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

2

An example Context-Free Grammar

Grammar G1

$A \rightarrow 0A1$

$A \rightarrow B$

$B \rightarrow \#$

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

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

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

One start variable: A

Substitution rules/productions
- Variable -> Variables, Terminals

3

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

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

4

Formal CFG Definition

A CFG is a 4-tuple (V, Σ, R, S)
- V is finite set of variables
- Σ finite set of terminals
- R finite set of rules
- $S \in V$ start variable

5

YetaAnother example

$G3 = (\{S\}, \{a, b\}, R, S)$

R: $S \rightarrow aSb \mid SS \mid \varepsilon$

Example strings generated:

L(G1) = { }

6

Another example

$G3 = (\{S\}, \{a, b\}, R, S)$

R: $S \rightarrow aSb \mid SS \mid \varepsilon$

Example rule expansion:

S -> aSb	S -> SS
aaSbb	aSb aSb
aaεbb	aεb aaSbb
aabb	aεb aaεbb
	abaabb

Example strings generated:
ε, ab, abab, aabb, aaabbbab,
ababababab, abaaabbb, …

L(G3) = {a's & b's; each a is followed by a matching b, every b matches exactly one corresponding preceding a}
(like parenthesis matching)

7

Another example

$G4 = (\{A, B, C\}, \{a, b, c\}, R, A)$

R: $A \rightarrow aA \mid BC \mid \varepsilon$

$B \rightarrow Bb \mid C$

$C \rightarrow c \mid \varepsilon$

Example strings generated: ε, aaa, cbbc, aacc

L(G4) = {Hard to describe… }

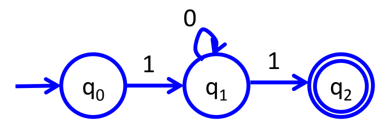
9

Designing CFGs

Creativity required

- If CFL is union of simpler CFL, design grammar for simpler ones (G1, G2, G3), then combine: S -> G1 | G2 | G3
- **If language is regular, can make CFG mimic DFA**

Example: express as CFG



Q_0 -> $1Q_1$
Q_1 -> $0Q_1$ | $1Q_2$
Q_2 -> ε

Designing CFGs

Creativity required

- **If language is regular, can make CFG mimic DFA**

Match each state with a single corresponding variable

$Q=\{q_0,\ldots,q_n\}$ $\quad$ $V=\{R_0, \ldots, R_n\}$

Start state q_0 corresponds to state variable S -> R_0

Replace transition function with Production rule

$\delta(q_i, a) = q_j$ $\quad$ $R_i \to aR_j$

Accept state q_k : transition to ε $\quad$ $R_k \to \varepsilon$

Chomsky Normal Form

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

$A \to BC$

$A \to a$

- 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

- $S_0 \to S$ where S was original start variable
- Remove $A \to \varepsilon$
- Shortcut all unit rules
  Given $A \to B$ and $B \to u$, add $A \to u$
- Replace variable-terminal rules with variable-variable rules
  Given $A \to Bc$, add $U_C \to c$ and change A to $A \to BU_C$
- Replace rules $A \to u_1 u_2 u_3 \dots u_k$ with:
  $A \to u_1 A_1,\ A_1 \to u_2 A_2,\ A_2 \to u_3 A_3,\ \dots,\ A_{k-2} \to u_{k-1} u_k$

Conversion practice

Non-normal form:

$S \to aSa|bX$
$X \to Ycc|\varepsilon$
$Y \to d|c$

Step 1: S_0->S,

$S_0 \to S$
$S \to aSa|bX$
$X \to Ycc|\varepsilon$
$Y \to d|c$

Step 2: Remove ε,

$S_0 \to S$
$S \to aSa|bX|b$
$X \to Ycc$
$Y \to d|c$

Step 3: Use unit rules,

$S_0 \to aSa|bX|b$
$S \to aSa|bX|b$
$X \to Ycc$
$Y \to d|c$

Conversion practice

Step 3: Use unit rules,

$S_0 \to aSa|bX|b$
$S \to aSa|bX|b$
$X \to Ycc$
$Y \to d|c$

Step 4: Replace terminals,

$S_0 \to ASA|BX|b$
$S \to ASA|BX|b$
$X \to YCC$
$Y \to d|c$
$A \to a$
$B \to b$
$C \to c$

Step 5: Reduce multi-variable

$S_0 \to AN|BX|b$
$S \to AN|BX|b$
$X \to YM$
$Y \to d|c$
$A \to a$
$B \to b$
$C \to c$
$N \to SA$
$M \to CC$

Ambiguity – examples

A grammar may generate a string in multiple ways

Math example:

Expr $\to$ Expr + Expr | Expr $\times$ Expr | Expr | a

English example:

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:

$S \rightarrow aa|bXc$

$X \rightarrow Xc|Y$

$Y \rightarrow Ycc|a$

Step 1: Replace unit rules

$S \rightarrow aa|bXc$

$X \rightarrow Xc|Ycc|a$

$Y \rightarrow Ycc|a$

Step2: Replace terminals

$S \rightarrow AA|BXC$

$X \rightarrow XC|YCC|a$

$Y \rightarrow YCC|a$

$A \rightarrow a$

$B \rightarrow b$

$C \rightarrow c$

Conversion practice

Step2: Replace terminals

$S \rightarrow AA|BXC$

$X \rightarrow XC|YCC|a$

$Y \rightarrow YCC|a$

$A \rightarrow a$

$B \rightarrow b$

$C \rightarrow c$

Step 3: Reduce multi-var

$S \rightarrow AA|BN$

$X \rightarrow XC|YM|a$

$Y \rightarrow YM|a$

$A \rightarrow a$

$B \rightarrow b$

$C \rightarrow c$

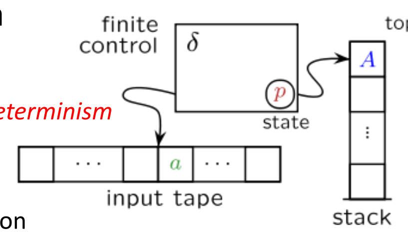
$N \rightarrow XC$

$M \rightarrow CC$

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

25

Definition of PDA

A PDA is a 6-tuple $(Q, \Sigma, \Gamma, \delta, q_0, F)$ where Q, Σ, Γ, and F are finite sets

- Q is sets of states
- Σ is the input alphabet
- Γ is the stack alphabet
- $\delta: Q \times \Sigma\varepsilon \times \Gamma\varepsilon \to P(Q \times \Gamma\varepsilon)$ is transition function
- $q_0 \in Q$ is start state
- $F \subseteq Q$ is set of accept states

26

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

27

Understanding transition δ

$a, b \to c$ 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 ε

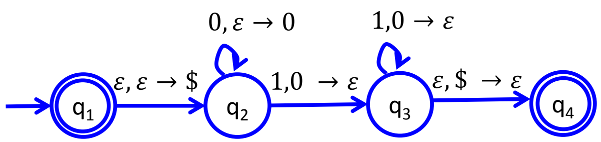
- If a is ε then change stack without reading a symbol
- If b is ε then push new symbol c without popping b
- If c is ε then no new symbol pushed, only pop b

28

PDA to accept 0^n1^n

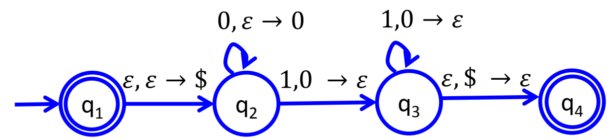
M1 is $(Q, \Sigma, \Gamma, \delta, q_0, F)$

- $Q = \{q_1, q_2, q_3, q_4\}$ $\Sigma = \{0,1\}$
- $\Gamma = \{0, \$\}$ $F = \{q_1, q_4\}$

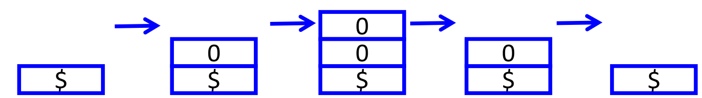


29

PDA to accept 0^n1^n



Input: 0011

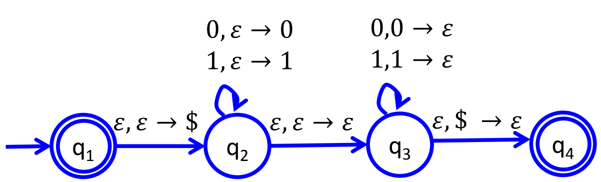


30

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

Power of non-determinism:

- At start, don't know where string w ends

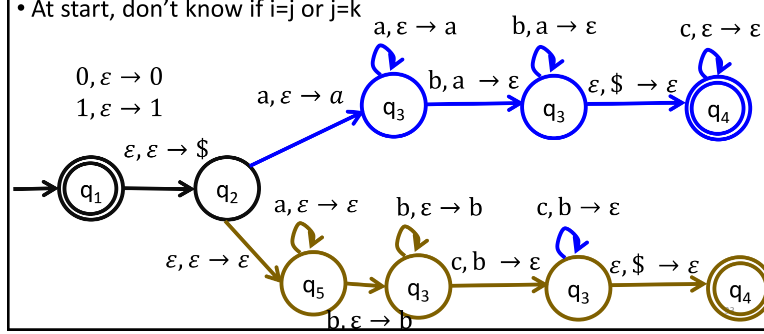


31

PDA to accept $a^ib^jc^k$, 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

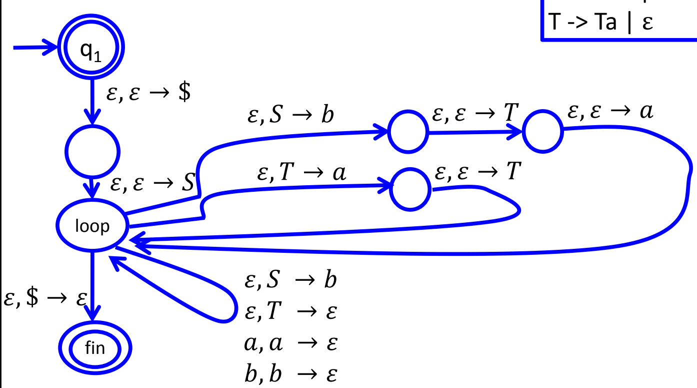
- Stack has set of terminals/variables to compare with input
- Place proper terminal/variable pattern onto stack based on rules
- Non-determinism: Clone your machine, following different branches of rules

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!

Example 2.25 in textbook

S -> aTb | b
T -> Ta | ε



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

- $L=\{ww^R\}$
- $L=\{a^nb^n \mid n\geq 0\}$

Languages **not** recognized by PDAs

- $L=\{ww\}$
- $L=\{a^nb^nc^n \mid n\geq 0\}$

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 $i \geq 0$, $uv^ixy^iz \in A$
2. $|vy| > 0$
3. $|vxy| \leq p$


Regular language PUMPING: Proof idea

If $|s| < p$, trivially true

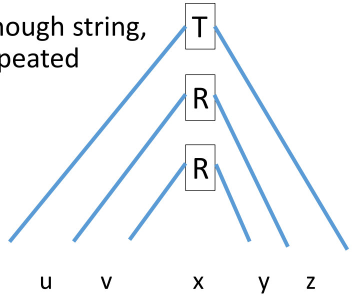
If $|s| \geq p$, consider the states the FSA goes through

- 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: S -> uRz
R -> x | vRy



Prove F={ww | 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^n1^n pattern