CS480
Translators

Intro to Parsing
Chap. 4
Things to Address...

• Test on Friday
• Demo Milestone 2 this week.
• Milestone 3 is posted.
Revisit Quiz #3

• Given $\Sigma = \{a,b\}$, provide regular expressions for languages below:
  – all strings beginning and ending in $a$
  – all strings of $a$’s and $b$’s of even length
  – all strings with an odd number of $a$’s
  – string of zero or more $a$’s followed by same number of $b$’s
What is the Parser?

Figure 4.1: Position of parser in compiler model
Types of Parsers

• Universal
• Top-Down
• Bottom-Up
Error Correction

- Panic Mode
- Phrase-Level Recovery
- Error Productions
- Global Recovery
Context Free Grammars

• Nonterminals, N
• Terminals, T
• Set of Productions, P
• Start Symbol, S

• Four-tuple (N, T, P, S)
Example

expression → expression + term
expression → expression − term
expression → term

term → term * factor
term → term / factor
term → factor

factor → ( expression )

factor → id

Figure 4.2: Grammar for simple arithmetic expressions
Context Free vs. Regular Languages

• \((a|b)^*abb\)
  
  \[ A \rightarrow aA \mid bA \mid abb \]

• \(L=\{a^n b^n \mid n \geq 1\}\)

Figure 4.6: DFA \(D\) accepting both \(a^i b^i\) and \(a^j b^j\).
Production/Derivation Notation

- ::= vs. ->
- ⇒
- ∗
- ⇀
- +
- ↝
- ⇒ lm
\[ E \rightarrow E + E \mid E \ast E \mid (E) \mid \text{id} \]

- Derivations

- \( E \Rightarrow (E) \)

- \( E \Rightarrow (E) \Rightarrow (\text{id}) \)

- \( E \Rightarrow (\text{id}) \) or \( E \Rightarrow (\text{id}) \)
\[ E \rightarrow E + E \mid E * E \mid - E \mid (E) \mid \text{id} \]

- \[ E \Rightarrow -E \Rightarrow -(E + E) \Rightarrow -(\text{id} + E) \Rightarrow -(\text{id} + \text{id}) \]
  \[ lm \quad lm \quad lm \quad lm \]

- \[ E \Rightarrow -E \Rightarrow -(E + E) \Rightarrow -(E + \text{id}) \Rightarrow -(\text{id} + \text{id}) \]
  \[ rm \quad rm \quad rm \quad rm \]

```
Figure 3: Syntax tree for the expression id + id
```

**Figure 3: Syntax tree for the expression id + id**
Ambiguity

Figure 4.5: Two parse trees for \texttt{id+id*id}
Eliminate Left Recursion

• Immediate Left Recursion
  \[ A \rightarrow A\alpha_1 | A\alpha_2 | ... | A\alpha_m | \beta_1 | \beta_2 | ... | \beta_n \]
• \[ A \rightarrow \beta_1A' | \beta_2A' | ... | \beta_nA' \]
• \[ A' \rightarrow \alpha_1A' | \alpha_2A' | ... | \alpha_mA' | \epsilon \]

• Example:
  \[ E \rightarrow E + E | E * E | (E) | \text{id} \]
  \[ E \rightarrow (E) E' | \text{id} E' \]
  \[ E' \rightarrow + E E' | * E E' | \epsilon \]
Eliminate Left Recursion cont.

- Example:
  \[ S \rightarrow A \text{a} \mid \text{b} \]
  \[ A \rightarrow A \text{c} \mid S \text{d} \mid \varepsilon \]
- Not immediate
  \[ S \Rightarrow A \text{a} \Rightarrow S \text{d a} \]
- Substitute all S productions in A
  \[ A \rightarrow A \text{c} \mid A \text{a d} \mid \text{b d} \mid \varepsilon \]
Eliminate Left Factoring

A -> \alpha \beta_1 | \alpha \beta_2

• A -> \alpha A'

• A' -> \beta_1 | \beta_2

• Example:

  stmt -> if expr then stmt else stmt | if expr then stmt

  stmt -> if expr then stmt E

  E -> else stmt | \epsilon
Criteria for Parsing

• Efficient – proportional to size
• Determine action by fixed # tokens
• Practical Considerations
  – 1 Lookahead
  – No backtracking
  – LL(1) grammar

• What is LL(k)?
LL Grammars

- Top-down Parsing
  - Recursive descent
    - General
    - Predictive
  - Table-driven
Top Down Parsing

```c
void A() {
1) Choose an A-production, A → X_1 X_2 ⋯ X_k;
2) for ( i = 1 to k ) {
3) if ( X_i is a nonterminal )
4) call procedure X_i();
5) else if ( X_i equals the current input symbol a )
6) advance the input to the next symbol;
7) else /* an error has occurred */;
}
}
```

- How does this change for the production below?

```
A-> ab | a
```