Class Information

REMINDERS

- First homework deadline: TODAY, February 3, 11:59pm. No late submissions.
- Second homework will be posted tomorrow.
- Don’t forget to work on your Linux skills (ilab)
Review - Simple BNF Grammar ($\mathcal{G}$)

**Terminals** letters, digits, :=

**Nonterminals** `<letter>` `<digit>` `<identifier>` `<stmt>`

**Productions**

1. `<letter>` ::= A | B | C | ... | Z
2. `<digit>` ::= 0 | 1 | 2 | ... | 9
3. `<identifier>` ::= `<letter>` |
   `<identifier>` `<letter>` |
   `<identifier>` `<digit>`
4. `<stmt>` ::= `<identifier>` ::= 0

**Start Symbol** `<stmt>`
A parse tree of $x_2 := 0$ in $G$:

Each internal node is a nonterminal; its children are the RHS of a production for that NT.

The parse tree demonstrates that the grammar generates the terminal string on the frontier.
Grammars are not Unique

Consider $G'$:

**Terminals** letters, digits, :=

**Nonterminals** `<letter> <digit> <ident> <stmt> <letterordigit>`

**Productions**

1. `<letter>` ::= A | B | C | ... | Z
2. `<digit>` ::= 0 | 1 | 2 | ... | 9
3. `<ident>` ::= `<letter>` |
   `<ident> <letterordigit>`
4. `<stmt>` ::= `<ident> := 0`
5. `<letterordigit>` ::= `<letter>` | `<digit>`

**Start Symbol** `<stmt>`
\( G \) and \( G' \) generate the same language, but yield different parse trees.

Example: A parse tree of \( x_2 := 0 \) in \( G' \).
Grammars and Programming Languages

Many grammars may correspond to one programming language.

Good grammars:

- capture the logical structure of the language
  \[ \Rightarrow \text{structure carries some semantic information} \]
  (example: expression grammar)

- use meaningful names

- are easy to read,

- are unambiguous

- \ldots
Definition of Ambiguous Grammars

“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.
Simple Statement Grammar

\(<\text{start}>\) ::= \(<\text{stmt}>\)

\(<\text{stmt}>\) ::= \(<\text{if-stmt}>\) | \(<\text{assign}>\)

\(<\text{if-stmt}>\) ::= \textbf{if} \(<\text{expr}>\) \textbf{then} \(<\text{stmt}>\) | \textbf{if} \(<\text{expr}>\) \textbf{then} \(<\text{stmt}>\) \textbf{else} \(<\text{stmt}>\)

\(<\text{assign}>\) ::= \(<\text{id}>\) ::= \(<\text{d}>\)

\(<\text{expr}>\) ::= \(<\text{id}>\) = 0

\(<\text{d}>\) ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

\(<\text{id}>\) ::= a | b | c | ... | z
Dangling Else Ambiguity

How are nested if statements parsed with this grammar?

```plaintext
if x = 0 then if y = 0 then z := 1 else z := 2
```
Dangling Else Ambiguity

\[
\text{if } x = 0 \text{ then if } y = 0 \text{ then } z := 1 \text{ else } z := 2
\]

How to deal with ambiguity?

1. Change the language to include \textbf{delimiters} (e.g.: new terminal symbol)
   Examples: dangling else, expression grammar

2. Change the grammar
   Example: impose \textbf{associativity} and \textbf{precedence}
   in an arithmetic expression grammar
Changing the Language to Include Delimiters

Algol 68 if statement:

\[
<\text{if-stmt}> ::= \textbf{if} <\text{expr}> \textbf{then} <\text{stmt}> \textbf{fi} \mid
\textbf{if} <\text{expr}> \textbf{then} <\text{stmt}>
\textbf{else} <\text{stmt}>
\textbf{fi}
\]

How would you use this syntax to express the meaning of the two different parse trees for:

\texttt{if }x = 0 \texttt{ then if }y = 0 \texttt{ then }z := 1 \texttt{ else }z := 2
Arithmetic Expression Grammar

<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 “$8 - 3 \ast 2$”: 
Changing the Language to Include Delimiters

<expr> ::= (<expr>) − (<expr>) |

(<expr>) * (<expr>) |

<l> | <d>

(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”: 
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:

\[
\begin{align*}
\textit{<expr>} &::= \textit{<d>} - \textit{<expr>} \\
\textit{<d>} &::= 0 | 1 | 2 | 3 | \ldots | 9
\end{align*}
\]

or

\[
\begin{align*}
\textit{<expr>} &::= \textit{<expr>} - \textit{<d>} \\
\textit{<d>} &::= 0 | 1 | 2 | 3 | \ldots | 9
\end{align*}
\]
Associativity

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

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

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

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

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

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

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

\[ \text{<i>} ::= a | b | c | \ldots | z \]
Dealing with Ambiguity

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
Regular vs. Context Free

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

Example:

\[ N ::= X \mid Y \]
\[ X ::= a \mid X \, b \]
\[ Y ::= c \mid Y \, c \]

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 recognized by specific grammars

Complexity:

- Regular grammars: *dfas*  \( \mathcal{O}(n) \)
- LR grammars: Knuth’s algorithm  \( \mathcal{O}(n) \)
- Arbitrary CFGs: Early’s algorithm  \( \mathcal{O}(n^3) \)
- Arbitrary CSGs: *lbas*  \( \text{P-SPACE} \)
  \( \text{COMPLETE} \)
Next Lecture

Top-down parsing, FIRST and FOLLOW sets, LL(1) grammars, recursive descent parsing

Things to do:

• read Scott, Ch. 2.3 - 2.5 (skip 2.3.3 Bottom-up Parsing)