Class Information

▶ Section 1 TA changed from Yikai Zhang to Bo Liu.
▶ Second homework posted. Due this coming Monday 11:55pm EDT.
▶ Six of you haven’t enrolled in piazza site:
  ▶ jiaxu.su8@rutgers.edu,
  ▶ jami.difilippo@rutgers.edu
  ▶ liu.julian@rutgers.edu
  ▶ jonathen.sheffer@rutgers.edu
  ▶ linda.cook@rutgers.edu
  ▶ nicholas.dehais@rutgers.edu

If you have already enrolled using a different email address, please let me know. Otherwise, please enroll in the class in piazza. Please contact me or the TAs if you have any difficulties.
Review: Context Free Grammars (CFGs)

- A formalism for describing languages
- A CFG $G$ is a quadruple $G = \langle T, N, P, S \rangle$:
  1. A set $T$ of terminal symbols (tokens)
  2. A set $N$ of nonterminal symbols
  3. A set $P$ production (rewrite) rules
  4. A special start symbol $S$

- The language $L(G)$ is the set of sentences of terminal symbols in $T^*$ that can be derived from the start symbol $S$:
  $L(G) = \{ w \in T^* | S \Rightarrow^* w \}$

CFGs are rewrite systems with restrictions on the form of rewrite (production) rules that can be used
Review: BNF Syntax

Terminal Symbol: Symbol-In-Boldface
Non-Terminal Symbol: Symbol-In-Angle-Brackets

Production Rule:
Non-Terminal ::= Sequence of Symbols
or
Non-Terminal ::= Sequence | Sequence | ...

Alternative Symbol: |
Empty String: $\epsilon$
Many grammars may correspond to one programming language. Good grammars:

▷ capture the logical structure of the language  
  ⇒ structure carries some semantic information (example: expression grammar)

▷ use meaningful names

▷ are easy to read,

▷ are unambiguous

▷ …

What’s the problem with ambiguity?
“Time flies like an arrow; fruit flies like a banana.”

A grammar \( G \) is ambiguous iff there exist a \( w \in L(G) \) such that there are

1. two distinct parse trees for \( w \), or
2. two distinct leftmost derivations for \( w \), or
3. two distinct rightmost derivations for \( w \).

We want a unique semantics of our programs, which typically requires a unique syntactic structure.
Arithmetic Expression Grammar

\[<\text{start}> ::= <\text{expr}>\]
\[<\text{expr}> ::= <\text{expr}> + <\text{expr}> | \]
\[<\text{expr}> - <\text{expr}> | \]
\[<\text{expr}> * <\text{expr}> | \]
\[<\text{expr}> / <\text{expr}> | \]
\[<\text{expr}> ^ <\text{expr}> | \]
\[<\text{d}> | <\text{l}>\]
\[<\text{d}> ::= 0 | 1 | 2 | 3 | \ldots | 9\]
\[<\text{l}> ::= a | b | c | \ldots | z\]
<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”: 
Precedence

- **Low Precedence:**
  Addition $+$ and Subtraction $-$

- **Medium Precedence:**
  Multiplication $\ast$ and Division $/$

- **Highest Precedence:**
  Exponentiation $\wedge$

$\Rightarrow$ Ordered lowest to highest in grammar.
3 − 2 − 1 still a problem:

▶ Grouping of operators of same precedence not disambiguated.
▶ Non-commutative operators: only one parse tree correct.
Simple grammars with left/right recursion for −:

our choices:

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

or

\[
<\text{expr}> ::= <\text{expr} > \text{−} <d> | <d>
\]
\[
<d> ::= 0 | 1 | 2 | 3 | \ldots | 9
\]
Associativity

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

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

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

\[
\text{<t>} ::= \text{<t>} \ast \text{<f>} \mid \text{<t>} / \text{<f>} \mid \text{<f>}
\]

\[
\text{<f>} ::= \text{<g>} \wedge \text{<f>} \mid \text{<g>}
\]

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

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

\[
\text{<i>} ::= \text{a} \mid \text{b} \mid \text{c} \mid \ldots \mid \text{z}
\]
1. Can't *always* remove an ambiguity from a grammar by restructuring productions

2. An inherently ambiguous language does not possess an unambiguous grammar

3. There is no algorithm that can examine an arbitrary context-free grammar and tell if it is ambiguous, i.e., detecting ambiguity in context-free grammars is an *undecidable* problem
Abstract versus Concrete Syntax

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

Abstract Syntax:
structure of meaningful components of each language construct
Abstract versus Concrete Syntax

Same abstract syntax, different concrete syntax:

Pascal

\[ \textbf{while x <> A[i] do} \]
\[ \quad i := i + 1 \]
\[ \textbf{end} \]

C

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

\[ \text{Consider } A*B-C: \]
Regular vs. Context Free

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

Example:

\[
\begin{align*}
N &::= X \mid Y \\
X &::= a \mid X b \\
Y &::= c \mid Y c
\end{align*}
\]

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:**

\[ N ::= X \ a \ b \]
\[ X ::= a \mid X \ b \]

**Right-linear:**

\[ N ::= b \mid b \ Y \]
\[ Y ::= a \ b \mid a \ b \ Y \]
Next time:

- Read Scott 2.3.1 - 2.3.2 (and some background - chapter on companion site)
- Additional reading: Scott 2.2 and 2.4 (RE, DFA, grammar and language classes)