 1 9

Parse tree = concrete syntax tree
AST = abstract syntax tree

I Syntax-directed translation

Running example:
<table>
<thead>
<tr>
<th>infix</th>
<th>post-fix</th>
</tr>
</thead>
<tbody>
<tr>
<td>3+3</td>
<td>3 3+</td>
</tr>
<tr>
<td>1+2*3</td>
<td>1 2 3*+</td>
</tr>
<tr>
<td>1-2-3</td>
<td>1 2 3-</td>
</tr>
</tbody>
</table>

Task: translate infix → post-fix operation
Translation scheme G4:

\[
E \rightarrow E + T \\
E \rightarrow E, T \\
T \rightarrow T, 0 \\
T \rightarrow 0 \\
T \rightarrow 1 \\
T \rightarrow \ldots \\
T \rightarrow \alpha
\]

Simplified attribute

Parent computed from child
During parsing, e.g.,
as part of return value

Inherited attribute
Child computed from parent
After parsing, during tree traversal,
e.g., as field of AST node

Recursive-descent parsers

First attempt: implementation of G4

String pE()
switch (lookahead){
  case '0': case '1': ... : case '9':
    return pT();
  case '+' : case '-':
    return pT();
  case '\\n':
    return pE();

String pR()
switch (lookahead){
  case '0': case '1': ... : case '9':
    return pR();
  case '+':
    return pE();
  case '-':
    return pE();

Problem: left-recursion
leads to stack overflow

"right-recursive predictive grammar" technique

String pE()
switch (lookahead){
  case '0': case '1': ... : case '9':
    return pT();
  case '+' : case '-':
    return pT();
  case '\\n':
    return pE();

String pR()
switch (lookahead){
  case '0': case '1': ... : case '9':
    return pR();
  case '+':
    return pE();
  case '-':
    return pE();

Reminders:
- hw1 due Fr 3/16 at 4 pm
- hw2 due Fr 3/23 at 4 pm
- read 3.1-3.6 for lecture on 3/21
- example solutions