CISC 4090
Theory of Computation

Context-Free Languages and
Push Down Automata

Professor Daniel Leeds
dleeds@fordham.edu
JMH 332

Languages: Regular and Beyond

Regular: $(a \cup b) \cdot c^* \cdot b \cdot (d \cup e \cup a)$

Not-regular: $c^nbd^n$

Context Free Grammars:
• Human language
• Parsing of computer language

An example Context-Free Grammar

Grammar G1

$A \rightarrow 0A1$
$A \rightarrow B$
$B \rightarrow #$

Variables: A, B; Terminals: 0, 1, #

One start variable

Substitution rules/productions
• Variable -> Variables, Terminals

Example strings generated:
#, 0#1, 00#11, 000#111, ...

$L(G1) = \{0^n#1^n | n \geq 0\}$

Example English Grammar

Sentence -> NounPhrase VerbPhrase
NounPhrase -> Article NounSub
NounSub -> Noun | Adjective NounSub
VerbPhrase -> Verb | Verb NounPhrase
Noun -> Girl | Boy | Duck | Ball
Article -> The | A
Verb -> Throws | Sings
Formal CFG Definition

A CFG is a 4-tuple \((V, \Sigma, R, S)\)

- \(V\) is finite set of variables
- \(\Sigma\) finite set of terminals
- \(R\) finite set of rules
- \(S \in V\) start variable

Another example

\[ G_3 = (\{S\}, \{a, b\}, R, S) \]

\[ R: \quad S \rightarrow aSb \mid SS \mid \epsilon \]

Example strings generated:

\(\epsilon, ab, abab, aabb, aaabbbab, ababababab, abaaaabbb, \ldots\)

\(L(G_1) = \{a's \& b's; each a is followed by a matching b, every b matches exactly one corresponding preceding a\} \)

(like parenthesis matching)

Designing CFGs

Creativity required

- If CFL is union of simpler CFL, design grammar for simpler ones \((G_1, G_2, G_3)\), then combine: \(S \rightarrow G_1 \mid G_2 \mid G_3\)

- If language is regular, can make CFG mimic DFA
  
  Match each state with a single corresponding variable
  
  \[ Q=\{q_0, \ldots, q_n\} \quad V=\{R_0, \ldots, R_n\} \]

  Replace transition function with Production rule
  
  \[ \delta(q_i, a) = q_j \quad R_i \rightarrow aR_j \]

  Accept state \(q_k\) : transition to \(\epsilon\)

  \[ R_k \rightarrow \epsilon \]

Ambiguity – examples

A grammar may generate a string in multiple ways

Math example:

\[ \text{Expr} \rightarrow \text{Expr} + \text{Expr} \mid \text{Expr} \times \text{Expr} \mid \text{Expr} \mid a \]

English example:

\textit{the girl touches the boy with the flower}
Ambiguity – definitions

A grammar generates a string ambiguously if there are two or more different parse trees.

Definitions:

- **Leftmost derivation**: at each step the leftmost remaining variable is replaced.
- **w** is derived **ambiguously** in CFG G if there exist more than one leftmost derivations.

a+axa+a leftmost derivations

<table>
<thead>
<tr>
<th>Derivation 1</th>
<th>Derivation 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Expr</td>
<td>Expr</td>
</tr>
<tr>
<td>Expr x Expr</td>
<td>Expr + Expr</td>
</tr>
<tr>
<td>Expr + Expr x Expr</td>
<td>Expr x Expr + Expr</td>
</tr>
<tr>
<td>a + Expr x Expr</td>
<td>Expr + Expr x Expr + Expr</td>
</tr>
<tr>
<td>a + a x Expr</td>
<td>a + Expr x Expr + Expr</td>
</tr>
<tr>
<td>a + a x a + Expr</td>
<td>a + a x a + Expr</td>
</tr>
<tr>
<td>a + a x a + a</td>
<td>a + a x a + a</td>
</tr>
</tbody>
</table>

Chomsky Normal Form

CFG is in Chomsky normal form if every rule takes form:

- A → BC
- A → a
- B and C may not be the start variables
- The start variable may transition to ε

Any CFL can be generated by CFG in Chomsky Normal Form.

Converting to Chomsky Normal Form

- S₀ → S where S was original start variable
- Remove A → ε
- For each multiple-occurrence of A, add new rules with A deleted: R → u₁A₁u₂ → u₁u₁u₂
- Shortcut all unit rules: Given A → B and B → u, add A → u
- Replace rules A → u₁₁u₂_1u₃_1...uₖ with: A → u₁₁A₁, A₁ → u₂₁A₂, A₂ → u₃₁A₃, ..., Aₖ₋₁ → uₖ₋₁uₖ
Conversion practice

Non-normal form:
- $S \rightarrow aXbX$
- $X \rightarrow aY|bY|\varepsilon$
- $Y \rightarrow X|c$

Conversion practice, answer part 1

Non-normal form:
- $S \rightarrow aXbX$
- $X \rightarrow aY|bY|\varepsilon$
- $Y \rightarrow X|c$

Step 1: $S_0 \rightarrow S,
- then place $\varepsilon$ for $X$
  - $S \rightarrow aXbX$
  - $S \rightarrow S$
  - $X \rightarrow aY|bY|\varepsilon$
  - $Y \rightarrow X|c$

Conversion practice, answer part 2

Step 1: $S_0 \rightarrow S,
- then place $\varepsilon$ for $X$
  - $S_0 \rightarrow S$
  - $S \rightarrow aXbX|abX|aXb|ab$
  - $S \rightarrow aXbX|abX|aXb|ab$
  - $X \rightarrow aY|bY|a|b$
  - $X \rightarrow aY|bY|a|b$

Step 2: place $\varepsilon$ for $Y$
  - $S_0 \rightarrow S$
  - $S \rightarrow aXbX|abX|aXb|ab$

Conversion practice, answer part 3

Step 2: place $\varepsilon$ for $Y$
  - $S_0 \rightarrow S$
  - $S \rightarrow aXbX|abX|aXb|ab$

Step 3: replace $S$ and $X$
  - $S \rightarrow aXbX|abX|aXb|ab$
  - $X \rightarrow aY|bY|a|b$
  - $X \rightarrow aY|bY|a|b$
  - $Y \rightarrow X|c$
  - $Y \rightarrow X|c$
  - $Y \rightarrow X|c$
  - $Y \rightarrow aY|bY|a|b|c$
Conversion practice, answer part 3

Step 4: replace terminal-variable combos with variable combos

\[ S_0 \rightarrow U_1 X U_2 X | U_1 U_2 X | U_1 X U_2 | U_1 U_2 \]
\[ S \rightarrow U_1 X U_2 X | U_1 U_2 X | U_1 X U_2 | U_1 U_2 \]
\[ X \rightarrow U_1 Y | U_2 Y | a | b \]
\[ Y \rightarrow U_1 Y | U_2 Y | a | b | c \]

Step 4: replace multi-variables with pairs

\[ S_0 \rightarrow U_1 C | U_1 D | U_1 E | U_1 U_2 \]
\[ S \rightarrow U_1 C | U_1 D | U_1 E | U_1 U_2 \]
\[ C \rightarrow X D \quad D \rightarrow U_2 X \quad E \rightarrow X U_2 \]
\[ X \rightarrow U_1 Y | U_2 Y | a | b \]
\[ U_1 \rightarrow a \quad U_2 \rightarrow b \quad Y \rightarrow U_1 Y | U_2 Y | a | b | c \]
Definition of PDA

A PDA is a 6-tuple \((Q, \Sigma, \Gamma, \delta, q_0, F)\) where \(Q, \Sigma, \Gamma,\) and \(F\) are finite sets
- \(Q\) is sets of states
- \(\Sigma\) is the input alphabet
- \(\Gamma\) is the stack alphabet
- \(\delta: Q \times \Sigma \times \Gamma \epsilon \rightarrow P(Q \times \Gamma \epsilon)\) is transition function
- \(q_0 \in Q\) is start state
- \(F \subseteq Q\) is set of accept states

PDA computation

\(M\) must start in \(q_0\) with empty stack
\(M\) must move according to transition function
To accept string, \(M\) must be at accept state at end of input
Start stack with \$. If you see \$ at top of stack, it is empty

Understanding transition \(\delta\)

\(a, b \rightarrow c\) means:
- when you read \(a\) from tape and \(b\) is on top of stack
- replace \(b\) with \(c\) on top of stack

\(a, b,\) or \(c\) can be \(\epsilon\)
- If \(a\) is \(\epsilon\) then change stack without reading a symbol
- If \(b\) is \(\epsilon\) then push new symbol \(c\) without popping \(b\)
- If \(c\) is \(\epsilon\) then no new symbol pushed, only pop \(b\)

PDA to accept \(0^n1^n\)

\(M1\) is \((Q, \Sigma, \Gamma, \delta, q_0, F)\)
- \(Q = \{q_1, q_2, q_3, q_4\}\)
- \(\Sigma = \{0,1\}\)
- \(\Gamma = \{0,\$\}\)
- \(F = \{q_1, q_4\}\)

\(0, \epsilon \rightarrow 0\)
\(1, 0 \rightarrow \epsilon\)
\(\epsilon, \epsilon \rightarrow \$
\(1, 0 \rightarrow \epsilon\)
\(\epsilon, \$ \rightarrow \epsilon\)
Theorem: A language is context free if and only if some PDA recognizes it

Let's prove: If a language $L$ is CFL, some PDA recognizes it

Idea: Show how CFG can define a PDA
• Stack has set of terminals/variables to compare with input
• Place proper terminal/variable pattern onto stack based on rules
• Non-determinism: Clone your machine, following different branches of rules

CFG -> PDA
• If top of stack is variable, sub one right-hand rule for the variable
• If top of stack is terminal, keep going iff terminal matches input
• If top of stack is $\$, accept!
Example 2.25 in textbook

Regular languages vs. CFLs

- CFGs define CFLs
- PDAs recognize CFLs and Regular languages
- FSAs recognize Regular languages, but **not** CFLs
- CFLs and Regular languages not equivalent

Non Context Free Languages

Languages recognized by PDAs
- \( L = \{ww^2\} \)
- \( L = \{a^n b^n | n \geq 0\} \)

Languages **not** recognized by PDAs
- \( L = \{ww\} \)
- \( L = \{a^n b^n c^n | n \geq 0\} \)

Proving non context free – NEW pumping lemma!

Every string in CFL \( A \) with length greater than or equal to the pumping length \( p \) can be “pumped”

Every string \( w \in A (|w| \geq p) \) can be written as \( uvxyz \) where

1. For each \( i \geq 0 \), \( uv^i xy^i z \in A \)
2. \(|vy| > 0 \)
3. \(|vxy| \leq p \)
Regular language PUMPING: Proof idea

If \(|s| < p\), trivially true

If \(|s| \geq p\), consider the states the FSA goes through
• Since there are only \(p\) states, \(|s| > p\), one state must be repeated
  • Pigeonhole principle: There must be a cycle

CFL pumping: Proof idea

Pigeonhole idea: Given a long enough string, some variable will need to be repeated

Example Grammar: \(T \rightarrow uRz\)
  \( \rightarrow x \mid vRy\)

Prove \(F = \{ww \mid w = (0 \cup 1)^*\} \) not CFL

Try a sample string \(s = \{0^p10^p1\} \quad |s| > p\)
• Can we define \(uvxyz = s\) so \(uvxy/yz \in F\) ?
  • Yes: \(u = 0^{p-1}, v = 0, x = 1, y = 0, z = 0^{p-1}1\)

Try another sample string \(s = \{0^p10^p10^p1\}\)
• Can we define \(uvxyz = s\) so \(uvxy/yz \in F\) ?
  • No:
    • If \(vxy\) is in first \(w\), pumping will make increase 1's and/or 0's in first \(w\) but not in second
    • If \(vxy\) straddles the middle, \(vxy\) will either increase 1's for first \(w\) and 0's for second \(w\), or will break the 0^n1^n pattern

Prove \(B = \{a^nb^n c^n \mid n \geq 0\} \) not CFL