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

Any grammar real-world? NO

Learn about YACC, LR, LALR, SLL, etc.
Error Correction

• Panic Mode
• Phrase-Level Recovery
• Error Productions
• Global Recovery

Keep ignoring until I see something that fits.

You don't have to.

Can we guess what you needed there?

Add "j"

Embed error productions in grammar
Context Free Grammars

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

- Four-tuple (N, T, P, S)
Example

\[
\text{expression} \rightarrow \text{expression} + \text{term} \\
\text{expression} \rightarrow \text{expression} - \text{term} \\
\text{expression} \rightarrow \text{term} \\
\text{term} \rightarrow \text{term} \times \text{factor} \\
\text{term} \rightarrow \text{term} / \text{factor} \\
\text{term} \rightarrow \text{factor} \\
\text{factor} \rightarrow ( \text{expression} ) \\
\text{factor} \rightarrow \text{id}
\]

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

- \((a|b)^*abb\)
  \[A \to aA \mid bA \mid abb\]
- \(L = \{a^n b^n \mid n \geq 1\}\)

![Diagram of DFA D accepting both \(a^i b^i\) and \(a^j b^j\).]

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

- ::= vs. ->
- \Rightarrow
- * \Rightarrow
- \downarrow \Rightarrow
- \Rightarrow lm
\[ E \rightarrow E + E \mid E \ast E \mid (E) \mid \text{id} \]

- Derivations
  - \[ E \Rightarrow (E) \]
  - \[ E \Rightarrow (E) \Rightarrow (\text{id}) \]
  - \[ E \Rightarrow (\text{id}) \text{ or } E \Rightarrow (\text{id}) \]

(\text{id})

\[ \text{start} \]

ending strips

\[ \text{or} \]

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

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

Diagram:

```
   E
  / \  \
-   E
  /   \
 E   E
/     \
id    id
```
Ambiguity

Figure 4.5: Two parse trees for $\text{id}+\text{id}\times\text{id}$ from the same type of derivation

have to have rules
Eliminate Left Recursion

- Immediate Left Recursion
  \[ A \rightarrow A\alpha_1 | A\alpha_2 | \ldots | A\alpha_m | \beta_1 | \beta_2 | \ldots | \beta_n \]
- \[ A \rightarrow \beta_1 A' | \beta_2 A' | \ldots | \beta_n A' \]
- \[ A' \rightarrow \alpha_1 A' | \alpha_2 A' | \ldots | \alpha_m A' | \varepsilon \]

- Example:
  \[ E \rightarrow E + E | E * E | (E) | id \]
  \[ E' \rightarrow (E) E' | id E' \]
  \[ E' \rightarrow + E E' | * E E' | \varepsilon \]
Eliminate Left Recursion cont.

- Example:
  \[ S \rightarrow \begin{array}{c} A \ a \mid b \\ A \rightarrow \begin{array}{c} A \ c \mid S \ d \mid \varepsilon \end{array} \end{array} \]
  
- Not immediate
  \[ S \Rightarrow A \ a \Rightarrow S \ d \ a \]

- Substitute all S productions in A
  \[ A \rightarrow \begin{array}{c} A \ c \mid A \ a \ d \mid b \ d \mid \varepsilon \end{array} \]
Eliminate Left Factoring

A → αβ₁ | αβ₂
• A → αA' 
• A' → β₁ | β₂

Example:

stmt → if expr then stmt else stmt | if expr then stmt

stmt → if expr then stmt E
E → else stmt | ε
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, \( A \rightarrow X_1X_2\cdots X_k \);
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 */;
8) }
9}
```

- How does this change for the production below?
  \[ A \rightarrow ab | a \]