April 3rd Exam

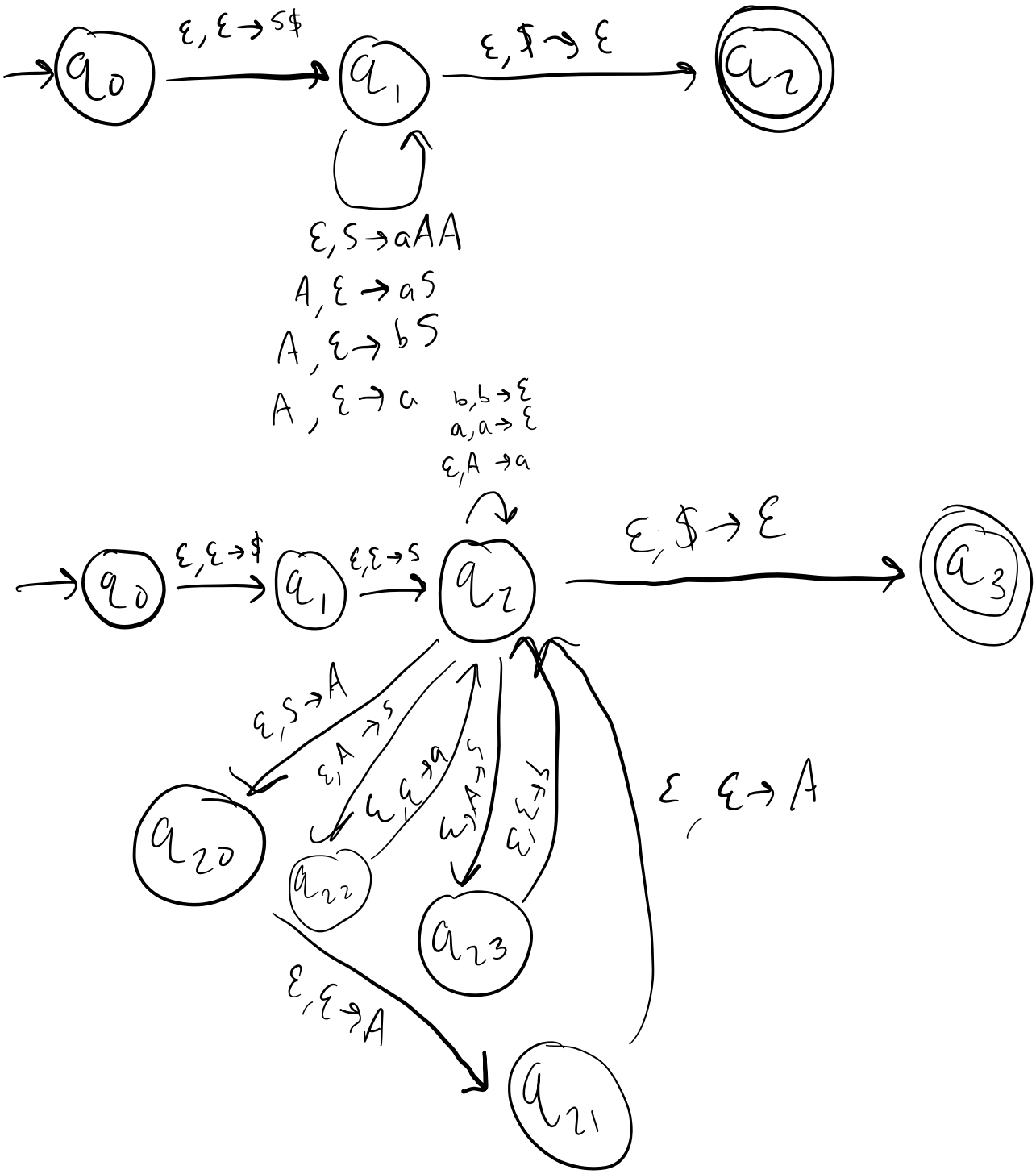
PDA TO CFG

COMP 4200 – Formal Language

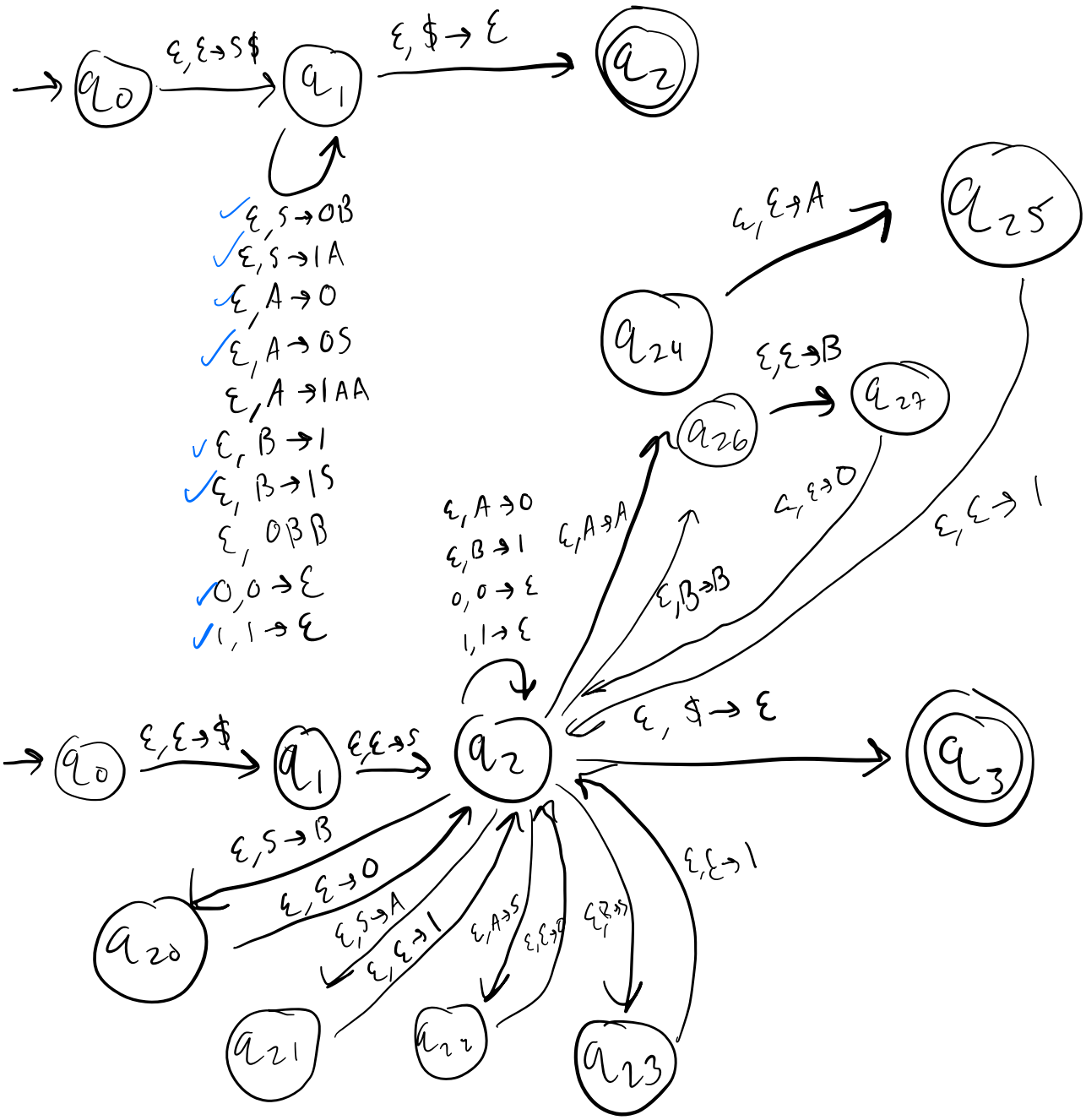
TRY YOURSELF!

1. Convert the following CFG into PDA S → aAA A → aS|bS|a
2. Convert the given CFG to PDA, S → aSb S → a | b | ε
3. Convert the given CFG to PDA, The production of CFG is S → 0B | 1A A →. 0 | 0S | 1AA B → 1 | 1S | 0BB

1. Convert the following CFG into PDA S → aAA A → aS|bS|a



3. Convert the given CFG to PDA, The production of CFG is S → 0B | 1A A →. 0 | 0S | 1AA B → 1 | 1S | 0BB



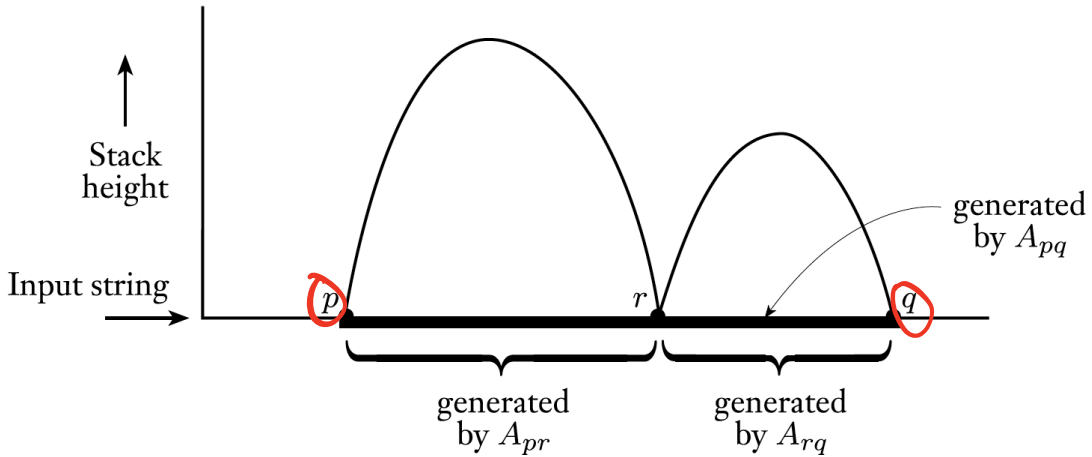
PDA TO CFG

IF A PUSHDOWN AUTOMATON RECOGNIZES SOME LANGUAGE, THEN IT IS CONTEXT FREE.

- For each pair of states p and q in P, the grammar will have a variable A_{pq}. This variable generates all the strings that can take P from p to q.
- Modifying P slightly to give it the following three features.
  - It has a single accept state, q_{accept}.
  - It empties its stack before accepting.
  - Each transition either pushes a symbol onto the stack (a push move) or pops one off the stack (a pop move), but it does not do both at the same time.
- Two possibilities occur during P's computation on x.
  - Either the symbol popped at the end is the symbol that was pushed at the beginning, or not. If so, the stack could be empty only at the beginning and end of P's computation on x.
  - If not, the initially pushed symbol must get popped at some point before the end of x and thus the stack becomes empty at this point.
- We simulate the with the rule $A_{pq} \rightarrow aA_{rs}b$, where a is the input read at the first move, b is the input read at the last move, r is the state following p, and s is the state preceding q.
- We simulate the latter possibility with the rule $A_{pq} \rightarrow A_{pr} A_{rq}$, where r is the state when the stack becomes empty.

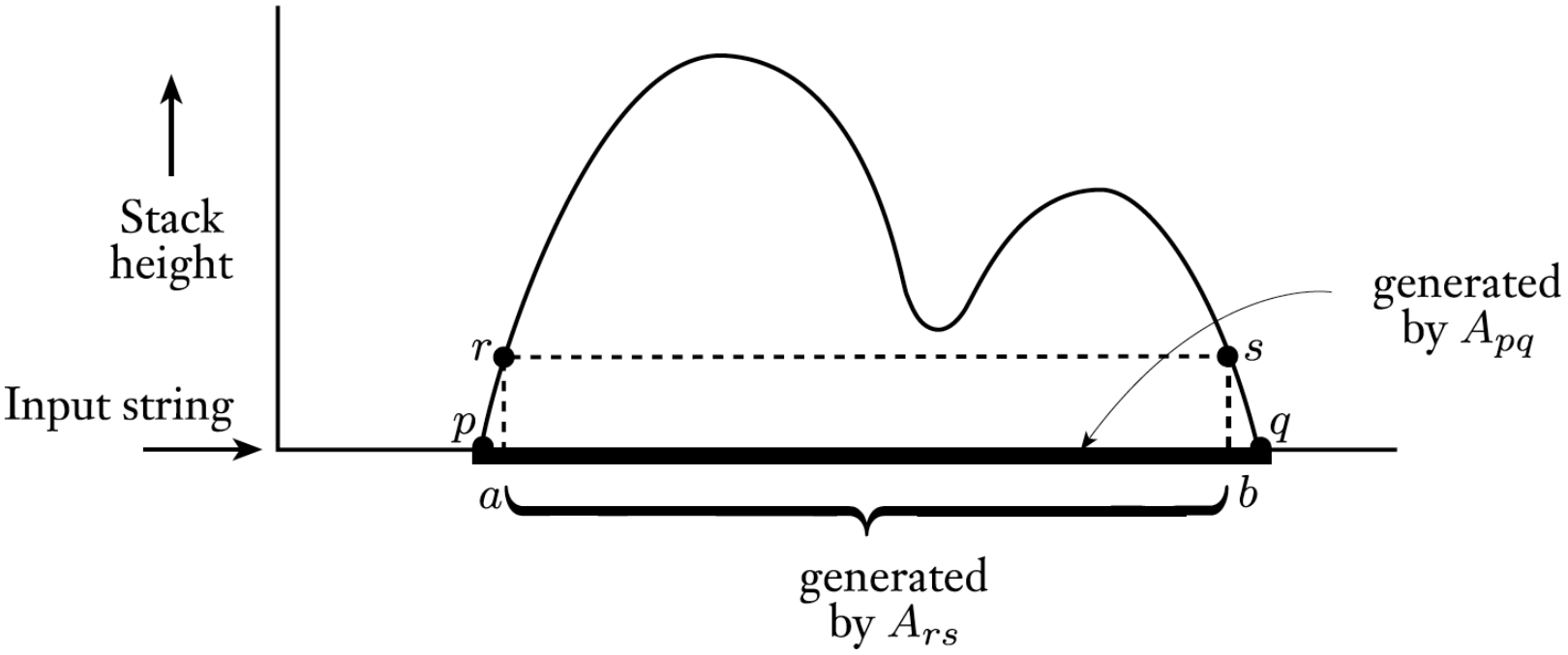
Proof:

- Say that $P = (Q, \Sigma, \Gamma, \delta, q_0, \{q_{accept}\})$ and construct G.
- The variables of G are $\{A_{pq} \mid p, q \in Q\}$.
- The start variable is $A_{q0,q_{accept}}$.
- Now we describe G's rules in three parts.
  - For each $p, q, r, s \in Q$, $u \in \Gamma$, and $a, b \in \Sigma_\varepsilon$, if $\delta(p, a, \varepsilon)$ contains (r, u) and $\delta(s, b, u)$ contains (q, ε), put the rule $A_{pq} \rightarrow aA_{rs}b$ in G.
  - For each $p, q, r \in Q$, put the rule $A_{pq} \rightarrow A_{pr}A_{rq}$ in G.
  - Finally, for each $p \in Q$, put the rule $A_{pp} \rightarrow \varepsilon$ in G.

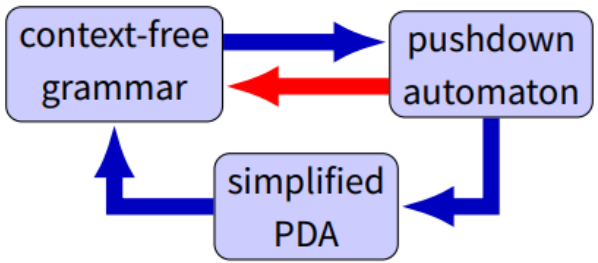


PDA computation corresponding to the rule $A_{pq} \rightarrow A_{pr}A_{rq}$

PDA computation corresponding to the rule $A_{pq} \rightarrow aA_{rs}b$



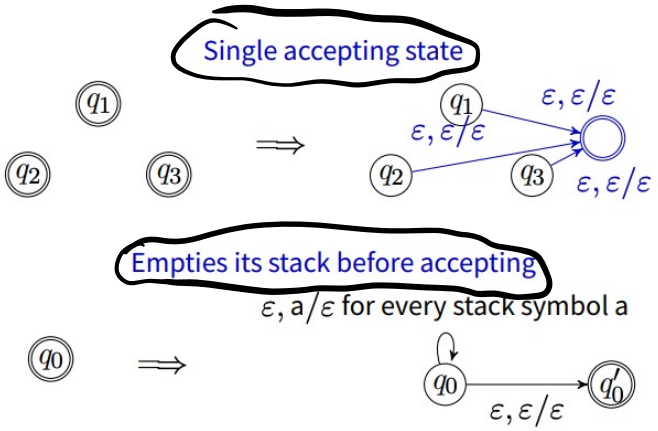
PDA TO CFG

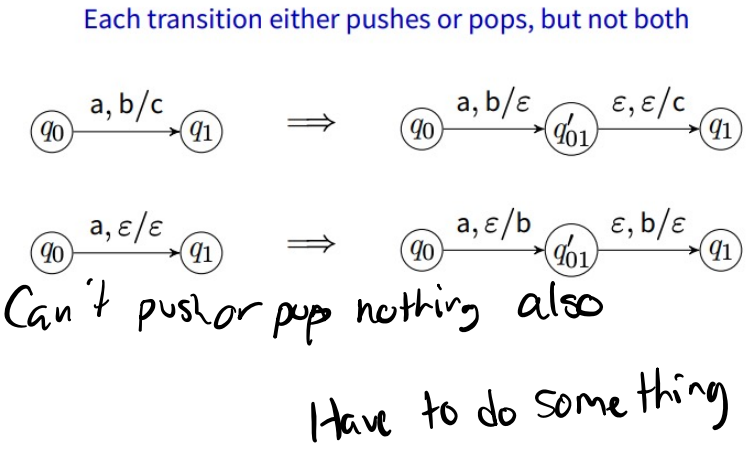


Simplified pushdown automaton:

- Has a single accepting state
- Empties its stack before accepting
- Each transition is either a push, or a pop, but not both

SIMPLIFYING

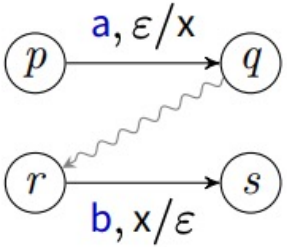




For every pair(q, r) of states in PDA, introduce variable A_{qr} in CFG

Intention: A_{qr} generates all strings that allow the PDA to go from q to r (with empty stack both at q and at r)

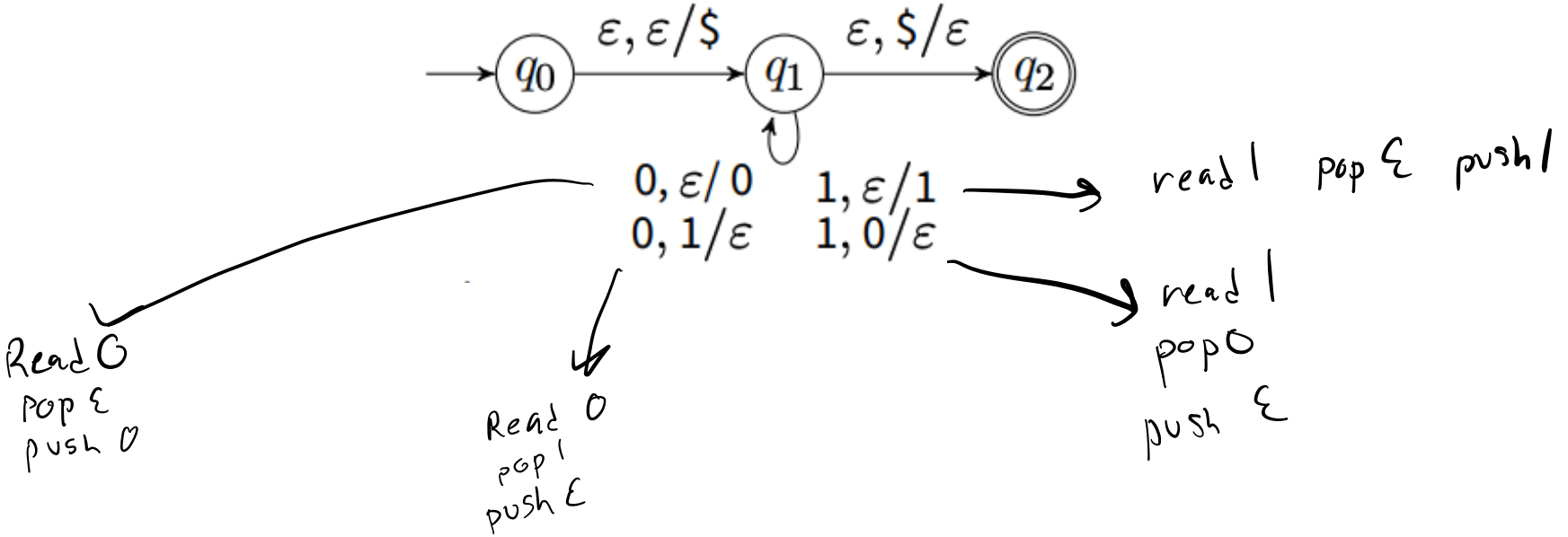
This is not on exam

PDA	CFG
	$A_{qq} \to \varepsilon$
	$A_{pr} \to A_{pq}A_{qr}$
	$A_{ps} \to \mathsf{a}A_{qr}\mathsf{b}$ $\mathsf{a} = \varepsilon$ or $\mathsf{b} = \varepsilon$ allowed

$A_{02} \rightarrow A_{11}$

$A_{11} \rightarrow 0A_{11}1$
$\mid A_{11}0$

EXAMPLE

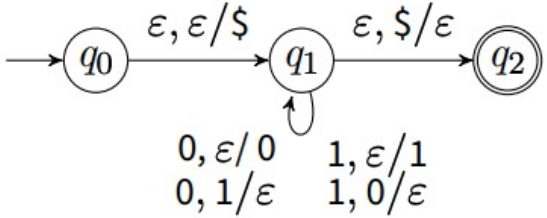


A_{q_0}

start symbol

↓

A_{02}



Every state will have its own variable

variables: $A_{00}, A_{11}, A_{22}, A_{01}, A_{02}, A_{12}$

start variable: A_{02}

A_{00}

A_{11}

A_{22}

A_{01}

A_{12}

A_{02}

productions:

$A_{02} \to A_{01}A_{12}$

$A_{01} \to A_{01}A_{11}$

$A_{12} \to A_{11}A_{12}$

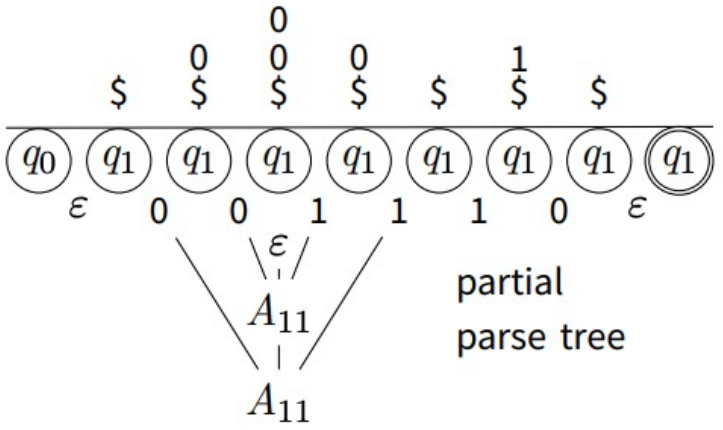
$A_{11} \to A_{11}A_{11}$

$A_{11} \to 0A_{11}1$

$A_{11} \to 1A_{11}0$

$A_{02} \to A_{11}$

$A_{00} \to \varepsilon,\ A_{11} \to \varepsilon,$
$A_{22} \to \varepsilon$



partial parse tree

$A_{00} \to \varepsilon$ $A_{11} \to \varepsilon$ $A_{22} \to \varepsilon$

$A_{02} \to A_{01}A_{12}$ $A_{12} \to A_{12}A_{22}$

$A_{01} \to A_{01}A_{11}$

A_{00}

A_{11}

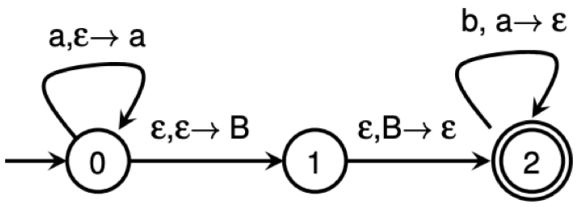
A_{22}

A_{01}

A_{12}

A_{02}

↑ start symbol



1. Single accept state
2. each transition is either pushing or poping
3. Stack is empty at final state

$A_{01} \to a A_{01} \varepsilon$

$A_{12} \to \varepsilon A_{12} b$

$A_{02} \to a A_{02} b$

$A_{11} \to \varepsilon$

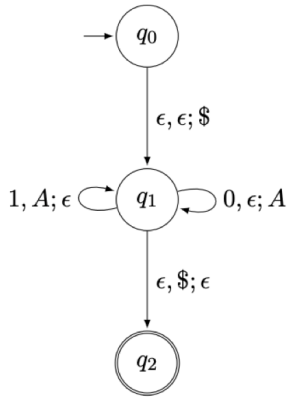
$A_{00} \to \varepsilon$

$A_{22} \to \varepsilon$

$A_{01} \to A_{01} A_{11}$

$A_{02} \to A_{01} A_{12}$

$A_{12} \to$



✓ 1. Single accept state

✓ 2. Each transition is either pushing or poping

✓ 3. Stack is empty at the end

$A_{00} \to \varepsilon$

$A_{11} \to \varepsilon \mid 0 A_{11} 1$

$A_{22} \to \varepsilon$

$A_{02} \to A_{01} A_{12}$

$A_{01} \to A_{01} A_{11}$

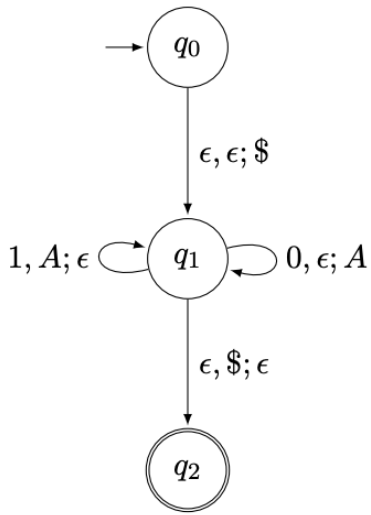
$A_{12} \to A_{11} A_{12}$

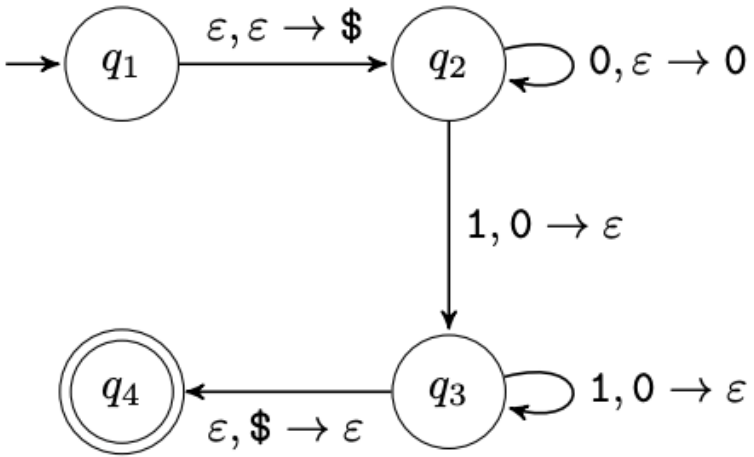
$A_{01} \to \varepsilon A_{11} 0 \mid \varepsilon A_{11} 1$

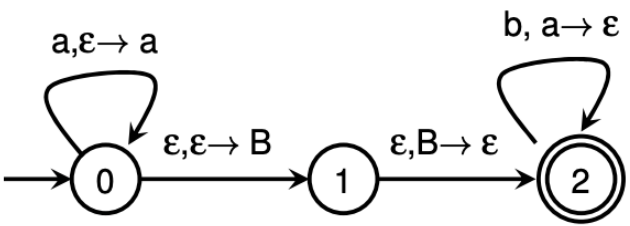
$A_{12} \to 0 A_{12} \varepsilon \mid 1 A_{12} \varepsilon$

$A_{02} \to A_{11}$

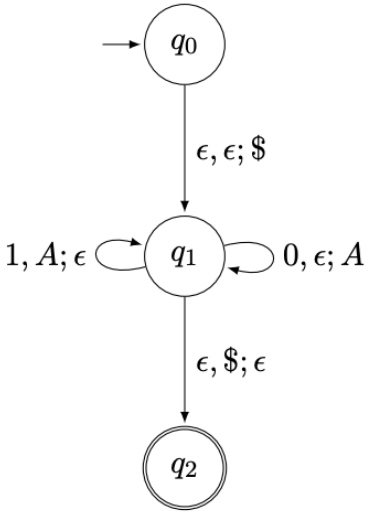
EXAMPLES TO TRY!







EXAMPLE


q_0
$\epsilon, \epsilon; \$$
$1, A; \epsilon$
q_1
$0, \epsilon; A$
$\epsilon, \$; \epsilon$
q_2

end up in q_2.

3. Introduce grammar rules corresponding to moves of the PDA of the form "push, compute, pop":

$$A_{02} \to A_{11}$$
$$A_{11} \to 0A_{11}1$$

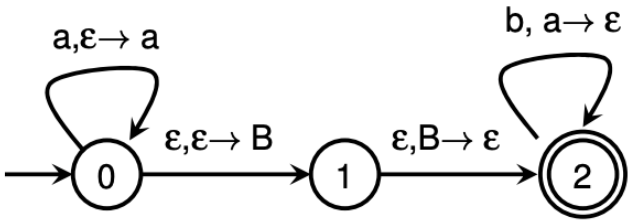
4. Introduce grammar rules corresponding to moves of the PDA which drive it from states p to r and then from r to q:

$$A_{00} \to A_{00}A_{00} \mid A_{01}A_{10} \mid A_{02}A_{20}$$
$$A_{11} \to A_{10}A_{01} \mid A_{11}A_{11} \mid A_{12}A_{21}$$
$$A_{22} \to A_{20}A_{02} \mid A_{21}A_{12} \mid A_{22}A_{22}$$
$$A_{01} \to A_{00}A_{01} \mid A_{01}A_{11} \mid A_{02}A_{21}$$
$$A_{10} \to A_{10}A_{00} \mid A_{11}A_{10} \mid A_{12}A_{20}$$
$$A_{02} \to A_{00}A_{02} \mid A_{01}A_{12} \mid A_{02}A_{22}$$
$$A_{20} \to A_{20}A_{00} \mid A_{21}A_{10} \mid A_{22}A_{20}$$
$$A_{12} \to A_{10}A_{02} \mid A_{11}A_{12} \mid A_{12}A_{22}$$
$$A_{21} \to A_{20}A_{01} \mid A_{21}A_{11} \mid A_{22}A_{21}$$

5. Introduce grammar rules for ϵ:

$$A_{00} \to \epsilon$$
$$A_{11} \to \epsilon$$
$$A_{22} \to \epsilon$$

EXAMPLE



REGULAR ⇒ CFL

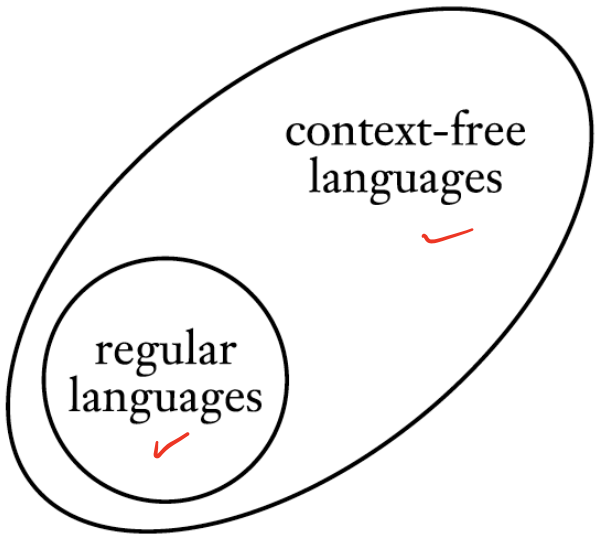
- If A is a regular language, then A is also a CFL

Proof

- Suppose A is regular.
- implies A has an NFA.
- But an NFA is just a PDA that ignores the stack.
- So A has a PDA.
- Implies A is context-free

Note:

- Converse is not true.
- For example, $\{0^n1^n \mid n \geq 0\}$ is CFL but not regular.



PUMPING LEMMA FOR CFLs

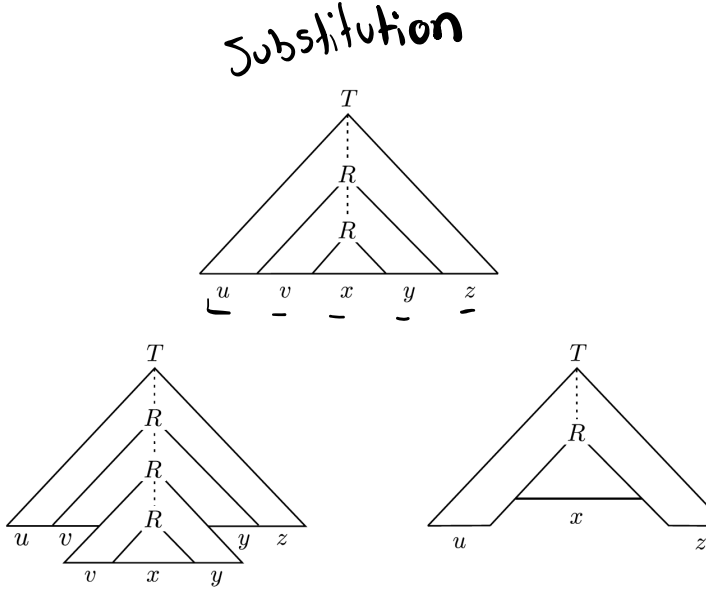
Theorem 2.34

- If A is context-free language, then there is a pumping length p where, if $s \in A$ with $|s| \geq p$, then s can be split into 5 pieces $s = uvxyz$ satisfying the conditions
  - $uv^ixy^iz \in A$ for each $i \geq 0$,
  - $|vy| > 0$, and
  - $|vxy| \leq p$.
- Remarks:
  - Condition 1 implies that $uxz \in A$ by taking $i = 0$.
  - Condition 2 says that vy cannot be the empty string.
  - Condition 3 states that the pieces v, x, and y together have length at most p and is sometimes useful.

$a^n b^n$

$a^n b^n c^n$ ✗ CFG
non CFG

PUMPING LEMMA FOR CFLs



- Let A be a CFL and let G be a CFG that generates it. We must show that any sufficiently long string s in A can be pumped and remain in A.
- Let s be a very long string in A. Because s is in A, it is derivable from G and so has a parse tree.
- The parse tree for s must be very tall because s is very long. That is, the parse tree must contain some long path from the start variable at the root of the tree to one of the terminal symbols at a leaf.
- On this long path, some variable symbol R must repeat.
- This repetition allows us to replace the subtree under the second occurrence of R with the subtree under the first occurrence of R and still get a parse tree.
- We may cut s into five pieces uvxyz, and we may repeat the second and fourth pieces and obtain a string still in the language.
- uv^ixy^iz is in A for any $i \geq 0$.

PUMPING LEMMA FOR CFLS

- Previously saw pumping lemma for regular languages.
- Analogous result for context-free language A.
- Basic Idea: Derivation of long string s ∈ A has repeated variable R.
  - Long string implies tall parse tree, so must have repeated variable.
  - Can split string s ∈ A into 5 pieces s = uvxyz based on R.
  - $uv^ixy^iz \in A$ for all $i \geq 0$.
- Consider language A with CFG G

  S → CDa | CD

  C → aD

  D → Sb | b
- Below "long" derivation using G repeats variable R = D:

  S ⇒ CDa ⇒ aDDa ⇒ abDa ⇒ abSba ⇒ abCDba ⇒ abaDDba ⇒ ababDba ⇒ ababbba

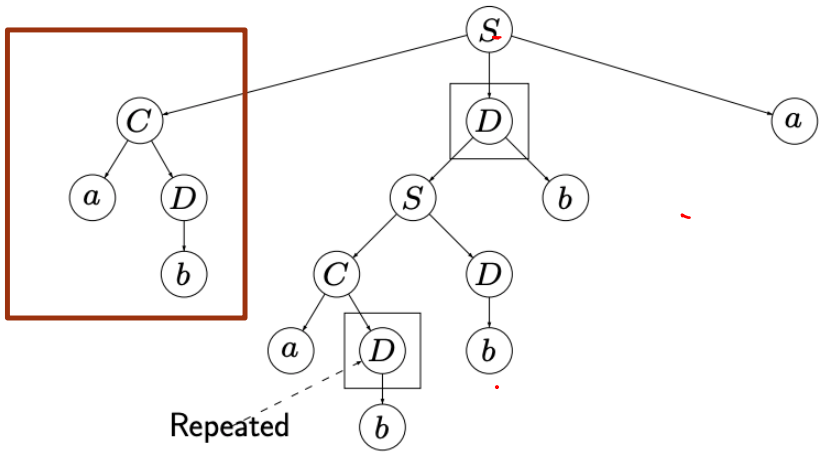
REPEATED VARIABLE IN PATH OF PARSE TREE

$S \to CDa \mid CD$
$C \to aD$
$D \to Sb \mid b$

- Derivation of "long" string s = ababbba ∈ A repeats variable D:

$$S \Rightarrow CDa \Rightarrow aDDa \Rightarrow abDa \Rightarrow abSba \Rightarrow abCDba \Rightarrow abaDDba \Rightarrow ababDba \Rightarrow ababbba$$

- "Tall" parse tree repeats variable D on path from root to leaf.



SPLIT STRING INTO 5 PIECES

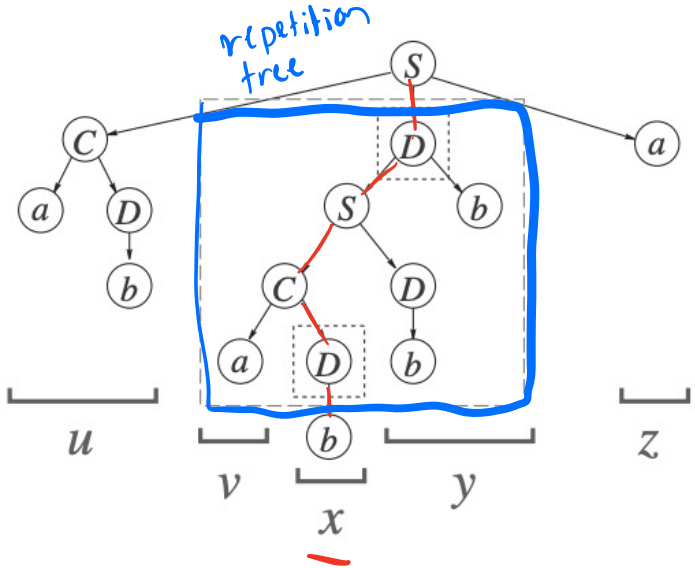
- Split string s ∈ A into

$$s = \underbrace{ab}_{u}\,\underbrace{a}_{v}\,\underbrace{b}_{x}\,\underbrace{bb}_{y}\,\underbrace{a}_{z}$$

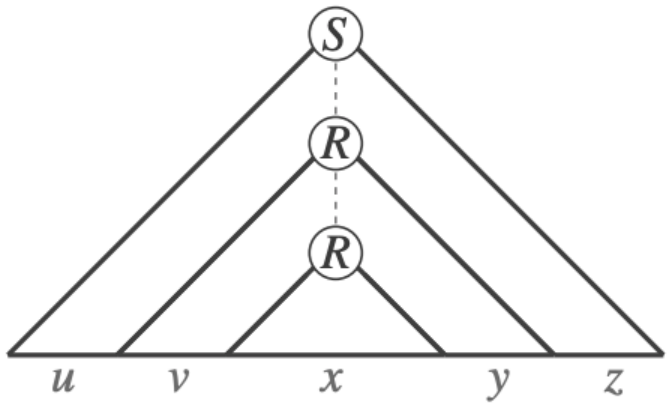
using repeated variable D.

- In depth-first traversal of tree
  - u = ab is before D-D subtree
  - v = a is before second D within D-D subtree
  - x = b is what second D eventually becomes
  - y = bb is after second D within D-D subtree
  - z = a is after D-D subtree

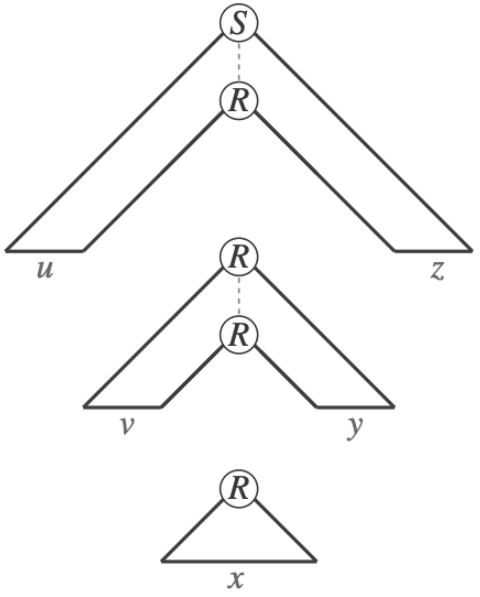
First find the longest one - x



- More generally, consider "long" string s ∈ A.
- Parse tree is "tall"
  - ∃ repeated variable R in path from root S to leaf.
- Split string s = uvxyz into 5 pieces based on repeated variable R:
  - u is before R-R subtree (in depth-first order)
  - v is before second R within R-R subtree
  - x is what second R eventually becomes
  - y is after second R within R-R subtree
  - z is after R-R subtree



SUBTREES YIELD . . .



$S \overset{*}{\Rightarrow} uRz$

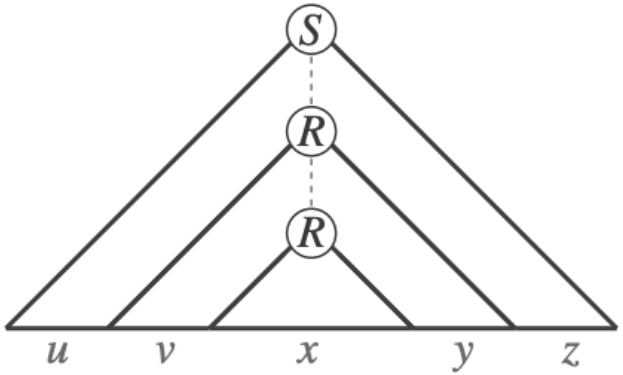
$R \overset{*}{\Rightarrow} vRy$

$R \overset{*}{\Rightarrow} x$

CAN PUMP TO OBTAIN OTHER STRINGS IN A

- Parse tree for string s ∈ A implies

$$S \overset{*}{\Rightarrow} uRz \quad \text{for } u, z \in \Sigma^*$$
$$R \overset{*}{\Rightarrow} vRy \quad \text{for } v, y \in \Sigma^*$$
$$R \overset{*}{\Rightarrow} x \quad \text{for } x \in \Sigma^*$$



- Can derive string s = uvxyz ∈ A

$$S \overset{*}{\Rightarrow} uRz \overset{*}{\Rightarrow} uvRyz \overset{*}{\Rightarrow} uvxyz \in A$$

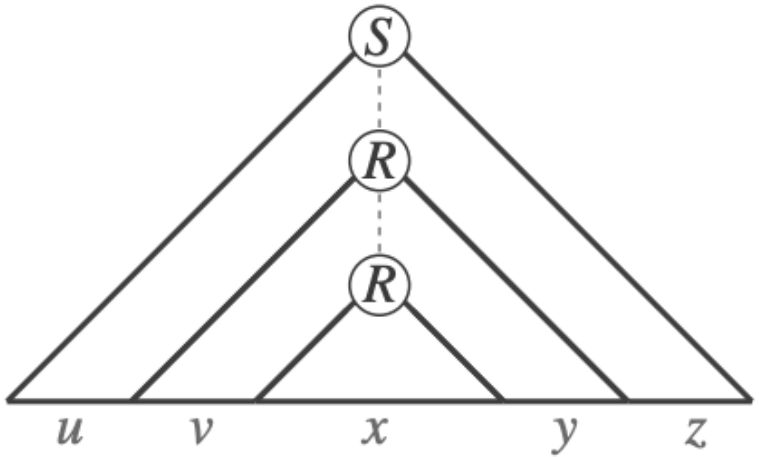
- Also for each i ≥ 0, can derive string

$$S \overset{*}{\Rightarrow} uRz \overset{*}{\Rightarrow} uvRyz \overset{*}{\Rightarrow} uvvRyyz \overset{*}{\Rightarrow} \cdots \overset{*}{\Rightarrow} uv^iRy^iz \overset{*}{\Rightarrow} uv^ixy^iz \in A$$

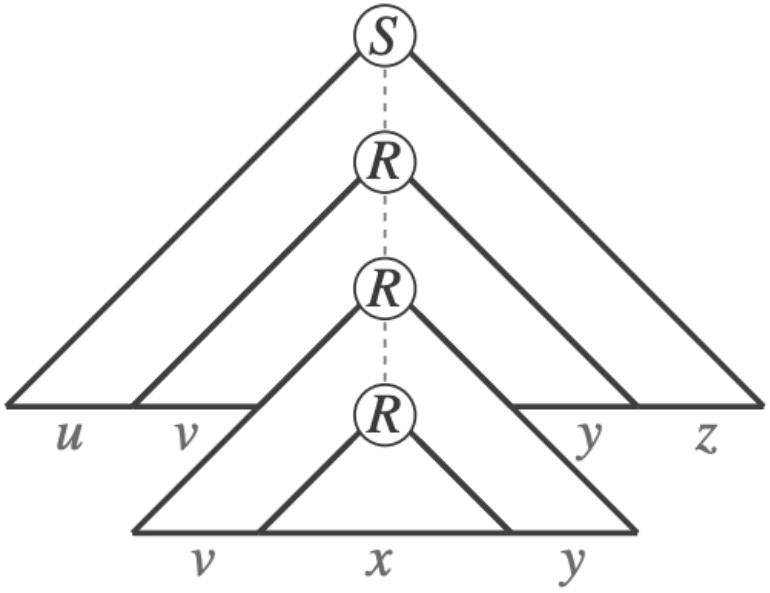
PUMPING A PARSE TREE

Consider parse tree of uvwxy ∈ A

$$S \overset{*}{\Rightarrow} uRz \overset{*}{\Rightarrow} uvRyz \overset{*}{\Rightarrow} uvxyz \in A$$



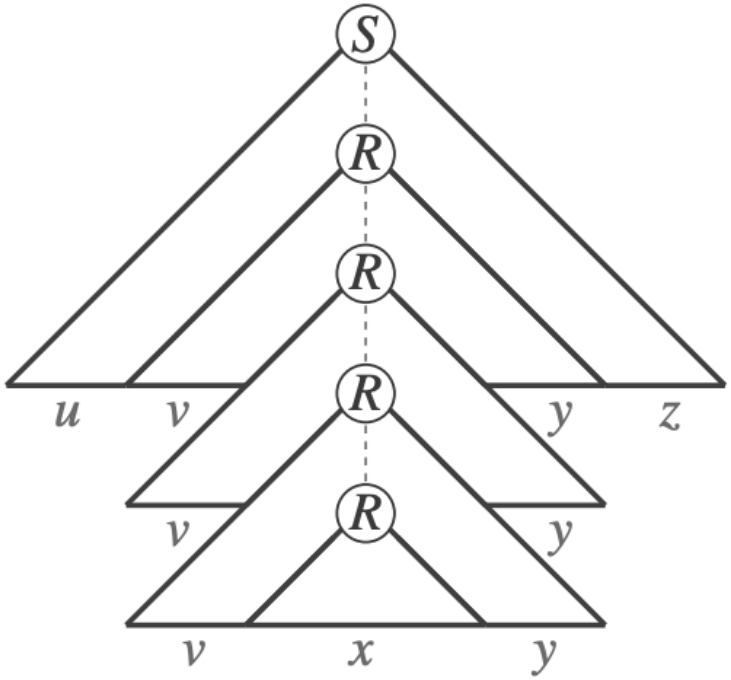
PUMPING UP A PARSE TREE



Recall: $S \overset{*}{\Rightarrow} uRz \overset{*}{\Rightarrow} uvRyz \overset{*}{\Rightarrow} uvxyz \in A$

Using R-R subtree twice shows uvvxyyz = $uv^2xy^2z \in A$

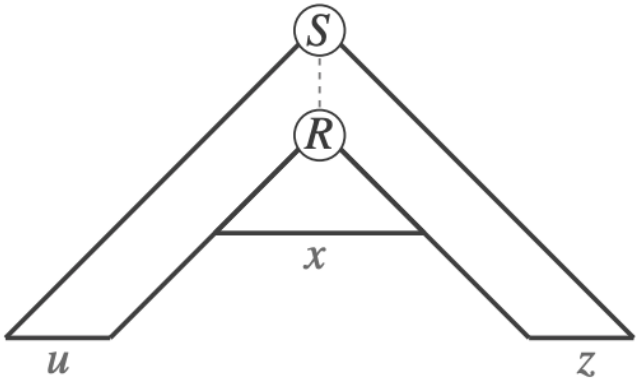
PUMPING UP MULTIPLE TIMES



Recall: $S \overset{*}{\Rightarrow} uRz \overset{*}{\Rightarrow} uvRyz \overset{*}{\Rightarrow} uvxyz \in A$

Using R-R subtree thrice shows $uv^3xy^3z \in A$

PUMPING DOWN A PARSE TREE



Recall: $S \overset{*}{\Rightarrow} uRz \overset{*}{\Rightarrow} uvRyz \overset{*}{\Rightarrow} uvxyz \in A$

Removing R-R subtree shows $uxz = uv^0xy^0z \in A$

WHEN IS PUMPING POSSIBLE?

- Key to Pumping: repeated variable R in parse tree.

  $S \overset{*}{\Rightarrow} uRz$ for $u, z \in \Sigma^*$
  $R \overset{*}{\Rightarrow} vRy$ for $v, y \in \Sigma^*$
  $R \overset{*}{\Rightarrow} x$ for $x \in \Sigma^*$
  string $s = uvxyz \in A$

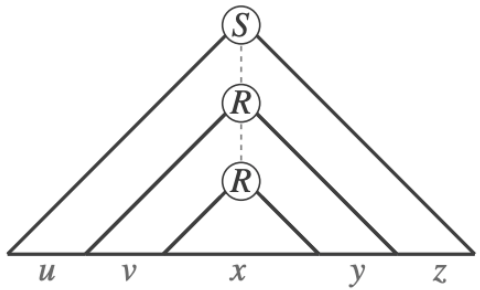
- Repeated variable $R \overset{*}{\Rightarrow} vRy$, so "v-y pumping" possible:

$$S \overset{*}{\Rightarrow} uRz \overset{*}{\Rightarrow} uvRyz \overset{*}{\Rightarrow} uv^iRy^iz \overset{*}{\Rightarrow} uv^ixy^iz \in A$$

- If tree is tall enough, then repeated variable in path from root to leaf.
  - CFG has finite number |V | of variables.
  - How tall does parse tree have to be to ensure pumping possible?
  - Length of path between two nodes = # edges in path.
  - Tree height = # edges on longest path from root to a leaf.

CAN PUMP IF PARSE TREE IS TALL ENOUGH

- Path from root S to leaf
  - Leaf is a terminal $\in \Sigma$
  - All other nodes along path are variables $\in V$.
- If height of tree $\geq |V| + 1$, where $|V|$ = # variables in CFG
  - then $\exists$ repeated variable on longest path from root to leaf.
- How long does string $s \in A$ have to be to ensure tall enough tree?



PREVIOUS EXAMPLE

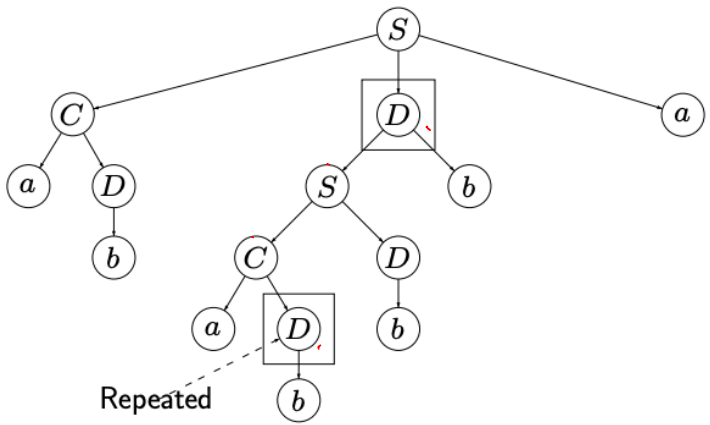
- |V | = 3 variables in below CFG:

  $S \rightarrow CDa \mid CD$

  $C \rightarrow aD$

  $D \rightarrow Sb \mid b$

- In parse tree for ababbba, longest path has length 5 ≥ |V | +1=4



IF STRING s IS LONG ENOUGH, THEN CAN PUMP

- Let A have CFG in which longest rule has right-side length $b \geq 2$:

$$C \rightarrow D_1 \cdots D_b$$

  - So each node in tree has $\leq b$ children.
  - At most b leaves one step from root.
  - At most b^2 leaves 2 steps from root, and so on.
  - If tree has height $\leq h$,
    - then $\leq b^h$ leaves, so generated string s has length $|s| \leq b^h$.
  - Equiv: If string $s \in A$ has $|s| \geq b^h + 1$, then tree height $\geq h + 1$.
- Let $|V|$ = # variables in CFG.
- If string $s \in A$ has length $|s| \geq p \equiv b^{|V|+1}$, then
  - tree height $\geq |V| + 1$ because $b^{|V|+1} \geq b^{|V|} + 1$.
  - some variable on longest path in tree is repeated
  - can pump parse tree

PUMPING LEMMA FOR CFLs

Theorem 2.34

- If A is context-free language, then ∃ pumping length p where, if s ∈ A with $|s| \geq p$, then s can be split into 5 pieces s = uvxyz satisfying the conditions
  - $uv^ixy^iz \in A$ for each $i \geq 0$,
  - $|vy| > 0$, and
  - $|vxy| \leq p$.
- Remarks:
  - Condition 1 implies that uxz ∈ A by taking i = 0.
  - Condition 2 says that vy cannot be the empty string.
  - Condition 3 states that the pieces v, x, and y together have length at most p and is sometimes useful.

NON CFL

- Remark: CFL Pumping Lemma (PL) mainly used to show certain languages are ***not*** CFL.
- Example: Prove that B = { $a^n b^n c^n \mid n \geq 0$ } is non-CFL.
- Proof:
  - Suppose B is CFL, it implies that B has pumping length $p \geq 1$.
  - Consider string $s = a^p b^p c^p \in B$, so $|s| = 3p \geq p$.
  - PL: can split s into 5 pieces $s = uvxyz = a^p b^p c^p$ satisfying
    - $uv^i xy^i z \in B$ for all $i \geq 0$ → 1
    - $|vy| > 0$ → 2
    - $|vxy| \leq p$ → 3
  - For contradiction, show cannot split s = uvxyz satisfying 1–3.
    - Show every possible split satisfying Condition 2 violates Condition 1.

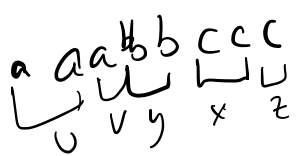
$p \geq 2$ $a^p b^p c^p$

aaabbcc

$|v| \geq p$

$6 \geq 2$

- Recall s = uvxyz = $\underbrace{aa\ldots.a}_{p}\ \underbrace{bb\ldots.b}_{p}\ \underbrace{cc\ldots\ldots c}_{p}$
- Possibilities for split s = uvxyz satisfying Condition 2: $|vy| > 0$
  - (i) Strings v and y are uniform [e.g., v = a ··· a and y = b ··· b].
    - Then uv^2xy^2z won't have same number of a's, b's and c's because $|vy| > 0$.
    - Hence, $uv^2xy^2z \notin B$.
  - (ii) Strings v and y are not both uniform [e.g., v = a ··· ab ··· b and y = b ··· b].
    - Then $uv^2xy^2z \notin L(a^*b^*c^*)$: symbols not grouped together.
    - Hence, $uv^2xy^2z \notin B$.
- Thus, every split satisfying Condition 2 has $uv^2xy^2z \notin B$, so Condition 1 violated.
- **Contradiction,** so $B = \{ a^nb^nc^n \mid n \geq 0 \}$ is not a CFL.



uv^2xy^2z

aa aabb bbcccc ∉A

This language is not context free

Prove C = { $a^i b^j c^k$ | $0 \le i \le j \le k$ } is not CFL

- Suppose C is CFL, so PL implies C has pumping length p.
- Take string s = $\underbrace{aa.....a}_{p}\ \underbrace{bb.....b}_{p}\ \underbrace{cc.....c}_{p}$ ∈ C, so |s| = 3p ≥ p.
- PL: can split s = $a^p b^p c^p$ into 5 pieces s = uvxyz satisfying
  1. $uv^i xy^i z$ ∈ C for every i ≥ 0,
  2. |vy| > 0,
  3. |vxy| ≤ p.
- Condition 3 implies vxy can't contain 3 different types of symbols.
- Two possibilities for v, x, y satisfying |vy| > 0 and |vxy| ≤ p:
  I. If vxy ∈ L(a*b*), then z has all the c's
     I. string uv^2xy^2z has too few c's because z not pumped
     II. Hence, $uv^2xy^2z \notin C$
  II. If vxy ∈ L(b*c*), then u has all the a's
     I. string $uv^0xy^0z = uxz$ has too many a's
     II. Hence, $uv^0xy^0z \notin C$
- Every split s = uvxyz satisfying 2–3 violates 1, so C isn't CFL.

Prove D = { ww | w ∈ {0, 1}* } is not CFL

- Suppose D is CFL, so PL implies D has pumping length p. •
- Take s = $\underbrace{00\ldots.0}_{p}\ \underbrace{11\ldots..1}_{p}\ \underbrace{00\ldots\ldots0}_{p}\ \underbrace{11\ldots..1}_{p}$ ∈ D, so |s| = 4p ≥ p.
- PL: can split s into 5 pieces s = uvxyz satisfying
  1. uv^ixy^iz ∈ D for every i ≥ 0,
  2. |vy| > 0,
  3. |vxy| ≤ p.

I. If vxy is entirely left of middle of $0^p\ 1^p\ 0^p\ 1^p$,
- then second half of uv^2xy^2z starts with a 1
- so can't write uv^2xy^2z as ww because first half starts with 0.

II. Similar reasoning: if vxy is entirely right of middle of $0^p\ 1^p\ 0^p\ 1^p$,
- then $uv^2xy^2z \notin D$

III. If vxy straddles middle of $0^p\ 1^p\ 0^p\ 1^p$,
- then $uv^0xy^0z = uxz = 0^p\ 1^j\ 0^k\ 1^p \notin D$ (because j or k < p)

- Every split s = uvxyz satisfying 2–3 violates 1, so D isn't CFL.

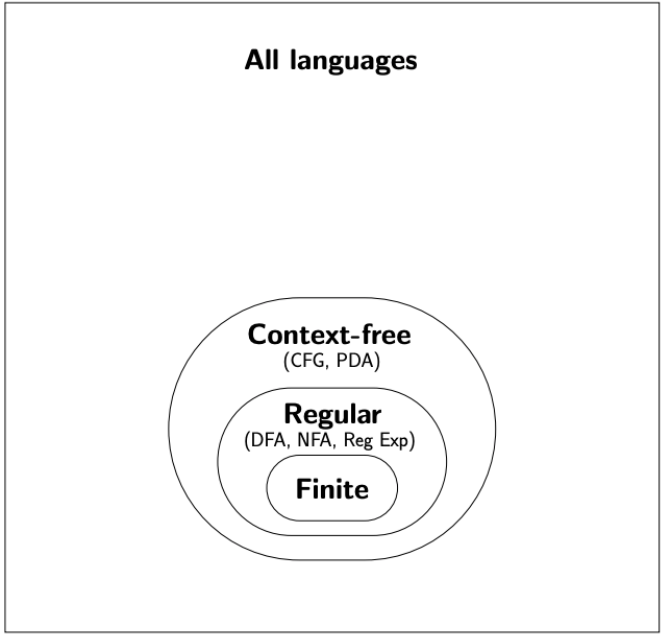
REMARKS ON CFL PUMPING LEMMA

Often more difficult to apply CFL pumping lemma (Theorem 2.34)

than pumping lemma for regular languages (Theorem 1.70).

- Carefully choose string s in language to get contradiction.
  - Not all strings s will give contradiction.
- CFL pumping lemma: "... can split s into 5 pieces s = uvxyz satisfying all of Conditions 1–3."
- To get contradiction, must show cannot split s into 5 pieces s = uvxyz satisfying all of Conditions 1–3.
  - Need to show every possible split s = uvxyz violates at least one of Conditions 1–3.

HIERARCHY OF LANGUAGES (SO FAR)



Examples

$\{ 0^n 1^n 2^n \mid n \geq 0 \}$

$\{ 0^n 1^n \mid n \geq 0 \}$

$(0 \cup 1)^*$

$\{ 110, 01 \}$

TURING MACHINE

OVERVIEW

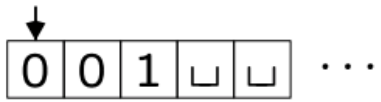
- Turing Machines
- Turing-recognizable
- Turing-decidable
- Variants of Turing Machines
- Algorithms
- Encoding input for TM

REVIEW

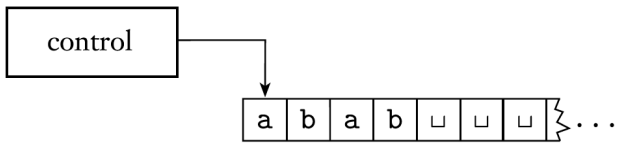
- DFA
  - Reads input from left to right
  - Finite control (i.e., transition function) based on
    - current state,
    - current input symbol read.
- PDA
  - Has stack for extra memory
  - Reads input from left to right
  - Can read/write to memory (stack) by popping/pushing
  - Finite control based on
    - current state,
    - what's read from input,
    - what's popped from stack

TURING MACHINE (TM)

- Infinitely long tape, divided into cells, for memory
- Tape initially contains input string followed by all blanks



- Tape head (↓) can move both right and left
- Can read from and write to tape
- Finite control based on
  - current state,
  - current symbol that head reads from tape.



- Machine has one accept state and one reject state.
- Machine can run forever: infinite loop.
- If it doesn't enter an accepting or a rejecting state, it will go on forever, never halting.

KEY DIFFERENCE BETWEEN TMS AND PREVIOUS MACHINES

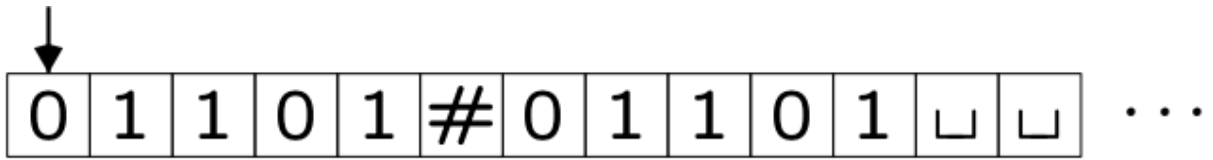
- Turing machine can both read from tape and write on it.
- Tape head can move both right and left.
- Tape is infinite and can be used for storage.
- Accept and reject states take immediate effect.

Example: Machine for recognizing language

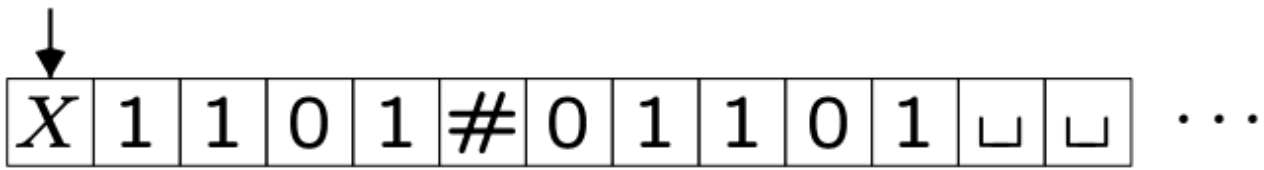
$$A = \{ s\#s \mid s \in \{0, 1\}^* \}$$

Idea: Zig-zag across tape, crossing off matching symbols.

- Consider string 01101#01101 ∈ A.
- Tape head starts over leftmost symbol

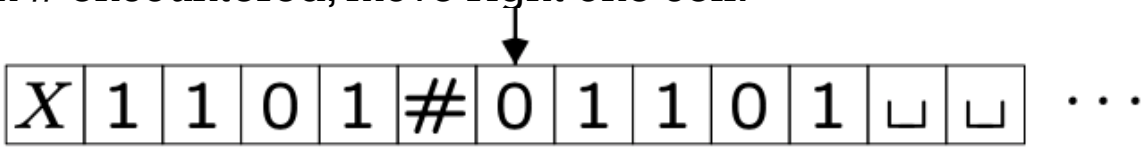


- Record symbol in control and overwrite it with X

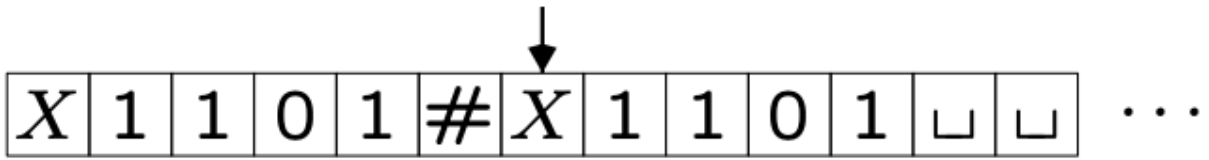


- Scan right: reject if blank "␣" encountered before #

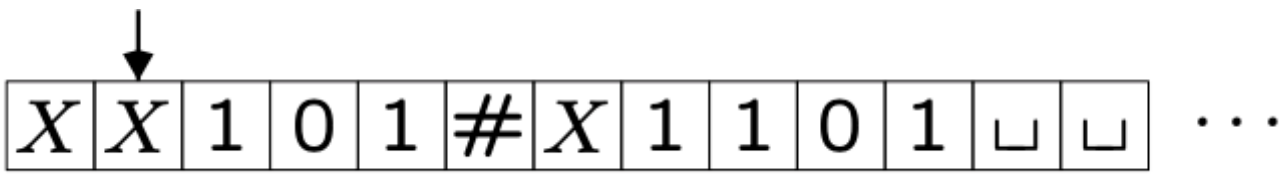
- When # encountered, move right one cell.



- If current symbol doesn't match previously recorded symbol, reject.
- Overwrite current symbol with X

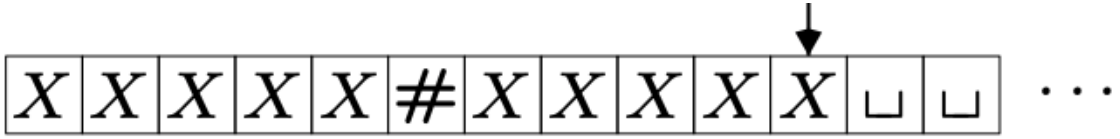


- Scan left, past # to X
- Move one cell right
- Record symbol and overwrite it with X

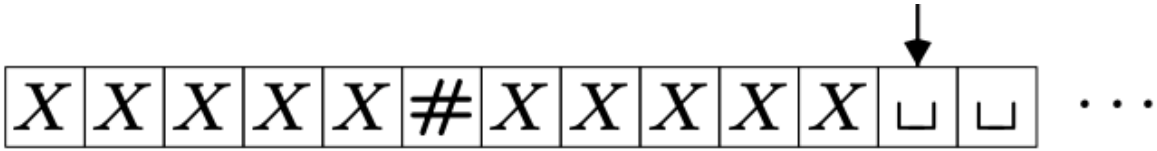


- Scan right past # to (last) X and move one cell to right . . .

- After several more iterations of zigzagging, we have



- After all symbols left of # have been matched to symbols right of #, check for any remaining symbols to the right of #.
  - If blank ␣ encountered, accept.
  - If 0 or 1 encountered, reject.



- The string that is accepted or not by our machine is the original input string 01101#01101.

Description of TM M_1 for $\{ s\#s \mid s \in \{0, 1\}^* \}$

M_1 = "On input string w:

1. Scan input to be sure that it contains a single #. If not, reject.
2. Zig-zag across tape to corresponding positions on either side of the # to check whether these positions contain the same symbol. If they do not, reject. Cross off symbols as they are checked off to keep track of which symbols correspond.
3. When all symbols to the left of # have been crossed off along with the corresponding symbols to the right of #, check for any remaining symbols to the right of the #. If any symbols remain, reject; otherwise, accept."

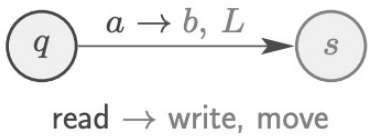
FORMAL DEFINITION OF TURING MACHINE

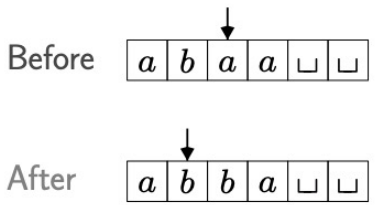
Definition: A Turing machine (TM) is a 7-tuple $M = (Q, \Sigma, \Gamma, \delta, q_0, q_{accept}, q_{reject})$, where

1. Q is a finite set of states
2. Σ is the input alphabet not containing blank symbol $\sqcup$
3. Γ is tape alphabet with blank $\sqcup \in \Gamma$ and $\Sigma \subseteq \Gamma$
4. $\delta : Q \times \Gamma \rightarrow Q \times \Gamma \times \{L, R\}$ is the transition function, where L means move tape head one cell to left R means move tape head one cell to right
5. $q_0 \in Q$ is the start state
6. $q_{accept} \in Q$ is the accept state
7. $q_{reject} \in Q$ is the reject state, with $q_{reject} \neq q_{accept}$

TRANSITION FUNCTION OF TM

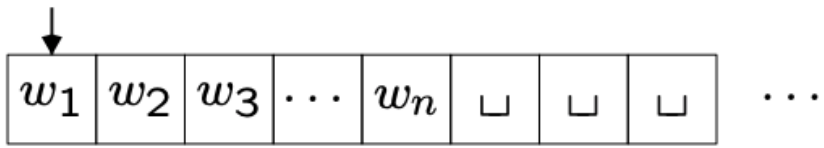
- Transition function $\delta : Q \times \Gamma \rightarrow Q \times \Gamma \times \{L, R\}$
- $\delta(q, a)=(s, b, L)$ means
  - if TM
    - in state $q \in Q$, and
    - tape head reads tape symbol $a \in \Gamma$,
  - then TM
    - moves to state $s \in Q$
    - overwrites a with $b \in \Gamma$
    - moves head left (i.e., $L \in \{L, R\}$)





START OF TM COMPUTATION

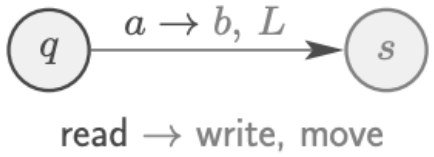
- A TM, M = (Q, Σ, Γ, δ, q0, q_{accept}, q_{reject}) begins computation as follows:
- Given input string w = $w_1 w_2 \cdots w_n \in \Sigma^*$ with each $w_i \in \Sigma$, i.e., w is a string of length n for some $n \geq 0$.
- TM begins in start state q_0
- Input string is on n leftmost tape cells



- Rest of tape contains blanks ⊔
- Head starts on leftmost cell of tape
- Because ⊔ ∉ Σ, first blank denotes end of input string.
- The computation continues until it enters either the accept or reject states, at which point it halts. If neither occurs, M goes on forever.

- When computation on TM M = ($Q, \Sigma, \Gamma, \delta, q0, q_{accept}, q_{reject}$) starts,
- TM M proceeds according to transition function

$$\delta : Q \times \Gamma \rightarrow Q \times \Gamma \times \{L, R\}$$



- If M tries to move head off left end of tape, then head remains on first cell.
- Computation continues until q_{accept} or q_{reject} is entered.
- Otherwise, M runs forever: infinite loop.
- In this case, input string is neither accepted nor rejected.
- As TM computes changes occurs in current state, the current tape contents, and the current head location, these items altogether is called configuration.

OPERATIONS ON THE TAPE

- Read or scan symbol below the tape head
- Update/write symbol below the tape head
- Move tape head one step Left ‘L’
- Move tape head one step Right ‘R’

UNDERSTANDING THE WAY, A TM COMPUTES

Configuration C_1 yields configuration C_2 if the TM can legally go from C_1 to C_2 in a single step.

- Definition:
  - Suppose that we have a, b, and c in Γ, as well as u and v in Γ^* and states q_i and q_j.
    - In that case, ua q_i bv and u q_j acv are two configurations.

      ua $q_i$bv *yields* u q_j acv
  - if in the transition function $\delta(q_i, b) = (q_j, c, L)$. That handles the case where the Turing machine moves leftward.
  - For a rightward move, say that ua q_i bv *yields* uac q_j v
  - if δ(qi, b) = (qj, c,R). Special cases occur when the head is at one of the ends of the configuration.
  - For the left-hand end, the configuration q_i bv *yields* q_j cv if the transition is left moving. It yields c q_j v for the right-moving transition.
  - For the right-hand end, the configuration uaq_i is equivalent to uaq_i ␣ because we assume that blanks follow the part of the tape represented in the configuration.

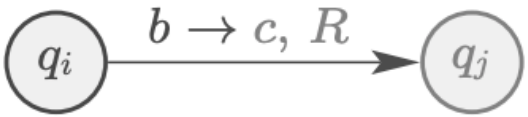
Configuration C_1 yields configuration C_2 if the Turing machine can legally go from C1 to C2 in a single step.

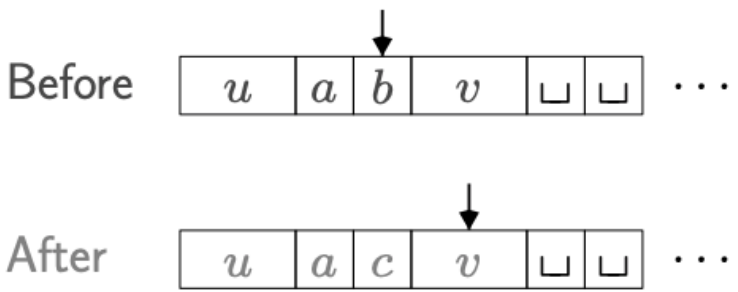
Specifically, for TM $M = (\Sigma, \Gamma, \delta, q0, q_{accept}, q_{reject})$, suppose

- $u, v \in \Gamma^*$
- $a, b, c \in \Gamma$
- $q_i, q_j \in Q$
- transition function $\delta : Q \times \Gamma \rightarrow Q \times \Gamma \times \{L, R\}$.

- Then configuration uaq_ibv yields configuration $uacq_jv$ if

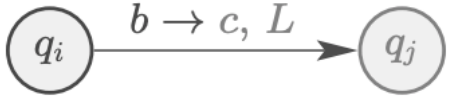
$$\delta(q_i, b) = (q_j, c, R).$$

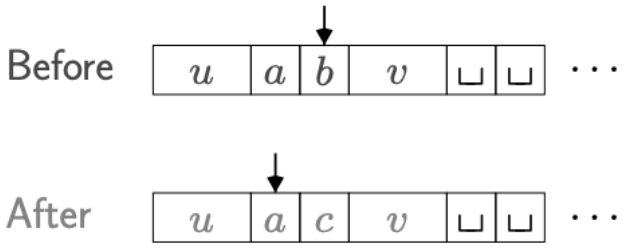




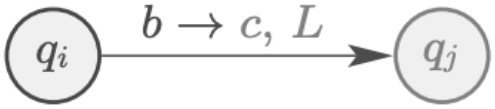
- Similarly, configuration uaq_ibv yields configuration uq_jacv if

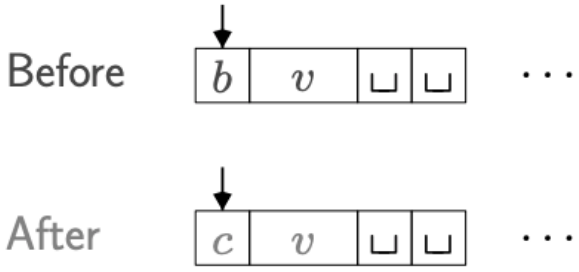
$$\delta(q_i, b) = (q_j, c, L).$$





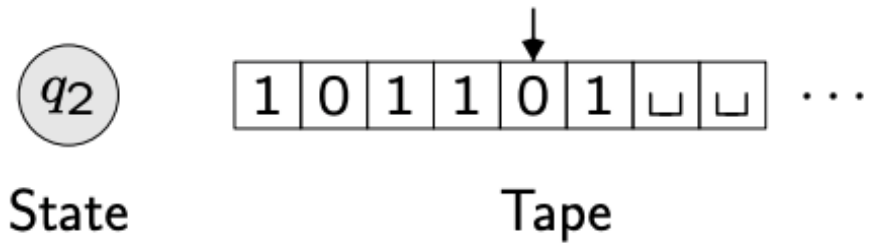
- Special case: $q_i bv$ yields $q_j cv$ if $\delta(q_i, b)=(q_j, c, L)$
- If head is on leftmost cell of tape and tries to move left, then it stays in same place.





TM CONFIGURATIONS

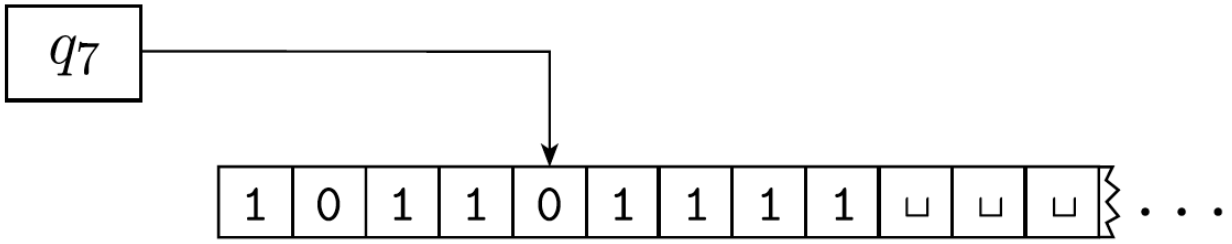
- Computation changes
  - current state
  - current head position
  - tape contents State



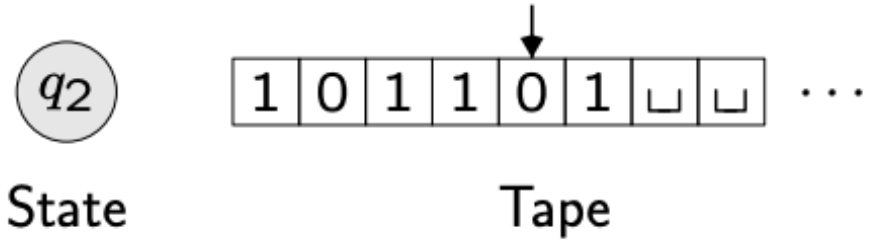
- Configuration provides "snapshot" of TM at any point during computation:
  - current state $q \in Q$
  - current tape contents $\in \Gamma^*$
  - current head location

CONFIGURATIONS

- For a state q and two strings u and v over the tape alphabet Γ, we write u q v for the configuration where the current state is q, the current tape contents is uv, and the current head location is the first symbol of v.
- The tape contains only blanks following the last symbol of v.
- For example, 1011q701111 represents the configuration when the tape is 10110111, the current state is q7, and the head is currently on the second 0.



- Configuration 1011q201 means
  - current state is q2
  - LHS of tape is 1011
  - RHS of tape is 01
  - head is on RHS 0



- Definition: a configuration of a TM $M = (Q, \Sigma, \Gamma, \delta, q_0, q_{accept}, q_{reject})$ is a string uqv with $u, v \in \Gamma*$ and $q \in Q$, and specifies that currently
  - M is in state q
  - tape contains uv
  - tape head is pointing to the cell containing the first symbol in v

CONFIGURATIONS

Consider TM M = (Q, Σ, Γ, δ, q_0, q_{accept}, q_{reject}).

- Starting configuration on input w $\in \Sigma^*$ is q_0w
- An accepting configuration is $uq_{accept}v$ for some u, v $\in \Gamma^*$
- A rejecting configuration is $uq_{reject}v$ for some u, v $\in \Gamma^*$
- Accepting and rejecting configurations are halting configurations.
- Configuration wq_i is the same as $wq_i\sqcup$

TERMINOLOGIES RECALL

- The collection of strings that M accepts is the language of M, or the language recognized by M, denoted L(M)
- Call a language Turing-recognizable if some TM recognizes it.
- When we start a TM on an input, 3 outcomes are possible. The machine may accept, reject, or loop.
  - loop - the machine does not halt.
- A TM M can fail to accept an input by entering the q_{reject} state and rejecting, or by looping.
- TMs that halt on all inputs, such machines never loop. These machines are called deciders because they always make a decision to accept or reject.
- A decider that recognizes some language also is said to decide that language.
- Call a language Turing-decidable or decidable if some TM decides it.