

CS 6041

Theory of Computation

Review 2

Kun Suo

Computer Science, Kennesaw State University

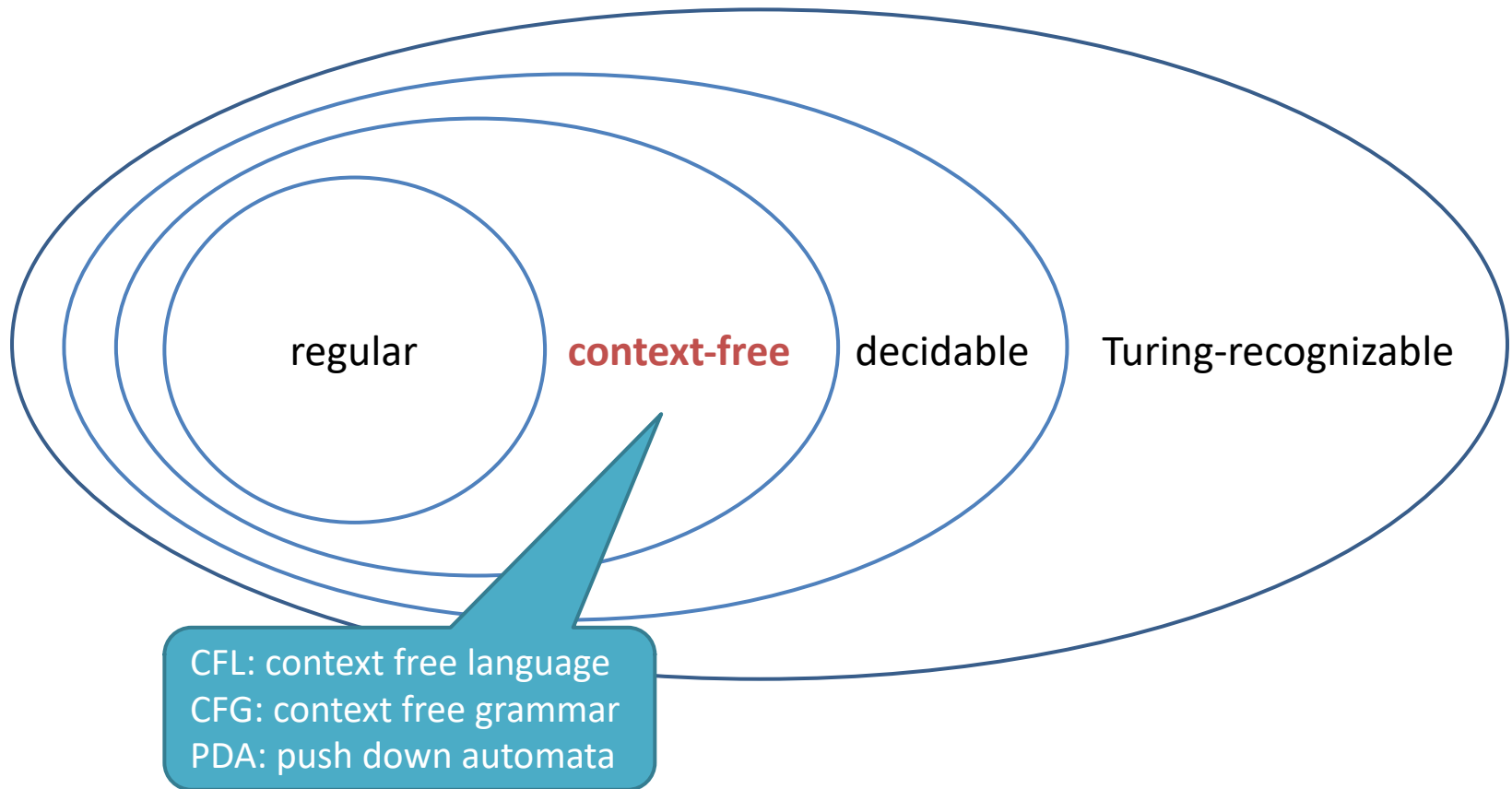
<https://kevinsuo.github.io/>

Exam 2

- 10 True/False question
 - 2 points each
- 4 short answer question
 - 20 points each
- $100 = 2 * 10 + 4 * 20$



Context-free language



Context Free Grammar

- Example, G_1

Variable:

A, B

Start variable:

A

*3 substitution rules
(productions)*

$A \rightarrow 0A1$

$A \rightarrow B$

$B \rightarrow \#$

Terminals:

$0, 1, \#$

$A \Rightarrow 0A1$
 $\Rightarrow 00A11$
 $\Rightarrow 000A111$
 $\Rightarrow 000B111$
 $\Rightarrow 000\#111$



The language of grammar

- Grammar G_1 :

$A \rightarrow 0A1$

$A \rightarrow B$

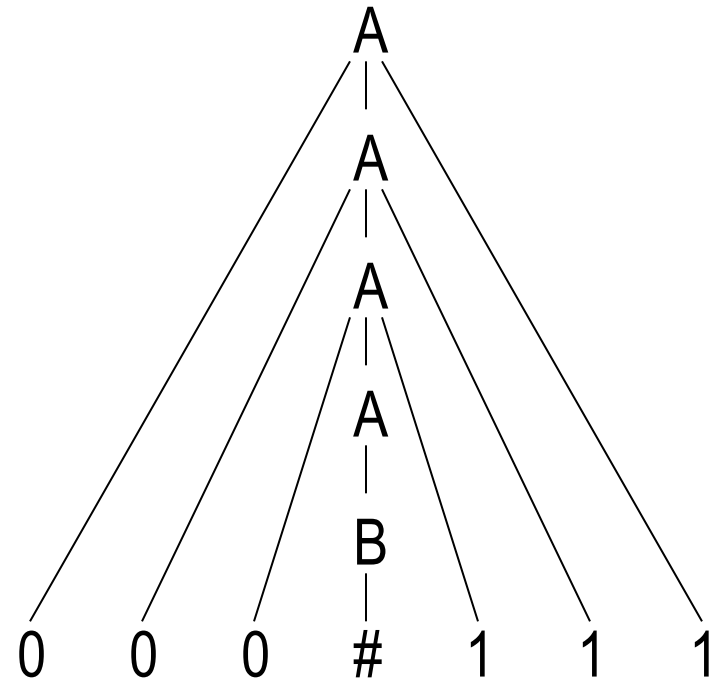
$B \rightarrow \#$

- The language of G_1 :

$L(G_1) = \{ 0^n \# 1^n \mid n \geq 0 \}$

- Context-free language

- Languages generated by context-free grammars



000#111

Definition of context-free grammar

- Context-free grammar is a 4-tuple $G=(V,\Sigma,R,S)$,

1) V : finite variable set

2) Σ : finite terminal set

3) R : finite rule set

$(A \rightarrow w, w \in (V \cup \Sigma)^*)$

4) $S \in V$: start variable



Design CFG for languages

- Design CFG is much difficult than designing an automata for language
- Basic idea:
 1. divide CFL into small parts
 2. design CFG for each small part
 3. combine them together



Design context-free grammar

- Design CFG for $\{w \mid w=0^n1^n \text{ or } w=1^n0^n, n \geq 0\}$



Design context-free grammar

- Design CFG for $\{w \mid w=0^n1^n \text{ or } w=1^n0^n, n \geq 0\}$
 - Design CFG for $\{w \mid w=0^n1^n, n \geq 0\}$
 - Design CFG for $\{w \mid w=1^n0^n, n \geq 0\}$



Design context-free grammar

- Design CFG for $\{w \mid w=0^n1^n \text{ or } w=1^n0^n, n \geq 0\}$
 - Design CFG for $\{w \mid w=0^n1^n, n \geq 0\}$

Generating same number of 0 and 1

Generating 0 before 1

01 0011 000111 00..011..1



Design context-free grammar

- Design CFG for $\{w \mid w=0^n1^n \text{ or } w=1^n0^n, n \geq 0\}$
 - Design CFG for $\{w \mid w=0^n1^n, n \geq 0\}$

Generating same number of 0 and 1

Generating 0 before 1

01 0011 000111 00..011..1



Design context-free grammar

- Design CFG for $\{w \mid w=0^n1^n \text{ or } w=1^n0^n, n \geq 0\}$
 - Design CFG for $\{w \mid w=0^n1^n, n \geq 0\}$

Generating same number of 0 and 1

Generating 0 before 1

01 0011 000111 00..011..1



Design context-free grammar

- Design CFG for $\{w \mid w=0^n1^n \text{ or } w=1^n0^n, n \geq 0\}$
 - Design CFG for $\{w \mid w=0^n1^n, n \geq 0\}$

Generating same number of 0 and 1

Generating 0 before 1

01 0011 000111 00..01..11

$S \rightarrow 0S1$



Design context-free grammar

- Design CFG for $\{w \mid w=0^n1^n \text{ or } w=1^n0^n, n \geq 0\}$
 - Design CFG for $\{w \mid w=0^n1^n, n \geq 0\}$
 - ▶ $G_1 = (\{S\}, \{0,1\}, \{S \rightarrow 0S1, S \rightarrow \epsilon\}, S)$
 - Design CFG for $\{w \mid w=1^n0^n, n \geq 0\}$
 - ▶ $G_2 = (\{S\}, \{0,1\}, \{S \rightarrow 1S0, S \rightarrow \epsilon\}, S)$



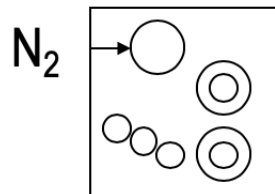
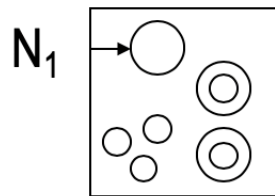
Design context-free grammar

- Design CFG for $\{w \mid w=0^n1^n \text{ or } w=1^n0^n, n \geq 0\}$
 - Design CFG for $\{w \mid w=0^n1^n, n \geq 0\}$
 - ▶ $G_1 = (\{S_1\}, \{0, 1\}, \{S_1 \rightarrow 0S_11, S_1 \rightarrow \varepsilon\}, S_1)$
 - Design CFG for $\{w \mid w=1^n0^n, n \geq 0\}$
 - ▶ $G_2 = (\{S_2\}, \{0, 1\}, \{S_2 \rightarrow 1S_20, S_2 \rightarrow \varepsilon\}, S_2)$

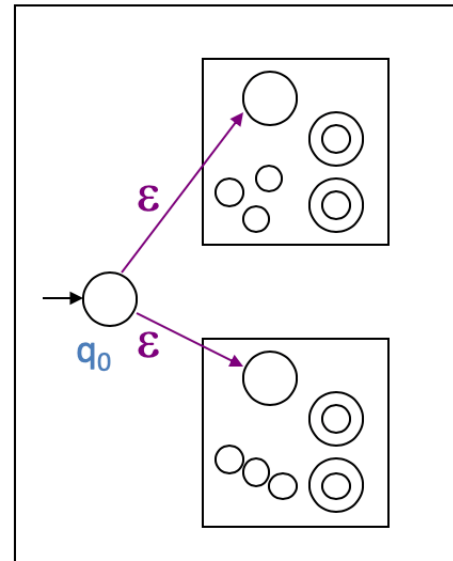


Design context-free grammar

- Design CFG for $\{w \mid w=0^n1^n \text{ or } w=1^n0^n, n \geq 0\}$
 - Design CFG for $\{w \mid w=0^n1^n, n \geq 0\}$
 - ▶ $G_1 = (\{S_1\}, \{0,1\}, \{S_1 \rightarrow 0S_11, S_1 \rightarrow \varepsilon\}, S_1)$
 - Design CFG for $\{w \mid w=1^n0^n, n \geq 0\}$
 - ▶ $G_2 = (\{S_2\}, \{0,1\}, \{S_2 \rightarrow 1S_20, S_2 \rightarrow \varepsilon\}, S_2)$



N



Design context-free grammar

- Design CFG for $\{w \mid w=0^n1^n \text{ or } w=1^n0^n, n \geq 0\}$
 - Design CFG for $\{w \mid w=0^n1^n, n \geq 0\}$
 - ▶ $G_1 = (\{S_1\}, \{0, 1\}, \{S_1 \rightarrow 0S_11, S_1 \rightarrow \varepsilon\}, S_1)$
 - Design CFG for $\{w \mid w=1^n0^n, n \geq 0\}$
 - ▶ $G_2 = (\{S_2\}, \{0, 1\}, \{S_2 \rightarrow 1S_20, S_2 \rightarrow \varepsilon\}, S_2)$
 - $G = (\{S, S_1, S_2\}, \{0, 1\},$
 $\{ S \rightarrow S_1, S \rightarrow S_2, S_1 \rightarrow 0S_11, S_1 \rightarrow \varepsilon, S_2 \rightarrow 1S_20, S_2 \rightarrow \varepsilon\},$
 $S)$



Design CFG for languages

- Design CFG is much difficult than designing an automata for language
- Other ideas:
 1. Simulate the regular expressions
 2. Look for a pattern from example strings
 3. ...



Design CFG for languages

- $L = \{w \mid w \text{ has at least three 1s}\}, \Sigma = \{0,1\}$

$$\Sigma^* 1 \Sigma^* 1 \Sigma^* 1 \Sigma^*$$



Design CFG for languages

- $L = \{w \mid w \text{ has at least three 1s}\}, \Sigma = \{0,1\}$

$$\Sigma^* 1 \Sigma^* 1 \Sigma^* 1 \Sigma^*$$

$$S \rightarrow R1R1R1R$$



Design CFG for languages

- $L = \{w \mid w \text{ has at least three 1s}\}, \Sigma = \{0,1\}$

$$\Sigma^* 1 \Sigma^* 1 \Sigma^* 1 \Sigma^*$$

$$S \rightarrow R1R1R1R$$

$$R \rightarrow 0R$$

$$R \rightarrow 1R$$

$$R \rightarrow \varepsilon$$



Design CFG for languages

- $L = \{w \mid w \text{ has odd length}\}, \Sigma = \{0,1\}$

$$\Sigma(\Sigma \Sigma)^*$$

Design CFG for languages

- $L = \{w \mid w \text{ has odd length}\}, \Sigma = \{0,1\}$

$$\Sigma(\Sigma \Sigma)^*$$

$$\left. \begin{array}{l} S \rightarrow 0 \\ S \rightarrow 1 \end{array} \right\}$$

Design CFG for languages

- $L = \{w \mid w \text{ has odd length}\}, \Sigma = \{0,1\}$

$\Sigma(\Sigma \Sigma)^*$

$S \rightarrow 0$

$S \rightarrow 1$

$S \rightarrow S00$

$S \rightarrow S01$

$S \rightarrow S10$

$S \rightarrow S11$



Design CFG for languages

- $L = \{w \mid w \text{ has odd length}\}, \Sigma = \{0,1\}$

$\Sigma(\Sigma \Sigma)^*$

$S \rightarrow 0$

$S \rightarrow 1$

$S \rightarrow S00$

$S \rightarrow S01$

$S \rightarrow S10$

$S \rightarrow S11$



Design CFG for languages

- $L = \{w \mid w \text{ has odd length and the middle symbol is } 0\}$, $\Sigma = \{0,1\}$

0
000
001
100
101
00011
...



Design CFG for languages

- $L = \{w \mid w \text{ has odd length and the middle symbol is } 0\}$, $\Sigma = \{0,1\}$

	0
$S \rightarrow 0$	000
$S \rightarrow 0S0$	001
$S \rightarrow 0S1$	100
$S \rightarrow 1S0$	101
$S \rightarrow 1S1$	00011
	...



Design CFG for languages

- $L = \{0^n 1^n \mid n \geq 0\}$. $\Sigma = \{0,1\}$

$\varepsilon, 01, 0011, \dots$

$S \rightarrow 0S1 \mid \varepsilon$



Design CFG for languages

- $L = \{0^n 1^{2n} \mid n \geq 0\}$. $\Sigma = \{0, 1\}$

$\epsilon, 011, 001111, \dots$

$S \rightarrow 0S11 \mid \epsilon$



Design CFG for languages

- $L = \{00^*11^*\}$. $\Sigma = \{0,1\}$

01, 011, 0011, ...

How to design 00^*

How to design 11^*



Design CFG for languages

- $L = \{00^*11^*\}$. $\Sigma = \{0,1\}$

How to design 00^*

$C \rightarrow 0$

$C \rightarrow 0C$



Design CFG for languages

- $L = \{00^*11^*\}$. $\Sigma = \{0,1\}$

How to design 11^*

$D \rightarrow 1$

$D \rightarrow 1D$



Design CFG for languages

- $L = \{00^*11^*\}$. $\Sigma = \{0,1\}$

How to design 00^*11^*

$S \rightarrow CD$

$C \rightarrow 0C \mid 0$

$D \rightarrow 1D \mid 1$

How to design 00^*

$C \rightarrow 0$

$C \rightarrow 0C$

How to design 11^*

$D \rightarrow 1$

$D \rightarrow 1D$



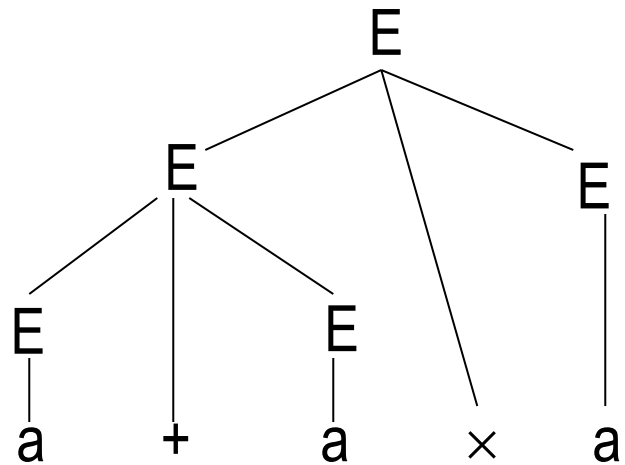
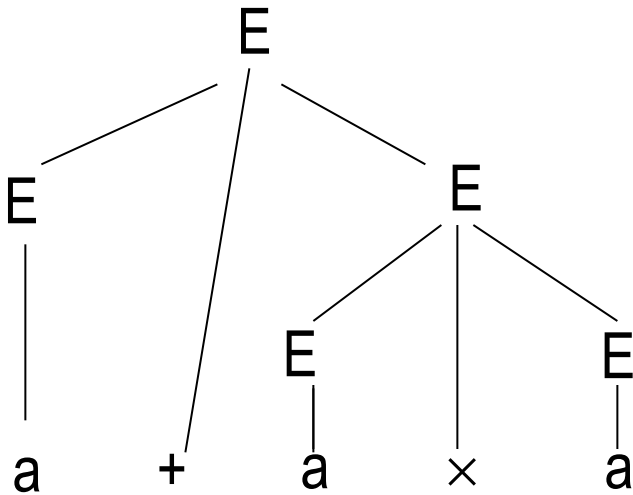
Ambiguity

- If a grammar generates the *same* string in several *different* ways, we say that the string is derived *ambiguously* in that grammar.
- If a grammar generates some string ambiguously, we say that the grammar is *ambiguous*.
- $G_5: E \rightarrow$
 $E + E \mid$
 $E \times E \mid$
 $(E) \mid a$



Ambiguity

- $G_5: E \rightarrow$
 $E + E \mid$
 $E \times E \mid$
 $(E) \mid a$



Leftmost derivation

- A derivation of a string w in a grammar G is a **leftmost derivation** if at every step the **leftmost** remaining variable is the one replaced

- $E \Rightarrow E+E$

$$\Rightarrow a+E$$

$$\Rightarrow a+E \times E$$

$$\Rightarrow a+a \times E \Rightarrow a+a \times a$$

- $G_5: E \rightarrow$

$$E+E \mid$$

$$E \times E \mid$$

$$(E) \mid a$$

Two different leftmost derivation

- E

$\Rightarrow E + E$

$\Rightarrow a + E$

$\Rightarrow a + E \times E$

$\Rightarrow a + a \times E$

$\Rightarrow a + a \times a$

- E

$\Rightarrow E \times E$

$\Rightarrow E + E \times E$

$\Rightarrow a + E \times E$

$\Rightarrow a + a \times E$

$\Rightarrow a + a \times a$

- $G_5: E \rightarrow$

$E + E \mid$

$E \times E \mid$

$(E) \mid a$



Chomsky normal form (CNF)

- CNF: only allow CFG in the following forms

- $S \rightarrow \varepsilon$

Only start variable S
can generate ε

- $A \rightarrow BC$

- $A \rightarrow a$

Variables can only generate:
1, two variables
2, single terminal



$S \rightarrow \varepsilon$



$A \rightarrow BC$



$A \rightarrow a$



$A \rightarrow \varepsilon$



$A \rightarrow B$



$A \rightarrow abcd$



$A \rightarrow aB$



Techniques for CNF

- Add new start variable if needed
- $A \rightarrow \varepsilon$, merge above rules with A
- $A \rightarrow B$, replace B with terminals or other rules
- $A \rightarrow aB$, replace with $U \rightarrow a$, $A \rightarrow UB$
- $A \rightarrow abcd$, replace with $A \rightarrow aU_1$,
 $U_1 \rightarrow bU_2$, $U_2 \rightarrow cd$
- $A \rightarrow BCD$, similar as the above

✓	$S \rightarrow \varepsilon$
✓	$A \rightarrow BC$
✓	$A \rightarrow a$
✗	$A \rightarrow \varepsilon$
✗	$A \rightarrow B$
✗	$A \rightarrow abcd$
✗	$A \rightarrow aB$

Chomsky normal form example

$G_6: S \rightarrow \underset{\times}{A} \underset{\times}{S} A \mid a \underset{\times}{B},$

$A \rightarrow \underset{\times}{B} \mid \underset{\times}{S}$

$B \rightarrow \underset{\checkmark}{b} \mid \underset{\times}{\varepsilon}$

Get the CNF for G_6



$S \rightarrow \varepsilon$



$A \rightarrow BC$



$A \rightarrow a$



$A \rightarrow \varepsilon$



$A \rightarrow B$



$A \rightarrow abcd$



$A \rightarrow aB$

Step 1

$$G_6: S \rightarrow ASA \mid aB,$$

$$A \rightarrow B \mid S$$

$$B \rightarrow b \mid \varepsilon$$

Red 'X' marks are placed under the 'A' in 'ASA', the 'B' in 'B', the 'S' in 'S', the 'B' in 'B', and the 'ε' in 'ε'. A green checkmark is placed under the 'b' in 'b'.

- ✓ $S \rightarrow \varepsilon$
- ✓ $A \rightarrow BC$
- ✓ $A \rightarrow a$
- ✗ $A \rightarrow \varepsilon$
- ✗ $A \rightarrow B$
- ✗ $A \rightarrow abcd$
- ✗ $A \rightarrow aB$

(1) $S_0 \rightarrow S$ ✗

To simply the start variable


$$S \rightarrow ASA \mid aB$$

$$A \rightarrow B \mid S$$



$$B \rightarrow b \mid \varepsilon$$



Red 'X' marks are placed under the 'A' in 'ASA', the 'B' in 'B', the 'S' in 'S', the 'B' in 'B', and the 'ε' in 'ε'. A green checkmark is placed under the 'b' in 'b'.

Step 2

(2a) $S_0 \rightarrow S$ 

$S \rightarrow ASA \mid aB$
 

$A \rightarrow B \mid S$
 

$B \rightarrow b \mid \epsilon$
 

Only start variable S
can generate ϵ



$S \rightarrow \epsilon$



$A \rightarrow BC$



$A \rightarrow a$



$A \rightarrow \epsilon$




$A \rightarrow B$



$A \rightarrow abcd$



$A \rightarrow aB$

$S_0 \rightarrow S$ 

$S \rightarrow ASA \mid aB \mid a$
  

$A \rightarrow B \mid S \mid \epsilon$
  

$B \rightarrow b$


Step 2

(2b) $S_0 \rightarrow S$

$S \rightarrow ASA$ | aB | a

$A \rightarrow B$ | S | ϵ

$B \rightarrow b$

Only start variable S
can generate ϵ



$S \rightarrow \epsilon$



$A \rightarrow BC$



$A \rightarrow a$



$A \rightarrow \epsilon$



$A \rightarrow B$



$A \rightarrow abcd$



$A \rightarrow aB$

$S_0 \rightarrow S$











$S \rightarrow ASA$ | aB | a | SA | AS | S








$A \rightarrow B$ | S










$B \rightarrow b$

No $A \rightarrow \epsilon$
anymore

Step 3

(3a) $S_0 \rightarrow S$ 
 $S \rightarrow ASA \mid aB \mid a \mid SA \mid AS \mid S$      
 $A \rightarrow B \mid S$  
 $B \rightarrow b$ 

 $S \rightarrow \epsilon$
 $A \rightarrow BC$
 $A \rightarrow a$
 $A \rightarrow \epsilon$
 $A \rightarrow B$
 $A \rightarrow abcd$
 $A \rightarrow aB$

$S_0 \rightarrow S$ 
 $S \rightarrow ASA \mid aB \mid a \mid SA \mid AS$     
 $A \rightarrow B \mid S$  
 $B \rightarrow b$ 

Step 3

(3b) $S_0 \rightarrow S$
 $S \rightarrow ASA \mid aB \mid a \mid SA \mid AS$
 $A \rightarrow B \mid S$
 $B \rightarrow b$

✓ $S \rightarrow \epsilon$
✓ $A \rightarrow BC$
✓ $A \rightarrow a$
✗ $A \rightarrow \epsilon$
✗ $A \rightarrow B$
✗ $A \rightarrow abcd$
✗ $A \rightarrow aB$

$S_0 \rightarrow ASA \mid aB \mid a \mid SA \mid AS$
 $S \rightarrow ASA \mid aB \mid a \mid SA \mid AS$
 $A \rightarrow B \mid S$
 $B \rightarrow b$



Step 3

(3c) $S_0 \rightarrow ASA \mid aB \mid a \mid SA \mid AS$
 $S \rightarrow ASA \mid aB \mid a \mid SA \mid AS$
 $A \rightarrow B \mid S$
 $B \rightarrow b$

$S_0 \rightarrow ASA \mid aB \mid a \mid SA \mid AS$
 $S \rightarrow ASA \mid aB \mid a \mid SA \mid AS$
 $A \rightarrow b \mid S$
 $B \rightarrow b$

✓ $S \rightarrow \varepsilon$
 ✓ $A \rightarrow BC$
 ✓ $A \rightarrow a$
 ✗ $A \rightarrow \varepsilon$
 ✗ $A \rightarrow B$
 ✗ $A \rightarrow abcd$
 ✗ $A \rightarrow aB$



Step 3

(3d) $S_0 \rightarrow ASA \mid aB \mid a \mid SA \mid AS$
 $S \rightarrow ASA \mid aB \mid a \mid SA \mid AS$
 $A \rightarrow b \mid S$
 $B \rightarrow b$

$S_0 \rightarrow ASA \mid aB \mid a \mid SA \mid AS$
 $S \rightarrow ASA \mid aB \mid a \mid SA \mid AS$
 $A \rightarrow b \mid ASA \mid aB \mid a \mid SA \mid AS$
 $B \rightarrow b$

✓ $S \rightarrow \epsilon$
 ✓ $A \rightarrow BC$
 ✓ $A \rightarrow a$
 ✗ $A \rightarrow \epsilon$
 ✗ $A \rightarrow B$
 ✗ $A \rightarrow abcd$
 ✗ $A \rightarrow aB$



Step 4

(4) $S_0 \rightarrow \text{ASA} \mid aB \mid a \mid SA \mid AS$
 $S \rightarrow \text{ASA} \mid aB \mid a \mid SA \mid AS$
 $A \rightarrow b \mid \text{ASA} \mid aB \mid a \mid SA \mid AS$
 $B \rightarrow b$

$S_0 \rightarrow \text{AA}_1 \mid aB \mid a \mid SA \mid AS$
 $S \rightarrow \text{AA}_1 \mid aB \mid a \mid SA \mid AS$
 $A \rightarrow b \mid \text{AA}_1 \mid aB \mid a \mid SA \mid AS$
 $B \rightarrow b$
 $A_1 \rightarrow SA$

✓ $S \rightarrow \varepsilon$
 ✓ $A \rightarrow BC$
 ✓ $A \rightarrow a$
 ✗ $A \rightarrow \varepsilon$
 ✗ $A \rightarrow B$
 ✗ $A \rightarrow abcd$
 ✗ $A \rightarrow aB$



Step 5

(5) $S_0 \rightarrow AA_1 \mid aB \mid a \mid SA \mid AS$
 $S \rightarrow AA_1 \mid aB \mid a \mid SA \mid AS$
 $A \rightarrow b \mid AA_1 \mid aB \mid a \mid SA \mid AS$
 $B \rightarrow b$
 $A_1 \rightarrow SA$

$S_0 \rightarrow AA_1 \mid UB \mid a \mid SA \mid AS$
 $S \rightarrow AA_1 \mid UB \mid a \mid SA \mid AS$
 $A \rightarrow b \mid AA_1 \mid UB \mid a \mid SA \mid AS$
 $B \rightarrow b$
 $A_1 \rightarrow SA$
 $U \rightarrow a$

✓ $S \rightarrow \varepsilon$
 ✓ $A \rightarrow BC$
 ✓ $A \rightarrow a$
 ✗ $A \rightarrow \varepsilon$
 ✗ $A \rightarrow B$
 ✗ $A \rightarrow abcd$
 ✗ $A \rightarrow aB$

Step 5

(5) $S_0 \rightarrow AA_1 \mid \text{UB} \mid a \mid$

$SA \mid AS$

$S \rightarrow AA_1 \mid \text{UB} \mid a \mid$

$SA \mid AS$

$A \rightarrow b \mid AA_1 \mid \text{UB} \mid a \mid$

$SA \mid AS$

$B \rightarrow b$

$A_1 \rightarrow SA$

$\text{U} \rightarrow a$



G_6 in CNF

$G_6: S \rightarrow ASA \mid aB,$

$A \rightarrow B \mid S$

$B \rightarrow b \mid \varepsilon$



$S \rightarrow \varepsilon$



$A \rightarrow BC$



$A \rightarrow a$



$A \rightarrow \varepsilon$



$A \rightarrow B$



$A \rightarrow abcd$



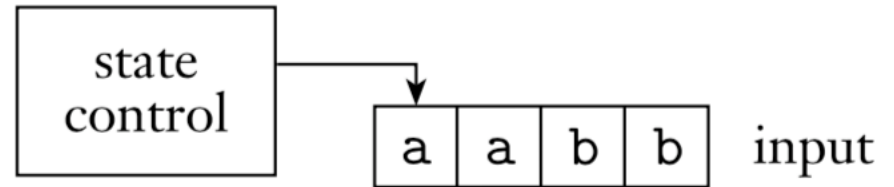
$A \rightarrow aB$



What does PDA look like?

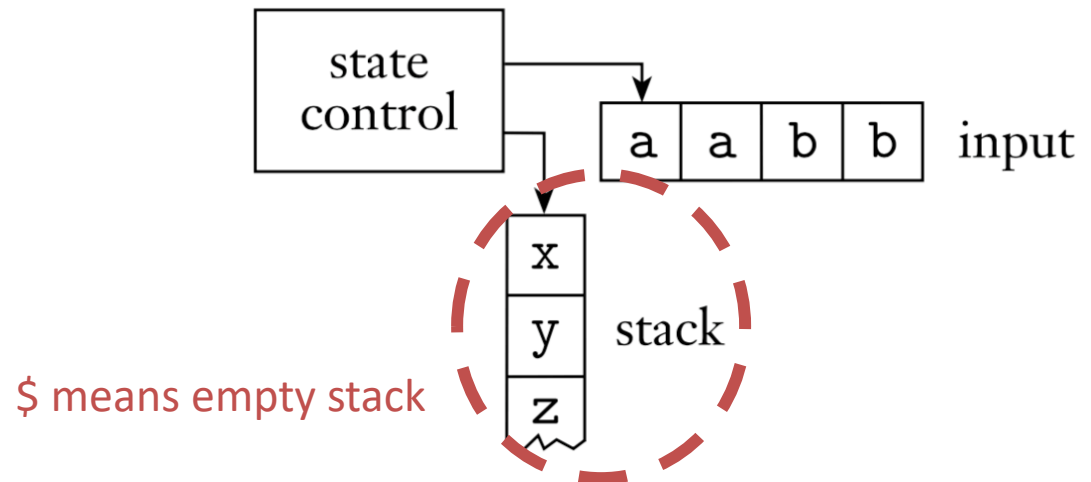
finite automaton

Memory = 1



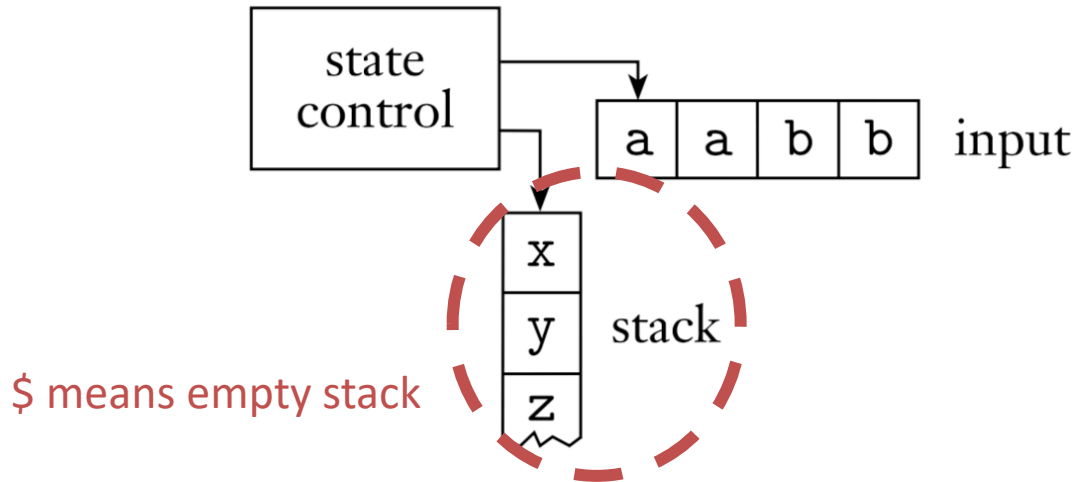
pushdown automaton

Memory = N



What does PDA looks like?

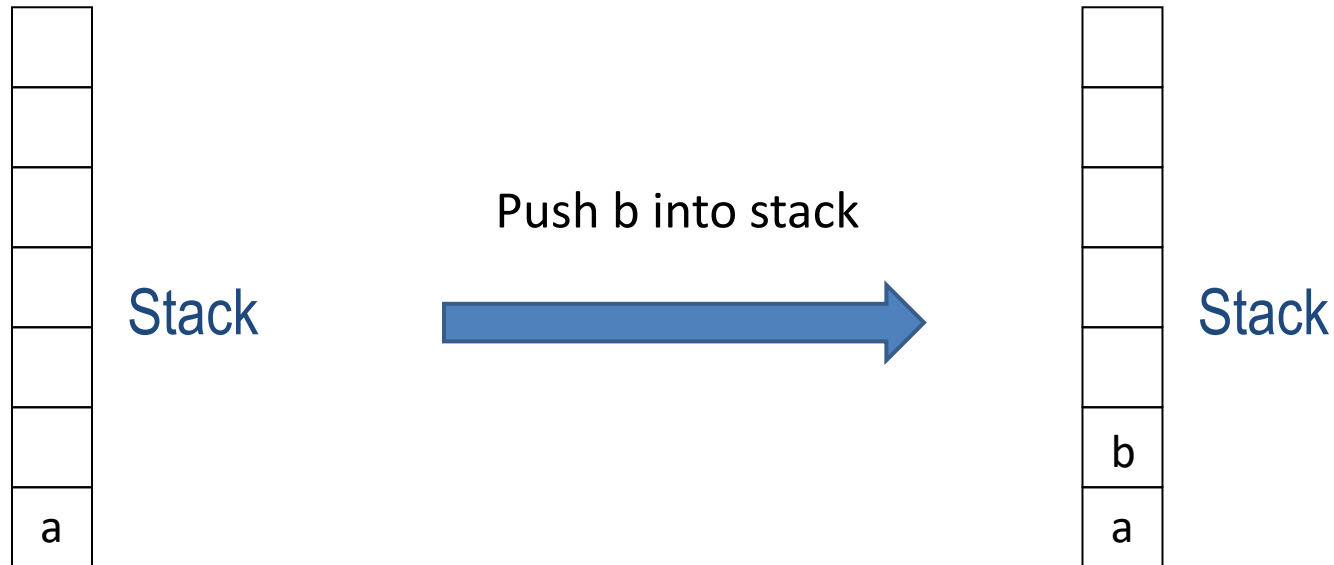
pushdown automaton



- Pushdown automata has more memories than finite automata
- PDA = finite automata + **A stack (unlimited size)**

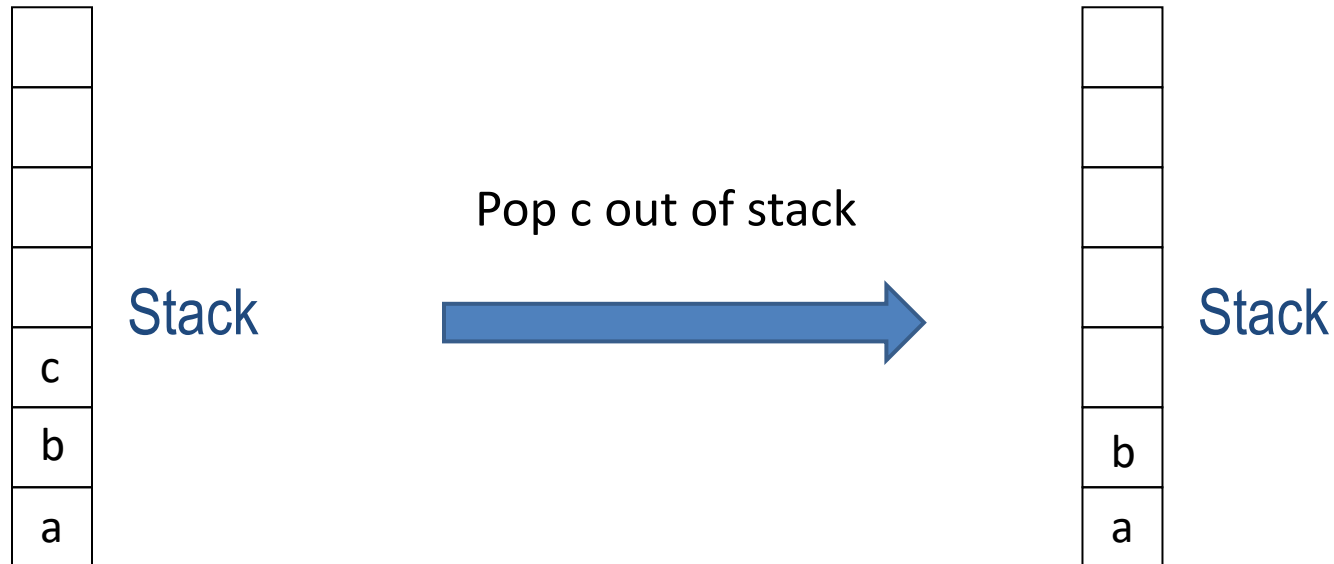
Stack operation

- Push: add to the top of stack



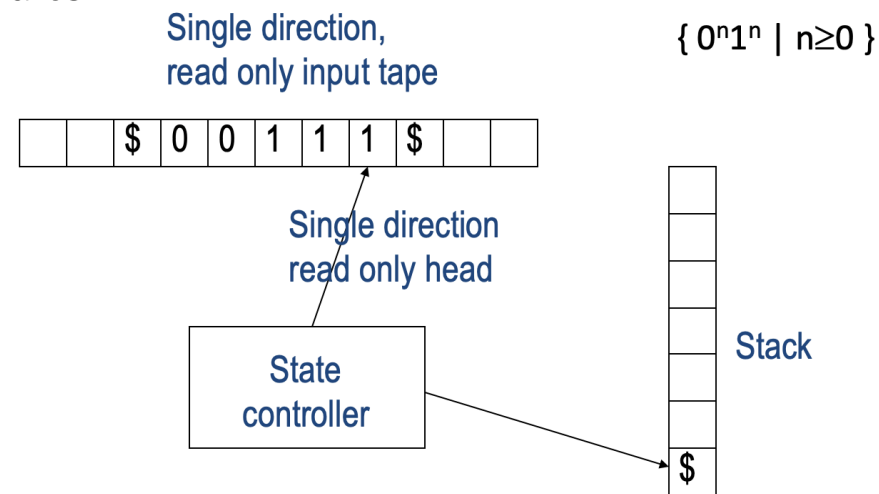
Stack operation

- Pop: remove from the top of stack



Informal description for PDA to recognize some languages

- $L = \{w \mid w \text{ has some features}\}$
- Read symbols from input
 - **STEP1: regular?**
 - If the language is regular, do not need to use stack; if not regular, define operations on stack
 - **STEP2: define operations:**
 - When to push
 - When to pop
 - **STEP 3: determine accept/reject:**
 - Under which cases, accept
 - Under which cases, reject



Definition of PDA (non-deterministic)

- PDA $M=(Q,\Sigma,\Gamma,\delta,q_0,F)$, where
 - 1) Q : set of states
 - 2) Σ : input alphabet, $\Sigma_\epsilon=\Sigma\cup\{\epsilon\}$
 - 3) Γ : stack alphabet, $\Gamma_\epsilon=\Gamma\cup\{\epsilon\}$
 - 4) $\delta: Q\times\Sigma_\epsilon\times\Gamma_\epsilon\rightarrow P(Q\times\Gamma_\epsilon)$,
transition function
 - 5) $q_0\in Q$: start state
 - 6) $F\subseteq Q$: accept state set



Design PDA

\$	aaa...a	bbb...b	ccc...c
q ₁	q ₂	q ₃	q ₄

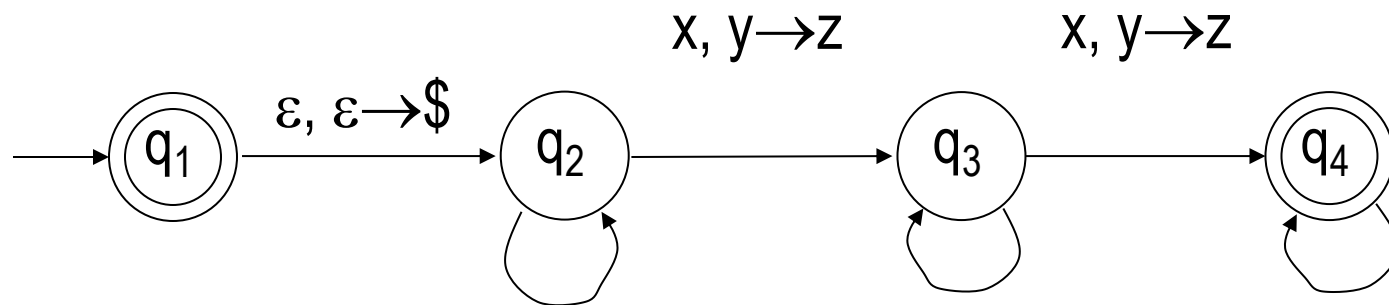
- $L(M_2) = \{ a^n b^n c^m \mid m, n \geq 0 \}$
 - ▶ Operation:
 - For an input a, and push a into stack
 - For an input b, pop one a from the stack
 - ▶ Determine accept/reject
 - If the stack is empty when finish reading b, then after reading all the cs, accept;
 - Otherwise, reject;



Design PDA

\$	aaa...a	bbb...b	ccc...c
q_1	q_2	q_3	q_4

- $L(M_2) = \{ a^n b^n c^m \mid m, n \geq 0 \}$



$x, y \rightarrow z$
 x : input
 $y \rightarrow z$: the top of stack changes

- $L(M_2) = \{ a^n b^n c^m \mid m, n \geq 0 \}$ $x, y \rightarrow z$ $x, y \rightarrow z$ $x, y \rightarrow z$

Operation:

- For an input a , and push a into stack
- For an input b , pop one a from the stack

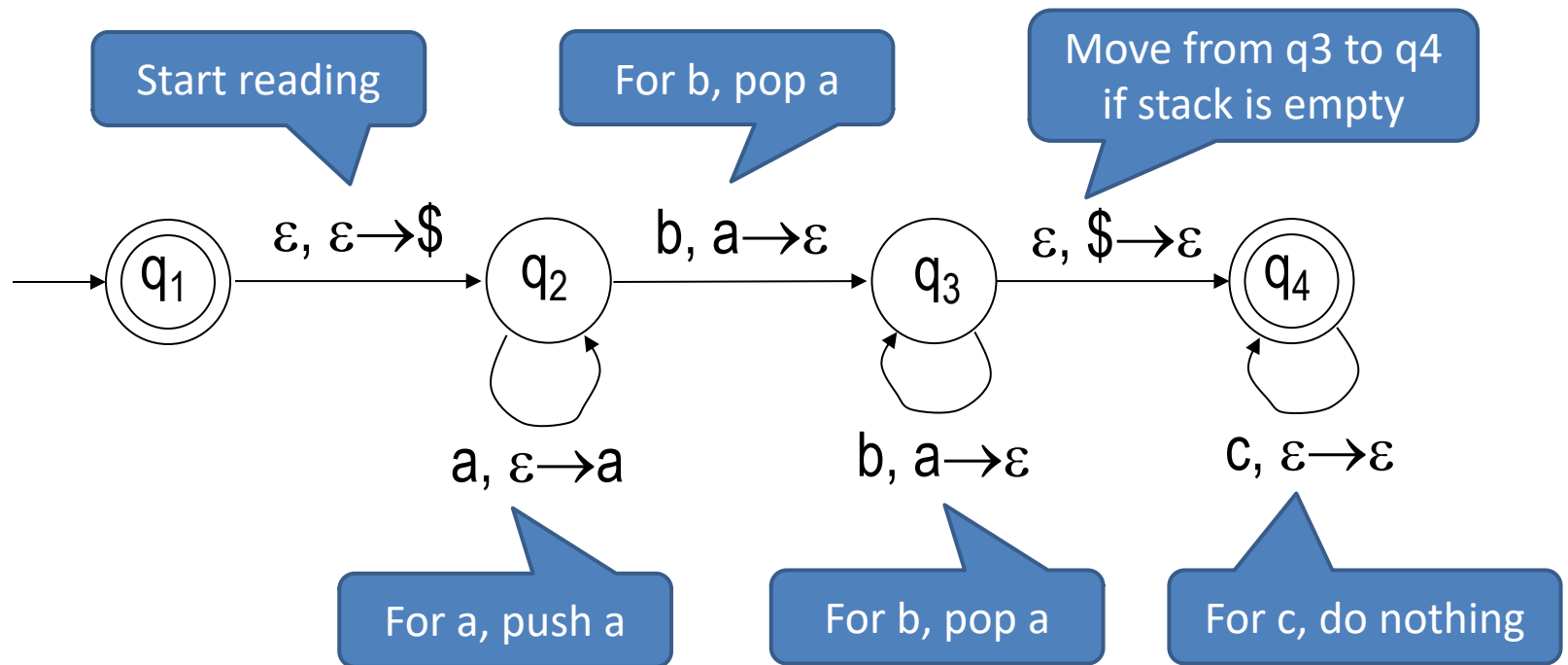
Determine accept/reject

- If the stack is empty when finish reading b , then after reading all the c s, accept;
- Otherwise, reject;

Design PDA

\$	aaa...a	bbb...b	ccc...c
q_1	q_2	q_3	q_4

- $L(M_2) = \{ a^n b^n c^m \mid m, n \geq 0 \}$



A language is CFL \Rightarrow some PDA recognizes it

- Details: how PDA recognizes a string of a CFL
 - Step 1: Compare the input with the top of stack
 - Step 2: If the top of stack is variable, then simulate the derivation
 - Step 3: If the top of stack is terminal, then do the match



Conclusion for A language is CFL \Rightarrow some PDA recognizes it

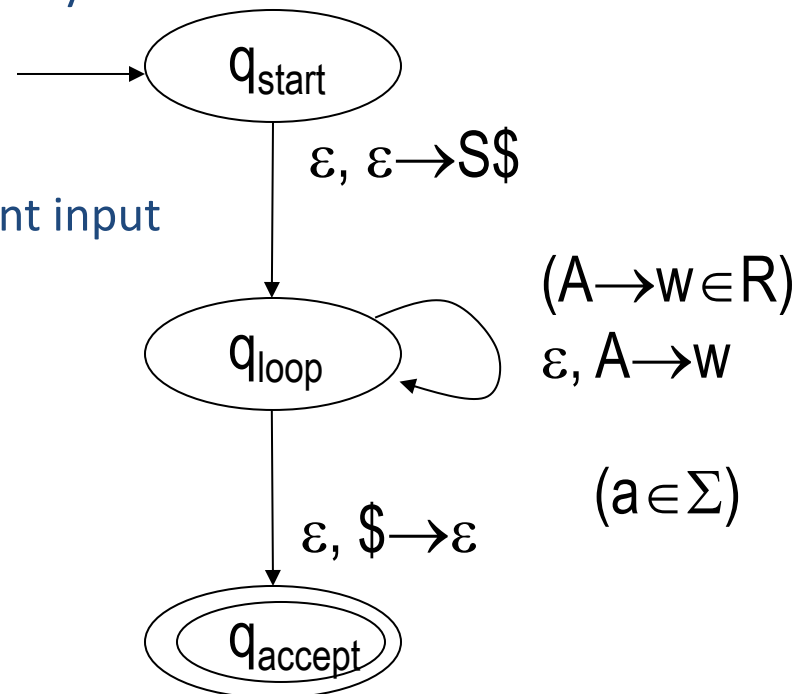
Put \$ and start variable into stack

Repeat the following steps

- If the top of stack is variable A, then pick up one rule $A \rightarrow w$, and replace A by w.
- If the top of stack is terminal a, then compare a with the symbol of current input
if match, repeat;
if not match, reject this branch.

If the top of stack is \$,
if the input is over, accept.

Create a PDA like this:



Example of CFL \Rightarrow PDA

- CFG G: $S \rightarrow aTb$, $T \rightarrow Ta$, $S \rightarrow b$, $T \rightarrow \varepsilon$

construct an equivalent PDA P_1 .



Example of CFL \Rightarrow PDA

- CFG G: $S \rightarrow aTb$, $T \rightarrow Ta$, $S \rightarrow b$, $T \rightarrow \varepsilon$

construct an equivalent PDA P_1 .

q_{start}

q_{loop}

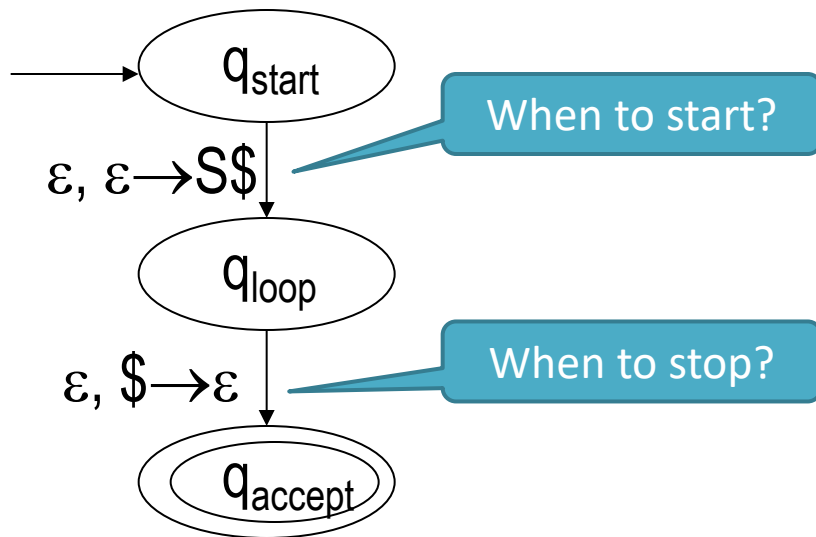
q_{accept}



Example of CFL \Rightarrow PDA

- CFG G : $S \rightarrow aTb$, $T \rightarrow Ta$, $S \rightarrow b$, $T \rightarrow \epsilon$

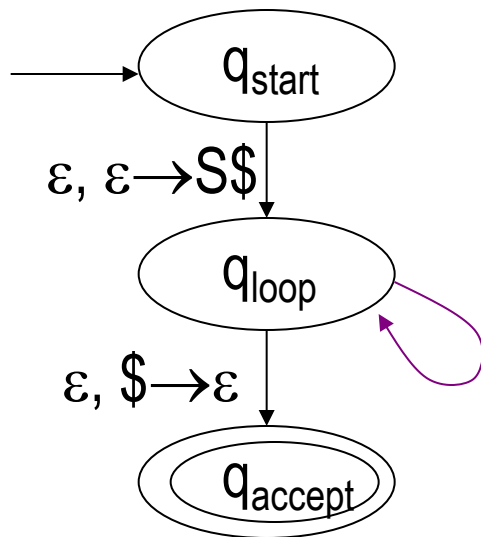
construct an equivalent PDA P_1 .



Example of CFL \Rightarrow PDA

- CFG G : $S \rightarrow aTb$, $T \rightarrow Ta$, $S \rightarrow b$, $T \rightarrow \epsilon$

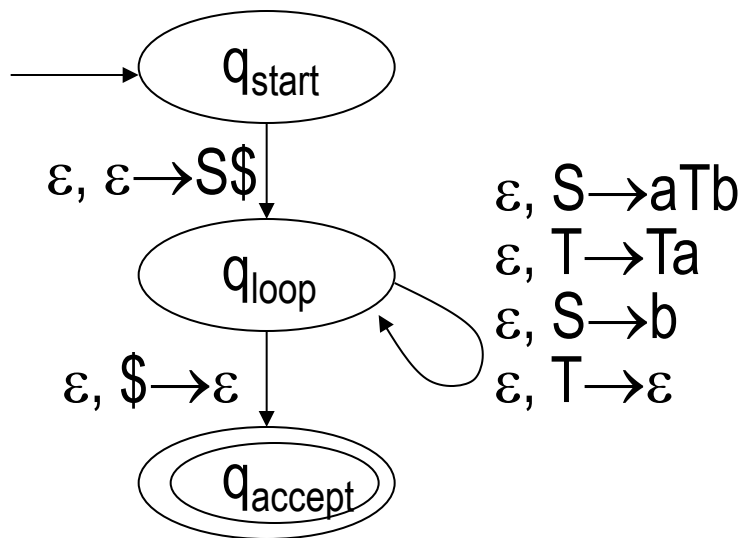
construct an equivalent PDA P_1 .



Example of CFL \Rightarrow PDA

- CFG G: $S \rightarrow aTb$, $T \rightarrow Ta$, $S \rightarrow b$, $T \rightarrow \epsilon$

construct an equivalent PDA P_1 .



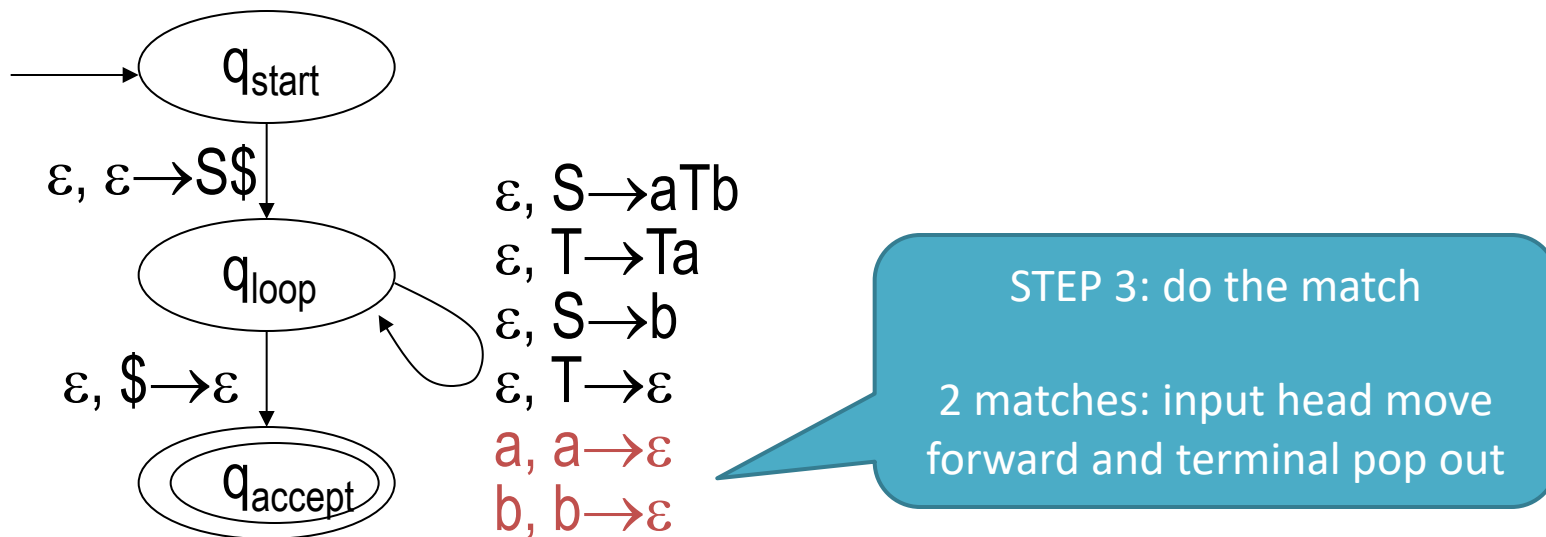
STEP 2: simulate the derivation
4 rules \rightarrow 4 transition functions



Example of CFL \Rightarrow PDA

- CFG G : $S \rightarrow aTb$, $T \rightarrow Ta$, $S \rightarrow b$, $T \rightarrow \epsilon$

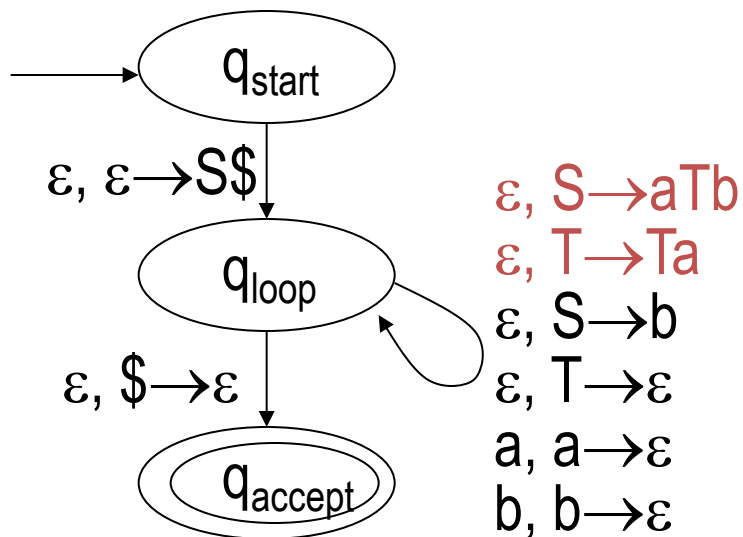
construct an equivalent PDA P_1 .



Example of CFL \Rightarrow PDA

- CFG G : $S \rightarrow aTb$, $T \rightarrow Ta$, $S \rightarrow b$, $T \rightarrow \epsilon$

construct an equivalent PDA P_1 .



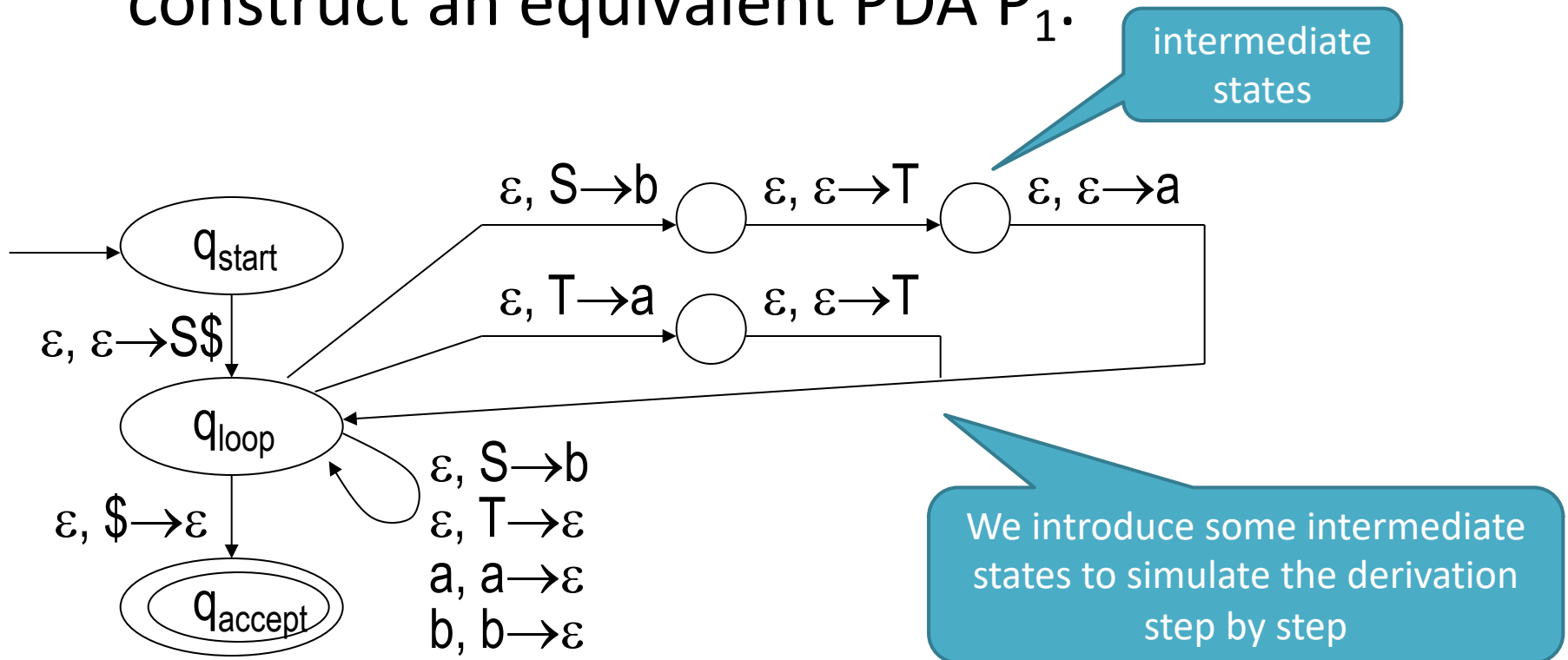
For these which right side has one more symbols, we introduce some intermediate states



Example of CFL \Rightarrow PDA

- CFG G : $S \rightarrow aTb$, $T \rightarrow Ta$, $S \rightarrow b$, $T \rightarrow \epsilon$

construct an equivalent PDA P_1 .



Pumping lemma

Suppose A is CFL,

then there exist a number p (the pumping length) where,

if $s \in A$ and $|s| \geq p$, then $s = UVXYZ$,

Satisfying the following

1) $\forall i \geq 0, uv^i xy^i z \in A$;

2) $|vy| > 0$;

3) $|vxy| \leq p$.



Example: $B = \{ a^n b^n c^n \mid n \geq 0 \}$

1) $\forall i \geq 0, uv^i xy^i z \in A;$

2) $|vy| > 0;$

3) $|vxy| \leq p.$

- **Proof:**

Suppose B is CFL, p is the pumping length,

let $s = a^p b^p c^p$

Then $s = uvxyz$, that

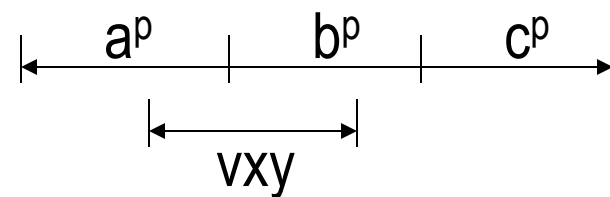
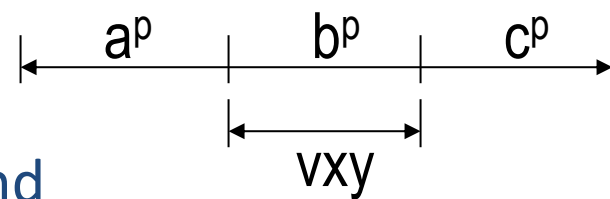
$\forall i \geq 0, uv^i xy^i z \in B;$

$|vy| > 0$, v and y have at least one kind

of symbol;

$|vxy| \leq p$, v and y have at most two

kinds of symbol;



Example: $B = \{ a^n b^n c^n \mid n \geq 0 \}$

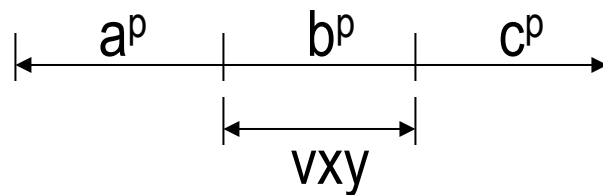
1) $\forall i \geq 0, uv^i xy^i z \in A;$

2) $|vy| > 0;$

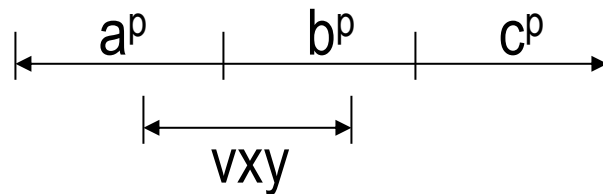
3) $|vxy| \leq p.$

- **Proof:**

If v and y have one kind of symbol,
then in $uv^i xy^i z$ ($i > 1$), $a/b/c$ has different
numbers;



If v and y have two kinds of symbol,
then in $uv^i xy^i z$ ($i > 1$), $a/b/c$ has different
numbers;



Contradiction.



Exam 2

- 10 True/False question
 - 2 points each
- 4 short answer question
 - 20 points each
- Time: Oct 28, 6:30-7:45pm @classroom

