CS 536

CFGs for Syntax Definition
Roadmap

• Last time
  – Defined context-free grammar basics

• This time
  – CFGs for syntax design
    • Language membership
    • List grammars
    • Resolving ambiguity
CFG Review

- $G = (N, \Sigma, P, S)$
- $\Rightarrow$ means derives
  + $\Rightarrow$ means derives in 1 or more steps
- CFG generates a string by applying productions until no non-terminals remain

Example: Nested parens

$N = \{ Q \}$
$\Sigma = \{ (, ) \}$
$P = Q \rightarrow ( Q )$
       $| \quad \varepsilon$
$S = Q$
Formal CFG Language Definition

Let $G = (N, \Sigma, P, S)$ be a CFG. Then

$L(G) = \left\{ w \left| S \Rightarrow^+ w \right. \right\}$ where

$S$ is the start nonterminal of $G$

$w$ is a sequence of terminals or $\varepsilon$
CFGs as Language Definition

• CFG productions define the syntax of a language

1. \textit{Prog} \rightarrow \textbf{begin} \textit{Stmts} \textbf{end}
2. \textit{Stmts} \rightarrow \textit{Stmts} \textbf{semicolon} \textit{Stmt}
3. \textbf{\mid} \textit{Stmt}
4. \textit{Stmt} \rightarrow \textit{id} \textbf{assign} \textit{Expr}
5. \textit{Expr} \rightarrow \textit{id}
6. \textbf{\mid} \textit{Expr} \textbf{plus} \textit{id}

• We call this notation “\textit{BNF}” or “\textit{enhanced BNF}”
• HTTP grammar using BNF:
  – http://www.w3.org/Protocols/rfc2616/rfc2616-sec2.html
List Grammars

• Useful to repeat a structure arbitrarily often

\[ Stmts \rightarrow Stmts \text{ semicolon } Stmt \mid Stmt \]
List Grammars

• Useful to repeat a structure arbitrarily often

\[ Stmts \rightarrow Stmt \text{ semicolon } Stmts \mid Stmt \]
List Grammars

• What if we allowed both “skews”?

\[
Stmts \rightarrow Stmts \text{ semicolon } Stmts \mid Stmt
\]
Derivation Order

- **Leftmost Derivation**: always expand the leftmost nonterminal
- **Rightmost Derivation**: always expand the rightmost nonterminal

1. \( Prog \rightarrow \text{begin} \ Stmts \ \text{end} \)
2. \( Stmts \rightarrow Stmts \ \text{semicolon} \ Stmt \)
3. \( \mid Stmt \)
4. \( Stmt \rightarrow id \ \text{assign} \ Expr \)
5. \( Expr \rightarrow id \)
6. \( \mid Expr \ \text{plus} \ id \)

Diagram:

- **Leftmost expands this nonterminal**
- **Rightmost expands this nonterminal**
Ambiguity

• Even with a fixed derivation order, it is possible to derive the same string in multiple ways

• For Grammar G and string w
  – G is ambiguous if
    • >1 leftmost derivation of w
    • >1 rightmost derivation of w
    • >1 parse tree for w
Example: Ambiguous Grammars

\[ Expr \rightarrow intlit \]
\[ \mid Expr \text{ minus } Expr \]
\[ \mid Expr \text{ times } Expr \]
\[ \mid \text{lparen } Expr \text{ rparen} \]

Derive the string 4 - 7 * 3

(assume tokenization)
Why is Ambiguity Bad?

- Eventually, we’ll be using CFGs as the basis for our parser
  - Parsing is much easier when there is no ambiguity in the grammar
  - The parse tree may mismatch user understanding!

```
Expr minus Expr
  intlit 4
    Expr times Expr
      intlit 7
        Expr minus Expr
          intlit 4
            intlit 3
      intlit 3
```

```
Expr times Expr
  intlit 4
    Expr minus Expr
      intlit 3
```

Resolving Grammar Ambiguity: Precedence

- Intuitive problem
  - “Context-freeness”
  - Nonterminals are the same for both operators
- To fix precedence
  - 1 nonterminal per precedence level
  - Parse lowest level first

\[
Expr \rightarrow \text{intlit} \\
| \;
Expr \; \text{minus} \; Expr \\
| \;
Expr \; \text{times} \; Expr \\
| \;
lparen \; Expr \; rparen
\]
Resolving Grammar Ambiguity: Precedence

1. lowest precedence level first
2. 1 nonterm per precedence level

Derive the string 4 - 7 * 3

```
Expr → intlit
| Expr minus Expr
| Expr times Expr
| lparen Expr rparen
```

```
Expr → Expr minus Expr
| Term
Term → Term times Term
| Factor
Factor → intlit
| lparen Expr rparen
```

```
Expr
  minus
  Expr
  Term
  Factor
  intlit 4
```
```
Expr
  Term
  times
  Term
  Factor
  intlit 7
```
```
Expr
  Term
  intlit 3
```

Diagram:
- Expr
  - minus
  - Expr
  - Term
  - Factor
  - intlit 4
- Expr
  - Term
  - times
  - Term
  - Factor
  - intlit 7
- Expr
  - Term
  - intlit 3
Resolving Grammar Ambiguity: Precedence

Fixed Grammar

Expr → expr minus expr
| Term

Term → Term times Term
| Factor

Factor → intlit
| lparen Expr rparen

Derive the string 4 - 7 * 3
Let’s try to re-build the wrong parse tree

We’ll never be able to derive minus without parens
Did we fix all ambiguity?

Fixed Grammar

\[\begin{align*}
\text{Expr} & \rightarrow \text{Expr} \; \text{minus} \; \text{Expr} \\
& \quad \mid \text{Term} \\
\text{Term} & \rightarrow \text{Term} \; \text{times} \; \text{Term} \\
& \quad \mid \text{Factor} \\
\text{Factor} & \rightarrow \text{intlit} \\
& \quad \mid \text{lparen} \; \text{Expr} \; \text{rparen}
\end{align*}\]

Derive the string 4 - 7 - 3

These subtrees could have been swapped!
Where we are so far

- **Precedence**
  - We want correct behavior on $4 - 7 \times 9$
  - A new nonterminal for each precedence level

- **Associativity**
  - We want correct behavior on $4 - 7 - 9$
  - Minus should be *left associative*: $a - b - c = (a - b) - c$
  - Problem: the *recursion* in a rule like
    $$Expr \rightarrow Expr \text{ minus } Expr$$
Definition: Recursion in Grammars

• A grammar is *recursive* in (nonterminal) $X$ if
  $$X \Rightarrow \alpha X \gamma$$
  for non-empty strings of symbols $\alpha$ and $\gamma$

• A grammar is *left-recursive* in $X$ if
  $$X \Rightarrow X \gamma$$
  for non-empty string of symbols $\gamma$

• A grammar is *right-recursive* in $X$ if
  $$X \Rightarrow \alpha X$$
  for non-empty string of symbols $\alpha$
Resolving Grammar Ambiguity: Associativity

- We’ll recognize left-associative operators with left-associative productions
- We’ll recognize right-associative operators with right-associative productions

\[
\begin{align*}
\text{Expr} &\rightarrow \text{Expr} \ \text{minus} \ \text{Expr} \\
&\quad | \ \text{Term} \\
\text{Term} &\rightarrow \text{Term} \ \text{times} \ \text{Term} \\
&\quad | \ \text{Factor} \\
\text{Factor} &\rightarrow \text{intlit} \ | \ \text{lparen} \ \text{Expr} \ \text{rparen}
\end{align*}
\]

Example: 4 – 7 – 9
Resolving Grammar Ambiguity: Associativity

\[ \text{Expr} \rightarrow \text{Expr} \text{ minus Term} \]
\[ \quad \mid \text{Term} \]
\[ \text{Term} \rightarrow \text{Term} \text{ times Factor} \]
\[ \quad \mid \text{Factor} \]
\[ \text{Factor} \rightarrow \text{intlit} \mid \text{lparen Expr rparen} \]

Example: \( 4 - 7 - 9 \)

Let’s try to re-build the wrong parse tree again

We’ll never be able to derive \texttt{minus} without parens
Example

• Language of Boolean expressions
  – bexp → TRUE
  – bexp → FALSE
  – bexp → bexp OR bexp
  – bexp → bexp AND bexp
  – bexp → NOT bexp
  – bexp → LPAREN bexp RPAREN

• Add nonterminals so that OR has lowest precedence, then AND, then NOT. Then change the grammar to reflect the fact that both AND and OR are left associative.

• Draw a parse tree for the expression:
  – true AND NOT true.
Another ambiguous example

Stmt →
    if Cond then Stmt |
    if Cond then Stmt else Stmt | ...

Consider this word in this grammar:
    if a then if b then s else s2
How would you derive it?
To understand how a parser works, we start by understanding **context-free grammars**, which are used to define the language recognized by the parser.

- (non)terminal symbol
- grammar rule (or production)
- derivation (leftmost derivation, rightmost derivation)
- parse (or derivation) tree
- the language defined by a grammar
- ambiguous grammar