CS480
Translators
Intro to Parsing
Chap. 4
Crazy Semantics

https://www.destroyallsoftware.com/talks/wat

....Thank you Kevin Strasser
Things to Address...

• Convert the NFA to DFA

• Write a finite state automata that will recognize any string consisting of a and b characters where the number of a's is even (or zero) and the number of b's is odd.

• Test on Friday

• Milestone 2 and 3.
Minimize a DFA

• First, group accepting states together and group other states together.

initially, let Π_{new} = Π;
for ( each group G of Π ) {
    partition G into subgroups such that two states s and t are in the same subgroup if and only if for all input symbols a, states s and t have transitions on a to states in the same group of Π;
    /* at worst, a state will be in a subgroup by itself */
    replace G in Π_{new} by the set of all subgroups formed;
}
Minimize a DFA for Lexical Analysis

Figure 3.54: Transition graph for DFA handling the patterns $a$, $abb$, and $a^*b^+$
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. →

• ⇒

• *

• ⊢

• +

• ⊢

• ⇒

• l m
\[ E \rightarrow E + E \mid E \ast E \mid (E) \mid \text{id} \]

- Derivations
  - \( E \Rightarrow (E) \)
  - \( E \Rightarrow (E) \Rightarrow (\text{id}) \)
  - \( E \Rightarrow (\text{id}) \) or \( E \Rightarrow (\text{id}) \)
$E \rightarrow E + E \mid E \ast E \mid -E \mid (E) \mid \text{id}$

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

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

Figure 4.5: Two parse trees for id+id*id
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:
  \[ E \rightarrow E + E | E \ast E | (E) | \text{id} \]
  \[ E \rightarrow (E) E' | \text{id} E' \]
  \[ E' \rightarrow E E' | \ast E E' | \varepsilon \]
Eliminate Left Recursion cont.

• Example:
  \[ S \rightarrow A \ a \ | \ b \]
  \[ A \rightarrow A \ c \ | \ S \ d \ | \ \varepsilon \]

• Not immediate
  \[ S \Rightarrow A \ a \Rightarrow S \ d \ a \]

• Substitute all S productions in A
  \[ A \rightarrow A \ c \ | \ A \ a \ d \ | \ b \ d \ | \ \varepsilon \]
Eliminate Left Factoring

\[ A \rightarrow \alpha \beta_1 \mid \alpha \beta_2 \]

- \( A \rightarrow \alpha A' \)
- \( A' \rightarrow \beta_1 \mid \beta_2 \)

- Example:
  
  \[ stmt \rightarrow \text{if expr then stmt else stmt} \mid \text{if expr then stmt} \]

  \[ stmt \rightarrow \text{if expr then stmt E} \]

  \[ E \rightarrow \text{else stmt} \mid \epsilon \]
void A() {

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

Figure 4.13: A typical procedure for a nonterminal in a top-down parser