Classes of Languages

1. Regular languages (finite automata)
2. Context-free languages (push-down automata)
3. Context-sensitive languages (linear bounded automata)
4. Recursively-enumerable languages (Turing machine)

Grammar for Java, a big language

• The Java™ Language Specification, 2nd Edition
  » Entire document
    • 500+ pages
  » Grammar productions with explanatory text
  » Chapter 18, Syntax
    • 8 pages of grammar productions, presented in "BNF-style"

Parsing

• The syntax of most programming languages can be specified by a context-free grammar (CFG)

• Parsing
  » Given a grammar $G$ and a sentence $w$ in $L(G)$, traverse the derivation (parse tree) for $w$ in some standard order and do something useful at each node
  » The tree might not be produced explicitly, but the control flow of a parser corresponds to a traversal

Parse Tree Example

```
G
```

```
program
    | statement
    | program statement
assignStmt
    | expr
    | id = expr
ifStmt
    | if ( expr ) stmt
expr
    | id
    | int
    | expr + expr
id
    | a | b | c | i | j | k | n | x | y | z
int
    | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
```

```
program
    | assignStmt
    | statement
    | program statement
assignStmt
    | expr
    | id = expr
ifStmt
    | if ( expr ) stmt
expr
    | id
    | int
    | expr + expr
id
    | a | b | c | i | j | k | n | x | y | z
int
    | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
```

```
w → a = 1 ; if ( a + 1 ) b = 2 ;
```
“Standard Order”

- For practical reasons we want the parser to be deterministic (no backtracking), and we want to examine the source program from left to right.

“Something Useful”

- At each point (node) in the traversal, perform some semantic action
  - Construct nodes of full parse tree
  - Construct abstract syntax tree
  - Construct linear, lower-level representation
  - Generate target code on the fly → 1-pass compiler

Common Orderings

- Top-down
  - Start with the root
  - Traverse the parse tree depth-first, left-to-right (leftmost derivation)
  - LL(k)
- Bottom-up
  - Start at leaves and build up to the root
    - Effectively a rightmost derivation in reverse
  - LR(k)

Context-Free Grammars

- Formally, a grammar $G$ is a tuple $<N, \Sigma, P, S>$ where
  - $N$ a finite set of non-terminal symbols
  - $\Sigma$ a finite set of terminal symbols
  - $P$ a finite set of productions
    - A subset of $N \times (N \cup \Sigma)^*$
  - $S$ the start symbol, a distinguished element of $N$

Standard Notations

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$a, b, c$</td>
<td>elements of $\Sigma$</td>
</tr>
<tr>
<td>$w, x, y, z$</td>
<td>elements of $\Sigma^*$</td>
</tr>
<tr>
<td>$A, B, C$</td>
<td>elements of $N$</td>
</tr>
<tr>
<td>$X, Y, Z$</td>
<td>elements of $N \cup \Sigma$</td>
</tr>
<tr>
<td>$\alpha, \beta, \gamma$</td>
<td>elements of $(N \cup \Sigma)^*$</td>
</tr>
<tr>
<td>$A \rightarrow \alpha$ (or $A ::= \alpha$) if $&lt;A, \alpha&gt;$ in $P$</td>
<td></td>
</tr>
</tbody>
</table>

Derivation Relations

- $\alpha A \gamma \Rightarrow \alpha \beta \gamma$ iff $A \rightarrow \beta$ in $P$
- $A \Rightarrow^* w$ if there is a chain of productions starting with $A$ that generates $w$
  - "Non-terminal $A$ derives the string of terminals $w"
Derivation Relations

- \( w \ A \gamma \Rightarrow_{lm} w \beta \gamma \) iff \( A \rightarrow \beta \) in \( P \)
  - derive by always picking the first non-terminal from the left
- \( \alpha A w \Rightarrow_{rm} \alpha \beta w \) iff \( A \rightarrow \beta \) in \( P \)
  - derive by always picking the first non-terminal from the right
- We will only be interested in leftmost and rightmost derivations – not random orderings

Languages

- For \( A \) in \( N \), \( L(A) = \{ \ w \mid A \Rightarrow^* w \} \)
  - for any non-terminal \( A \) defined for a grammar, the language generated by \( A \) is the set of strings \( w \) that can be derived from \( A \) using the productions
- If \( S \) is the start symbol of grammar \( G \), define \( L(G) = L(S) \)
  - The language derived by \( G \) is the language derived by the start symbol \( S \)

Reduced Grammars

- Grammar \( G \) is reduced iff for every production \( A \rightarrow \alpha \) in \( G \) there is a derivation \( S \Rightarrow^* x A z \Rightarrow^* x \alpha z \Rightarrow^* xyz \)

Ambiguity

- Grammar \( G \) is unambiguous iff every \( w \) in \( L(G) \) has a unique leftmost (or rightmost) derivation
  - Fact: unique leftmost or unique rightmost implies the other
- A grammar without this property is ambiguous
  - Note that other grammars that generate the same language may be unambiguous

Ambiguous Grammar for Expressions

\[
\begin{align*}
\text{expr} & \rightarrow \text{expr} + \text{expr} \mid \text{expr} - \text{expr} \\
& \mid \text{expr} \times \text{expr} \mid \text{expr} / \text{expr} \\
\text{int} & \rightarrow 0 \mid 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9
\end{align*}
\]

- Show that this is ambiguous
  - How? Show two different leftmost or rightmost derivations for the same string
  - Equivalently: show two different parse trees for the same string

Example Derivation

Give a leftmost derivation of \( 2+3\times4 \) and show the parse tree
Give a different leftmost derivation of $2 + 3 \times 4$ and show the parse tree

Another Derivation

$\text{expr} \rightarrow \text{expr} + \text{expr} | \text{expr} - \text{expr} | \text{expr} \times \text{expr} | \text{expr} \div \text{expr} | \text{int} \rightarrow 0|1|2|3|4|5|6|7|8|9$

Another Example

Give two different derivations of $5 + 6 + 7$

What’s going on here?

- The grammar has no notion of precedence or associativity
- Solution
  » Create a non-terminal for each level of precedence
  » Isolate the corresponding part of the grammar
  » Force the parser to recognize higher precedence subexpressions first

Classic Expression Grammar

$\text{expr} \rightarrow \text{expr} + \text{term} | \text{expr} - \text{term} | \text{term}$
$\text{term} \rightarrow \text{term} \times \text{factor} | \text{term} / \text{factor} | \text{factor}$
$\text{factor} \rightarrow \text{int} | ( \text{expr} )$
$\text{int} \rightarrow 0|1|2|3|4|5|6|7$

Derive $2 + 3 \times 4$

$\text{expr} \rightarrow \text{expr} + \text{term} | \text{expr} - \text{term} | \text{term} \times \text{factor} | \text{term} / \text{factor} | \text{factor}$
$\text{factor} \rightarrow \text{int} | ( \text{expr} )$
$\text{int} \rightarrow 0|1|2|3|4|5|6|7$
Another Classic Example

- Grammar for conditional statements
  
  \[ ifStmt \rightarrow \text{if ( cond ) stmt} \]
  
  \[ \text{if ( cond ) stmt else stmt} \]

  » Exercise: show that this is ambiguous
  
  - How?

Another Derivation

\[ ifStmt \rightarrow \text{if ( cond ) stmt} \]

\[ \text{if ( cond ) stmt else stmt} \]

Solving if Ambiguity

- Fix the grammar to separate if statements with else clause and if statements with no else
  
  » Done in Java reference grammar
  
  » Adds lots of non-terminals

- Use some ad-hoc rule in parser
  
  » “else matches closest unpaired if”