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:
• Captured by Regular Operations  \((a \cup b) \cdot c^* \cdot (d \cup e)\)
• Recognized by Finite State Machines

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

An example Context-Free Grammar

Grammar G1

<table>
<thead>
<tr>
<th>Rule</th>
<th>String Generated</th>
</tr>
</thead>
<tbody>
<tr>
<td>A → 0A1</td>
<td>#, 0#1, 00#11, 000#111, ...</td>
</tr>
<tr>
<td>A → B</td>
<td></td>
</tr>
<tr>
<td>B → #</td>
<td></td>
</tr>
</tbody>
</table>

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

Variables: A, B; Terminals: 0, 1, #
One start variable: A
Substitution rules/productions
• Variable → Variables, Terminals

Example English Grammar

Example 1:

- S → NP VP
  → A NS V
  → A N V
  → The Boy Sings

Example 2:

- S → NP VP
  → A NS V
  → A N V
  → A Duck Throws
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

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

\(R: \quad S \rightarrow aSb \mid SS \mid \varepsilon\)

Example strings generated:
- \(\varepsilon\), \(ab\), \(abab\), \(aabb\), \(aaabbbab\), \(ababababab\), …

\(L(G_3) = \{\text{a's & b's; each a is followed by a matching b, every b matches exactly one corresponding preceding a}\}\)

(like parenthesis matching)

Example rule expansion:
- \(S \rightarrow aSb\)
- \(S \rightarrow SS\)
- \(S \rightarrow \varepsilon\)

\(G_3\) example rule expansion:
- \(S \rightarrow aSb\)
- \(S \rightarrow SS\)
- \(S \rightarrow \varepsilon\)

Example strings generated:
- \(\varepsilon, ab, abab, aabb, aaabbbab,\)
- \(abababab, abaaabbb, …\)

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

Another example

\(G_4 = (\{A, B, C\}, \{a, b, c\}, R, A)\)

\(R: \quad A \rightarrow aA \mid BC \mid \varepsilon\)
- \(B \rightarrow Bb \mid C\)
- \(C \rightarrow c \mid \varepsilon\)

Example strings generated:
- \(\varepsilon, aaa, cbbc, aacc\)

\(L(G_4) = \{\text{Hard to describe...}\}\)
Designing CFGs
Creativity required

- If language is regular, can make CFG mimic DFA
  Match each state with a single corresponding variable
  \[ Q=\{q_0, \ldots, q_n\} \quad V=\{R_0, \ldots, R_m\} \]
  Start state \( q_0 \) corresponds to state variable \( S \rightarrow R_0 \)
  Replace transition function with Production rule
  \[ \delta(q_i, a) = q_j \quad R_i \rightarrow aR_j \]
  Accept state \( q_k \): transition to \( \epsilon \)
  \[ R_k \rightarrow \epsilon \]

Chomsky Normal Form

CFG is in Chomsky normal form if every rule takes form:
- \( A \rightarrow BC \)
- \( A \rightarrow a \)
  - B and C may not be the start variables
  - The start variable may transition to \( \epsilon \)

Any CFL can be generated by CFG in Chomsky Normal Form
Conversion practice

Non-normal form:
\[ S \rightarrow aSa|bX \]
\[ X \rightarrow Ycc|\varepsilon \]
\[ Y \rightarrow d|c \]

Step 1: \( S_0 \rightarrow S \), \( S \rightarrow aSa|bX \)
\[ S \rightarrow aSa|bX \]
\[ X \rightarrow Ycc|\varepsilon \]
\[ Y \rightarrow d|c \]

Step 2: Remove \( \varepsilon \), \( S \rightarrow aSa|bX \)
\[ S \rightarrow aSa|bX \]
\[ X \rightarrow Ycc \]
\[ Y \rightarrow d|c \]

Step 3: Use unit rules, \( S \rightarrow aSa|bX \)
\[ S \rightarrow aSa|bX \]
\[ X \rightarrow Ycc \]
\[ Y \rightarrow d|c \]

Step 4: Replace terminals, \( S \rightarrow AN|BX \)
\[ S \rightarrow AN|BX \]
\[ X \rightarrow YM \]
\[ A \rightarrow a \]
\[ B \rightarrow b \]
\[ C \rightarrow c \]
\[ N \rightarrow SA \]
\[ M \rightarrow CC \]

Step 5: Reduce multi-variable, \( S \rightarrow AN|BX \)
\[ S \rightarrow AN|BX \]
\[ X \rightarrow YM \]
\[ Y \rightarrow d|c \]
\[ A \rightarrow a \]
\[ B \rightarrow b \]
\[ C \rightarrow c \]

Ambiguity – examples

A grammar may generate a string in multiple ways

Math example:
\[ Expr \rightarrow Expr + Expr | Expr \times Expr | Expr | a \]

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

Conversion practice

Non-normal form:

\[
S \rightarrow aa|bXc \\
X \rightarrow Xc|Y \\
Y \rightarrow Ycc|a
\]

Conversion practice

**Step 1: Replace unit rules**

<table>
<thead>
<tr>
<th>Non-normal form</th>
<th>Step 1: Replace unit rules</th>
<th>Step 2: Replace terminals</th>
</tr>
</thead>
<tbody>
<tr>
<td>( S \rightarrow aa</td>
<td>bXc )</td>
<td>( S \rightarrow AA</td>
</tr>
<tr>
<td>( X \rightarrow Xc</td>
<td>Y )</td>
<td>( Y \rightarrow YCC</td>
</tr>
<tr>
<td>( Y \rightarrow Ycc</td>
<td>a )</td>
<td>( A \rightarrow a )</td>
</tr>
<tr>
<td></td>
<td>( B \rightarrow b )</td>
<td>( C \rightarrow c )</td>
</tr>
</tbody>
</table>

Conversion practice

**Step 2: Replace terminals**

<table>
<thead>
<tr>
<th>Non-normal form</th>
<th>Step 1: Replace unit rules</th>
<th>Step 2: Replace terminals</th>
<th>Step 3: Reduce multi-var</th>
</tr>
</thead>
<tbody>
<tr>
<td>( S \rightarrow AA</td>
<td>BXC )</td>
<td></td>
<td>( S \rightarrow AA</td>
</tr>
<tr>
<td>( X \rightarrow XC</td>
<td>YCC</td>
<td>a )</td>
<td></td>
</tr>
<tr>
<td>( Y \rightarrow YCC</td>
<td>a )</td>
<td></td>
<td>( Y \rightarrow YM</td>
</tr>
<tr>
<td>( A \rightarrow a )</td>
<td></td>
<td>( A \rightarrow a )</td>
<td>( A \rightarrow a )</td>
</tr>
<tr>
<td>( B \rightarrow b )</td>
<td></td>
<td>( B \rightarrow b )</td>
<td>( B \rightarrow b )</td>
</tr>
<tr>
<td>( C \rightarrow c )</td>
<td></td>
<td>( C \rightarrow c )</td>
<td>( C \rightarrow c )</td>
</tr>
<tr>
<td>( N \rightarrow XC )</td>
<td></td>
<td>( N \rightarrow XC )</td>
<td>( N \rightarrow XC )</td>
</tr>
<tr>
<td>( M \rightarrow CC )</td>
<td></td>
<td>( M \rightarrow CC )</td>
<td>( M \rightarrow CC )</td>
</tr>
</tbody>
</table>
Push down automata

FSA augmented with memory
Equivalent to CFG *if use non-determinism*

Finite control: transition function
Tape: holds input string
Stack: Can write to/read from stack
Input is Last In First Out ("LIFO")

PDA and Language $0^n1^n$

Read symbol from input, push each 0 onto stack
As soon as see 1’s, start popping 0 for each 1 seen
• If finish reading and stack empty, accept
• If stack is empty and 1’s remain, reject
• If inputs finished but stack still has 0’s, reject
• In 0 appears on input, reject

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
• $\delta: Q \times \Sigma \varepsilon \times \Gamma \varepsilon \rightarrow P(Q \times \Gamma \varepsilon)$ 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 $\delta$

$a, b \to c$ 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 $\varepsilon$

- If $a$ is $\varepsilon$ then change stack without reading a symbol
- If $b$ is $\varepsilon$ then push new symbol $c$ without popping $b$
- If $c$ is $\varepsilon$ 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, $\varepsilon \to 0$  1, $0 \to \varepsilon$

PDA to accept $\{ww^R\}$

Power of non-determinism:

- At start, don’t know where string $w$ ends

0, $\varepsilon \to 0$  0, $0 \to \varepsilon$

1, $\varepsilon \to 1$  1, $1 \to \varepsilon$
PDA to accept $a^i b^i c^k$, $i=j$ or $j=k$

Power of non-determinism:
- At start, don’t know if $i=j$ or $j=k$

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!
Non Context Free Languages

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

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

CFL pumping: Proof idea

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

Example Grammar: \( S \rightarrow uRz \)
\( R \rightarrow x \mid vRy \)
Prove $F=\{ww \mid w=(0 \cup 1)^*\}$ not CFL

Try a sample string $s=0^p10^p1$ \(|s|>p$

• Can we define $uvxyz=s$ so $uv^ixy^iz \in F$?
• Yes: $u=0^{p-1}$, $v=0$, $x=1$, $y=0$, $z=0^{p-1}1$

Try another sample string $s=0^p1^p0^p1^p$

• Can we define $uvxyz=s$ so $uv^ixy^iz \in F$?
• 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 $0^p1^n$ pattern