Defining syntax using CFGs
Roadmap

Last time
  – Defined context-free grammar

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

• G = (N, ΣP, S)
• ⇒+ means derives 
derives in 1 or more 
steps
• CFG generates a 
string by applying 
productions until no 
non-terminals remain

Example: Nested parens
N = { Q }
Σ = { ( , ) }
P = Q → ( Q )
      ε
S = Q
Formal CFG Language Definition

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

$L(G) = wS \Rightarrow + \vdash w$ 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. \( \text{Prog} \rightarrow \text{begin Stmts end} \)
2. \( \text{Stmts} \rightarrow \text{Stmts semicolon Stmt} \)
3. \( \mid \text{Stmt} \)
4. \( \text{Stmt} \rightarrow \text{id assign Expr} \)
5. \( \text{Expr} \rightarrow \text{id} \)
6. \( \mid \text{Expr plus id} \)

We call this notation “BNF” or “extended BNF”

HTTP grammar using BNF:

– http://www.w3.org/Protocols/rfc2616/rfc2616-sec2.html
List Grammars

- Useful to repeat a structure arbitrarily often

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

• Useful to repeat a structure arbitrarily often

\[ \text{Stmts} \rightarrow \text{Stmt} \text{ semicolon } \text{Stmts} | \text{Stmt} \]

List skews right
List Grammars

• What if we allowed both “skews”?

\[ \text{Stmts} \rightarrow \text{Stmts \ semicolon \ Stmts} \mid \text{Stmt} \]
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. $| \ Stmt$
4. $Stmt \rightarrow \text{id assign } \text{Expr}$
5. $Expr \rightarrow \text{id}$
6. $| \ Expr \text{ plus } \text{id}$
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

\[ Expr \rightarrow \text{intlit} \]
\[ \quad | \; Expr \; \text{minus} \; Expr \]
\[ \quad | \; Expr \; \text{times} \; Expr \]
\[ \quad | \; \text{lparen} \; Expr \; \text{rparen} \]

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!

```
4 - 7 * 3
```

![Operator precedence diagram]
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

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

lowest precedence level first
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
Resolving Grammar Ambiguity: Precedence

Fixed Grammar

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

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

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

Derive the string 4 - 7 - 3

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

Precedence
  – We want correct behavior on $4 - 7 * 9$
  – 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

    $Expr \to Expr \textbf{minus} Expr$
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
  
  $X \Rightarrow + \gamma X \gamma$ for non-empty string of symbols $\gamma$

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

Recognize left-assoc operators with left-associative productions
Recognize right-assoc operators with right-associative productions

Example:

\[ 4 - 7 - 9 \]

```
Expr → Expr minus Expr
| Term

Term → Term times Term
| Factor

Factor → intlit | lparen Expr rparen
```

Example tree:

```
E       -
|--       |
T       T
|--       |
F       F
|--       |
intlit  9
```

```
E       -
|--       |
T       T
|--       |
F       F
|--       |
intlit  7
```

```
E       -
|--       |
intlit  9
```

```
E       -
|--       |
4       |
```

```
E       -
|--       |
T       T
|--       |
F       F
|--       |
intlit  7
```

```
E       -
|--       |
4       |
```

```
E       -
|--       |
T       T
|--       |
F       F
|--       |
intlit  7
```

```
E       -
|--       |
intlit  9
```

```
E       -
|--       |
4       |
```

```
E       -
|--       |
T       T
|--       |
F       F
|--       |
intlit  7
```

```
E       -
|--       |
intlit  9
```

```
E       -
|--       |
4       |
```

```
E       -
|--       |
T       T
|--       |
F       F
|--       |
intlit  7
```

```
E       -
|--       |
intlit  9
```

```
E       -
|--       |
4       |
```
Resolving Grammar Ambiguity: Associativity

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

Example: 4 − 7 − 9

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

We’ll never be able to derive \text{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?
Summary

To understand how a parser works, we start by understanding **context-free grammars**, which are used to define the language recognized by the parser.

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