Class Announcements

- Second homework will be posted soon

- We will have around 8 homeworks. Homeworks count for 10% of your overall course grade. **The 2 homeworks with the lowest grades will not count.** Homeworks are meant for exam preparation.

- Everyone should have access to canvas and piazza, including students who signed up using SPNs.

- My office hours will be Fridays, 2:20pm to 4:00pm, in CoRE 305.
Syntax Analysis (Scott 2.3 - 2.5 - skip 2.3.4)

token sequence

```
if  id <a>  <=  id <b>
then id <c>  :=  num <1>
```

parser

```
<if_stmt>

if <expr> then <stmt>

id <a> <= id <b> id <c> := num <1>
```

parse tree
<start> ::= <expr>

<expr> ::= = <expr> + <expr> |
         <expr> - <expr> |
         <expr> * <expr> |
         <expr> / <expr> |
         <expr> ^ <expr> |
         <d> | <l>

<d> ::= 0 | 1 | 2 | 3 | ... | 9

<l> ::= a | b | c | ... | z
Possible Parse Trees

Parse “8 − 3 * 2”:

There are two parse trees for the same input string. Each parse tree is represented by a unique leftmost and unique rightmost derivation. Therefore, there are two leftmost and two rightmost derivations.
Changing the Language to Include Delimiters

\[<\text{expr}> ::= (\text{<expr>}) − (\text{<expr>}) | (\text{<expr>}) \times (\text{<expr>}) | \langle l \rangle | \langle d \rangle\]

\[(8)-(5)*(2)\]
\[((8)-(5))*2\]

Pretty ugly, isn’t it? Is there any other way to disambiguate our expression grammar?
Changing the Grammar to Impose Precedence

<expr> ::= <expr> − <expr> |
         <term>
<term> ::= <term> * <term> |
         <factor>
<factor> ::= 0 | 1 | 2 | 3 | ... | 9
Grouping In Parse Tree
Now Reflects Precedence

Parse “8 − 3 * 2”:

There is only a single possible parse tree for the input string. By changing the grammar, we got a parse tree structure that is (1) unique and (2) matches the operator precedence that we want.
Precedence

- Low Precedence:
  Addition + and Subtraction −

- Medium Precedence:
  Multiplication * and Division /

- Highest Precedence:
  Exponentiation ^

⇒ Ordered lowest to highest in grammar.
Still Have Ambiguity...

3 − 2 − 1 still a problem:

- Grouping of operators of same precedence not disambiguated.
- Non-commutative operators: only one parse tree correct.
Imposing Associativity

Simple grammars with left/right recursion for $-$:

our choices:

\[
\text{<expr> ::= <d> } - \text{<expr>} \ |
\]
\[
\text{<d>}
\]
\[
\text{<d> ::= 0 | 1 | 2 | 3 | ... | 9}
\]

or

\[
\text{<expr> ::= <expr> } - \text{<d>} \ |
\]
\[
\text{<d>}
\]
\[
\text{<d> ::= 0 | 1 | 2 | 3 | ... | 9}
\]
**Associativity**

- Deals with operators of same precedence
- Implicit grouping or parenthesizing
- Left to Right: *, /, +, −
- Right to Left: ^
Complete, Unambiguous Arithmetic Expression Grammar

\[
\text{<start>} ::= \text{<e>}
\]

\[
\text{<e>} ::= \text{<e>} + \text{<t>} | \text{<e>} - \text{<t>} | \text{<t>}
\]

\[
\text{<t>} ::= \text{<t>} * \text{<f>} | \text{<t>} / \text{<f>} | \text{<f>}
\]

\[
\text{<f>} ::= \text{<g>} \ ^\text{<f>} | \text{<g>}
\]

\[
\text{<g>} ::= ( \text{<e>}) | \text{n} | \text{i}
\]

\[
\text{<n>} ::= 0 | 1 | 2 | \ldots | 9
\]

\[
\text{<i>} ::= a | b | c | \ldots | z
\]
Regular vs. Context Free

- All regular languages are context free languages
- Not all context free languages are regular languages

Example:

\[
N ::= X | Y \\
X ::= a | X \ b \\
Y ::= c | Y \ c
\]

is equivalent to:

\[
ab^* | c^+
\]

Is \( \{a^n b^n | n \geq 0 \} \) a context free language?

Is \( \{a^n b^n | n \geq 0 \} \) a regular language?
Regular Grammars

CFGs with restrictions on the shapes of production rules.

**Left-linear:**
N ::= X a b
X ::= a | X b

**Right-linear:**
N ::= b | b Y
Y ::= a b | a b Y
 Complexity of Parsing

Classification of languages that can be specified by specific grammars and recognized by their associated abstract machines.

Complexity:

- Regular grammars: dfas, $O(n)$
- LR grammars: Knuth’s algorithm, $O(n)$
- Arbitrary CFGs: Early’s algorithm, $O(n^3)$
- Arbitrary CSGs: lbas, P-SPACE, COMPLETE
Abstract versus Concrete Syntax

Concrete Syntax (Parse Tree):
representation of a construct in a particular language, including placement of keywords and delimiters

Abstract Syntax Tree (AST):
structure of meaningful components of each language construct
Abstract versus Concrete Syntax

Same abstract syntax, different concrete syntax:

Pascal

\[
\text{while } x <> A[i] \text{ do} \quad \begin{align*}
&i := i + 1 \\
&\text{end}
\end{align*}
\]

C

\[
\text{while ( x != A[i] )} \\
\text{} \quad i = i + 1;
\]
Example

\[
\begin{align*}
\langle S \rangle &::= \langle E \rangle \\
\langle E \rangle &::= \langle E \rangle - \langle T \rangle \mid \langle T \rangle \\
\langle T \rangle &::= \langle T \rangle \ast \text{id} \mid \text{id}
\end{align*}
\]

Consider $A \ast B - C$:
Top-Down Parsing - LL(1)

Basic Idea:

- The parse tree is constructed from the root, expanding **non-terminal** nodes on the tree’s frontier following a left-most derivation.

- The input program is read from left to right, and input tokens are read (consumed) as the program is parsed.

- The next **non-terminal** symbol is replaced by one of its rules. The particular choice **has to be unique**, and uses parts of the input (partially parsed program), for instance the first **token** of the remaining input.
Top-Down Parsing - LL(1) (cont.)

How can we parse (automatically construct a left-most derivation) an input string, for example \texttt{a a a b b b}, using a PDA (push-down automaton) and only the first symbol of the remaining input?

Example:
\[ S ::= a\ S \mid b\ S \mid \epsilon \]

INPUT: \texttt{a a a b b b eof}

Example:
\[ S ::= a\ S\ b \mid \epsilon \]

INPUT: \texttt{a a a b b b eof}
Next Lecture

Things to do:
Start programming in C. Check out the web for tutorials.

Next time:

- More on LL(1) grammars
- Recursive descent parsing
- Examples of syntax directed translation: interpreter, compiler, type checker, static performance predictor