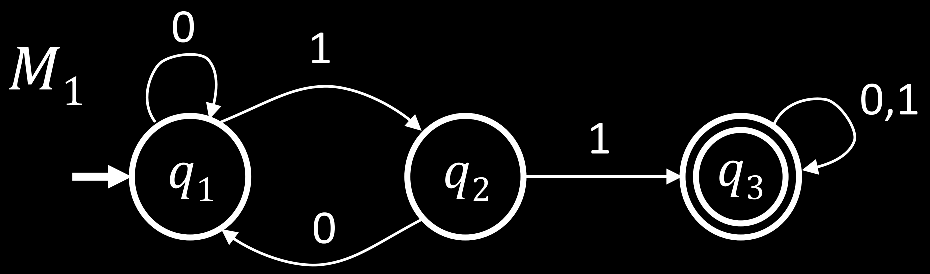
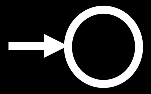
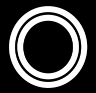
Let's begin: Finite Automata



States: $q_1\ q_2\ q_3$

Transitions: 

Start state: 

Accept states: 

Input: finite string
Output: Accept or Reject

Computation process: Begin at start state, read input symbols, follow corresponding transitions, Accept if end with accept state, Reject if not.

Examples: 01101 → Accept
00101 → Reject

M_1 accepts exactly those strings in A where
$A = \{w \mid w \text{ contains substring } 11\}$.

Say that A is the language of M_1 and that M_1 recognizes A and that $A = L(M_1)$.

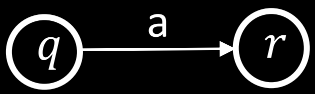
Finite Automata – Formal Definition

Defn: A <u>finite automaton</u> M is a 5-tuple $(Q, \Sigma, \delta, q_0, F)$

Q finite set of states

Σ finite set of alphabet symbols

δ transition function $\delta\colon Q \times \Sigma \rightarrow Q$

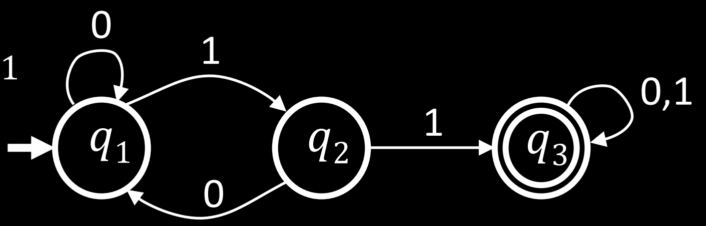
$\delta\,(q,\ a) = r$ means 

q_0 start state

F set of accept states

Example:

M_1



$M_1 = (Q, \Sigma, \delta, q_1, F)$

$Q = \{q_1, q_2, q_3\}$

$\Sigma = \{0, 1\}$

$F = \{q_3\}$

$\delta =$

	0	1
q_1	q_1	q_2
q_2	q_1	q_3
q_3	q_3	q_3

Finite Automata – Computation

Strings and languages

- A string is a finite sequence of symbols in Σ
- A language is a set of strings (finite or infinite)
- The empty string ε is the string of length 0
- The empty language $\emptyset$ is the set with no strings

Defn: M accepts string $w = w_1 w_2 \ldots w_n$ each $w_i \in \Sigma$
if there is a sequence of states $r_0, r_1, r_2, , \ldots, r_n \in Q$
where:

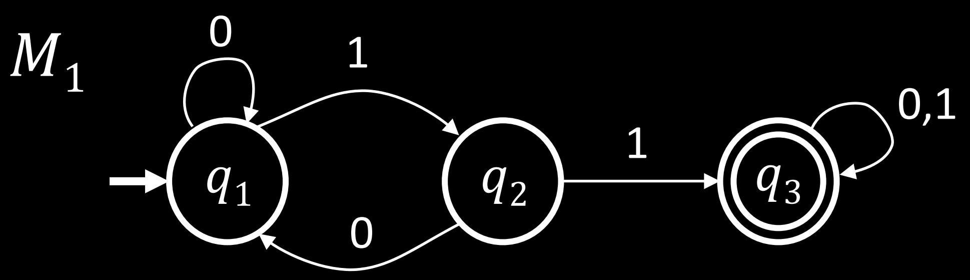
- $r_0 = q_0$
- $r_i = \delta(r_{i-1}, w_i)$ for $1 \leq i \leq n$
- $r_n \in F$

Recognizing languages

- $L(M) = \{w \mid M \text{ accepts } w\}$
- $L(M)$ is the language of M
- M recognizes $L(M)$

Defn: A language is regular if some finite automaton recognizes it.

Regular Languages – Examples



$L(M_1) = \{w|\ w \text{ contains substring } 11\} = A$

<u>Therefore A is regular</u>

More examples:

Let $B = \{w|\ w$ has an even number of 1s$\}$
B is regular (make automaton for practice).

Let $C = \{w|\ w$ has equal numbers of 0s and 1s$\}$
C is <u>not</u> regular (we will prove).

Goal: Understand the regular languages

Regular Expressions

Regular operations. Let A, B be languages:

- Union: $A \cup B = \{w | \; w \in A \text{ or } w \in B\}$
- Concatenation: $A \circ B = \{xy | \; x \in A \text{ and } y \in B\} = AB$
- Star: $A^* = \{x_1 \dots x_k | \text{ each } x_i \in A \text{ for } k \geq 0\}$

  Note: $\varepsilon \in A^*$ always

Example. Let $A = \{\text{good, bad}\}$ and $B = \{\text{boy, girl}\}$.

- $A \cup B = \{\text{good, bad, boy, girl}\}$
- $A \circ B = AB = \{\text{goodboy, goodgirl, badboy, badgirl}\}$
- $A^* = \{\varepsilon, \text{good, bad, goodgood, goodbad, badgood, badbad, goodgoodgood, goodgoodbad}, \dots\}$

Regular expressions

- Built from Σ, members $\Sigma, \emptyset, \varepsilon$ [Atomic]
- By using $\cup, \circ, *$ [Composite]

Examples:

- $(0 \cup 1)^* = \Sigma^*$ gives all strings over Σ
- $\Sigma^* 1$ gives all strings that end with 1
- $\Sigma^* 11 \Sigma^* =$ all strings that contain $11 = L(M_1)$

Goal: Show finite automata equivalent to regular expressions

Closure Properties for Regular Languages

Theorem: If A_1, A_2 are regular languages, so is $A_1 \cup A_2$ (closure under $\cup$)

Proof: Let $M_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$ recognize A_1
$M_2 = (Q_2, \Sigma, \delta_2, q_2, F_2)$ recognize A_2

Construct $M = (Q, \Sigma, \delta, q_0, F)$ recognizing $A_1 \cup A_2$

M should accept input w if either M_1 or M_2 accept w.

Check-in 1.1

In the proof, if M_1 and M_2 are finite automata where M_1 has k_1 states and M_2 has k_2 states Then how many states does M have?

(a) $k_1 + k_2$
(b) $(k_1)^2 + (k_2)^2$
(c) $k_1 \times k_2$

Components of $\boldsymbol{M}$:

$Q = Q_1 \times Q_2$
$= \{(q_1, q_2) | q_1 \in Q_1 \text{ and } q_2 \in Q_2\}$

$q_0 = (q_1, q_2)$

$\delta((q, r), a) = (\delta_1(q, a), \delta_2(r, a))$

$F = \cancel{F_1 \times F_2}$ NO! [gives intersection]

$F = (F_1 \times Q_2) \cup (Q_1 \times F_2)$

Check-in 1.1

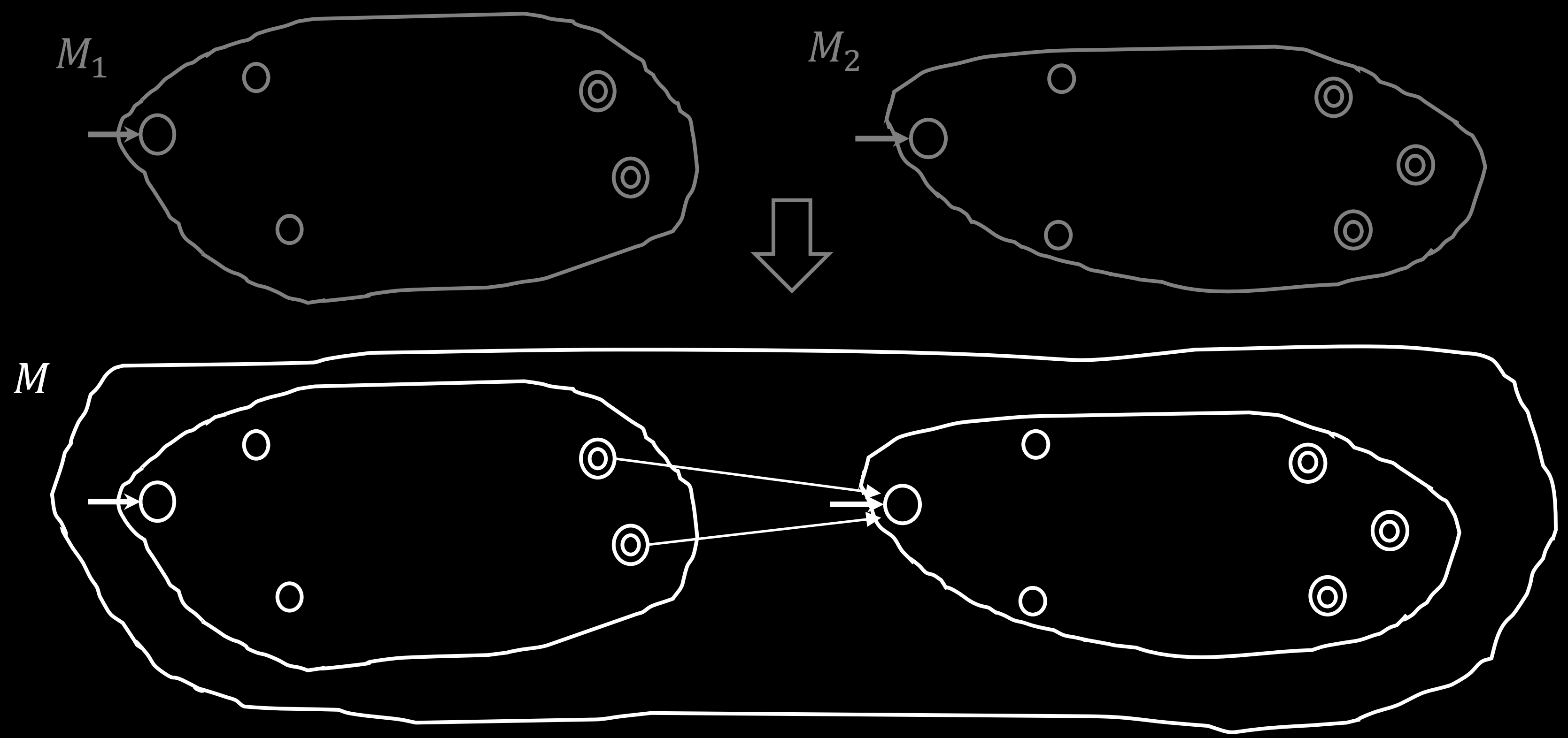
Closure Properties continued

Theorem: If A_1, A_2 are regular languages, so is A_1A_2 (closure under $\circ$)

Proof: Let $M_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$ recognize A_1

$M_2 = (Q_2, \Sigma, \delta_2, q_2, F_2)$ recognize A_2

Construct $M = (Q, \Sigma, \delta, q_0, F)$ recognizing A_1A_2



M should accept input w
if $w = xy$ where
M_1 accepts x and M_2 accepts y.

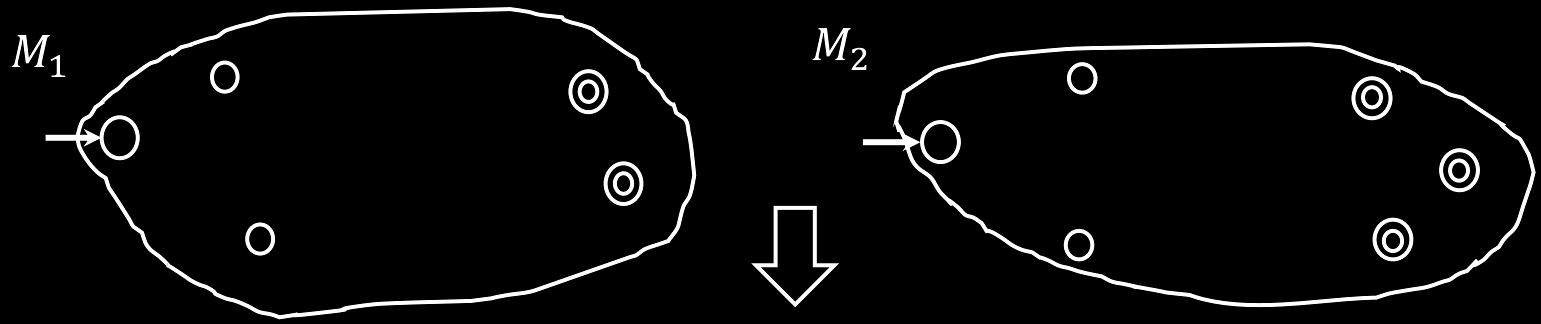
Doesn't work: Where to split w?

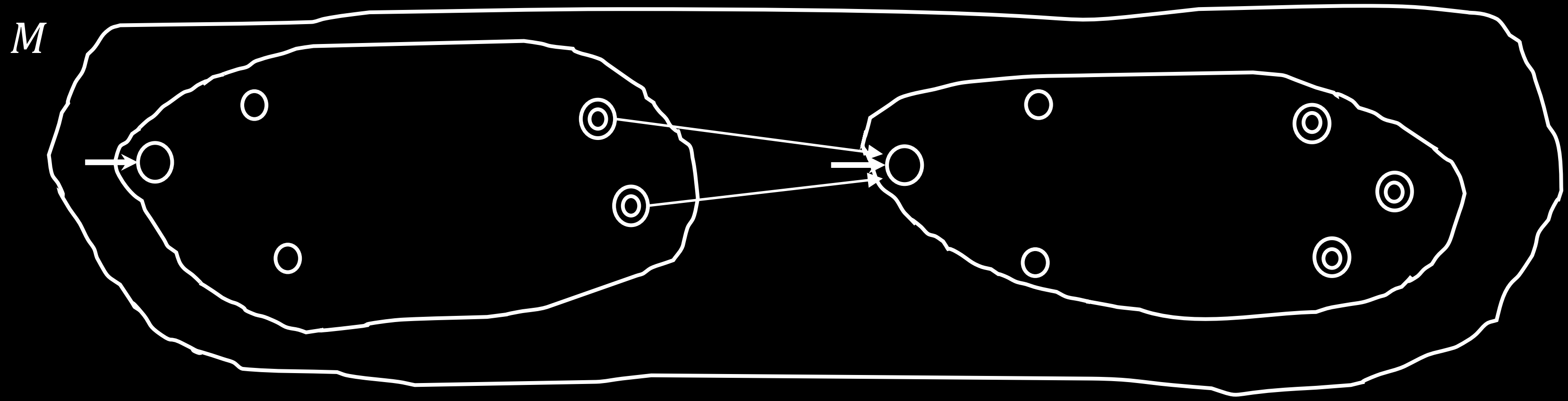
Closure Properties for Regular Languages

Theorem: If A_1, A_2 are regular languages, so is A_1A_2 (closure under $\circ$)

Recall proof attempt: Let $M_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$ recognize A_1
$M_2 = (Q_2, \Sigma, \delta_2, q_2, F_2)$ recognize A_2

Construct $M = (Q, \Sigma, \delta, q_0, F)$ recognizing A_1A_2



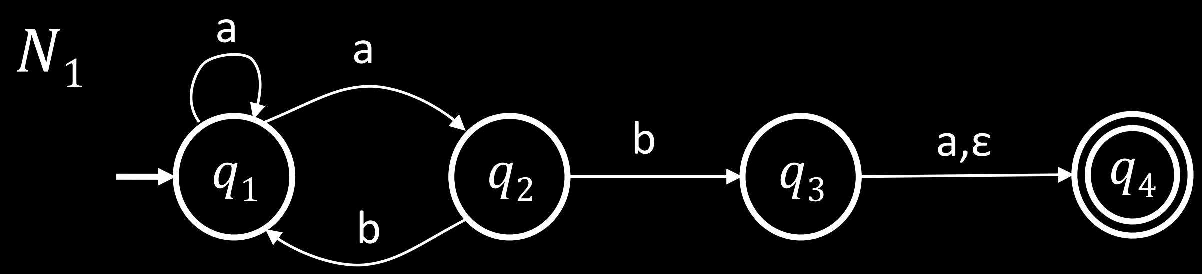


M should accept input w
if $w = xy$ where
M_1 accepts x and M_2 accepts y.

Doesn't work: Where to split w?

Hold off. Need new concept.

Nondeterministic Finite Automata



New features of nondeterminism:

- multiple paths possible (0, 1 or many at each step)
- ε-transition is a "free" move without reading input
- Accept input if <u>some</u> path leads to ◎ accept

Example inputs:

- ab
- aa
- aba
- abb

Check-in 2.1

What does N_1 do on input aab ?

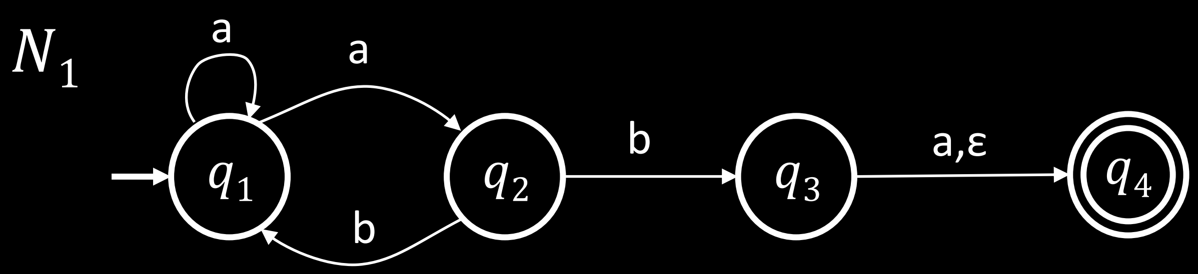
(a) Accept

(b) Reject

(c) Both Accept and Reject

Check-in 2.1

NFA – Formal Definition



Defn: A <u>nondeterministic finite automaton (NFA)</u>
N is a 5-tuple $(Q, \Sigma, \delta, q_0, F)$

states
alphabet
transition function
start state
accept states

- all same as before except δ
- $\delta: Q \times \underbrace{\Sigma_\varepsilon}_{\Sigma \cup \{\varepsilon\}} \to \mathcal{P}(Q) = \{R | R \subseteq Q\}$ power set
- In the N_1 example: $\delta(q_1, \mathrm{a}) = \{q_1, q_2\}$
  $\delta(q_1, \mathrm{b}) = \emptyset$

Ways to think about nondeterminism:

<u>Computational:</u> Fork new parallel thread and accept if any thread leads to an accept state.

<u>Mathematical:</u> Tree with branches.
Accept if any branch leads to an accept state.

<u>Magical:</u> Guess at each nondeterministic step which way to go. Machine always makes the right guess that leads to accepting, if possible.

Converting NFAs to DFAs

Theorem: If an NFA recognizes A then A is regular

Proof: Let NFA $M = (Q, \Sigma, \delta, q_0, F)$ recognize A

Construct DFA $M' = (Q', \Sigma, \delta', q_0', F')$ recognizing A

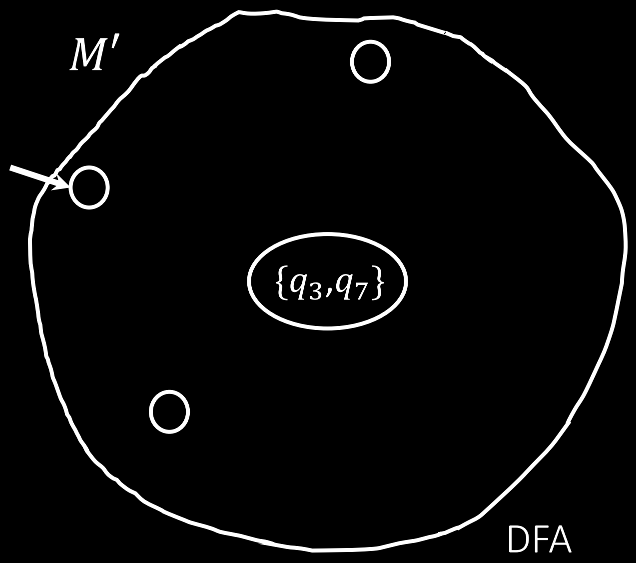
(Ignore the **ε-transitions, can easily modify to handle them)**

IDEA: DFA M' keeps track of the <u>subset of possible states</u> in NFA M.

Check-in 2.2

If M has n states, how many states does M' have by this construction?

(a) $2n$
(b) n^2
(c) 2^n



Construction of M':

$Q' = \mathcal{P}(Q)$

$\delta'(\underbrace{R}_{R \in Q'}, a) =$

$q_0' = \{q_0\}$

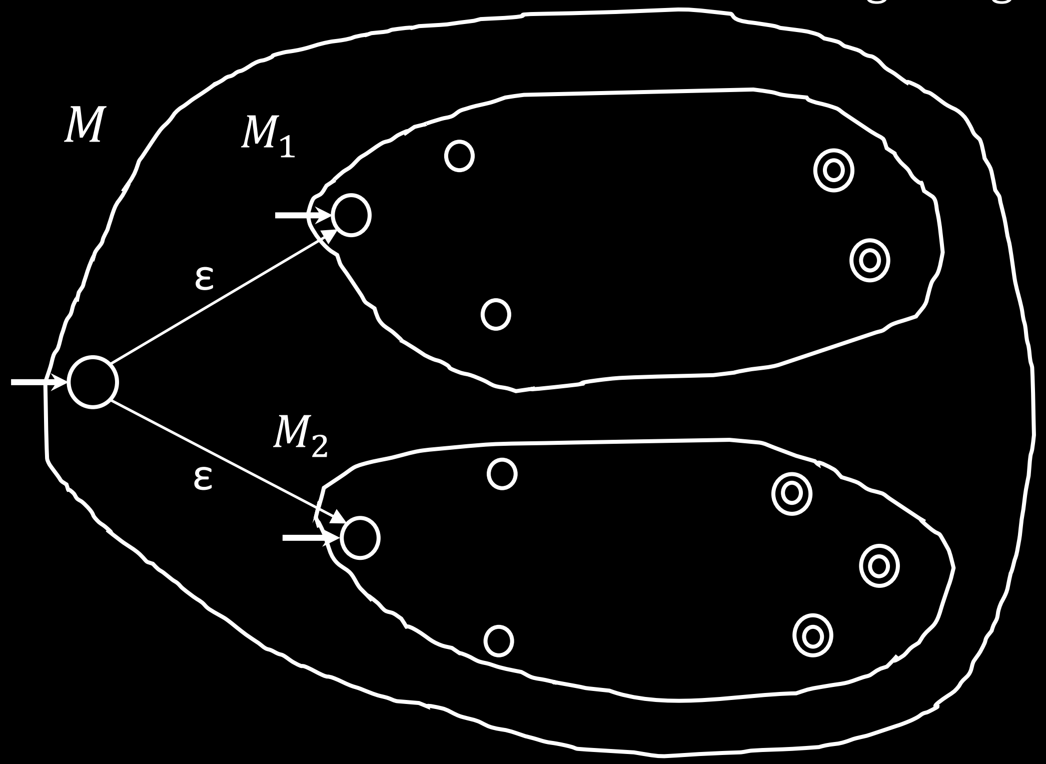
$F' = \{R \in Q' \mid R \text{ intersects } F\}$

Check-in 2.2

Return to Closure Properties

Recall Theorem: If A_1, A_2 are regular languages, so is $A_1 \cup A_2$
(The class of regular languages is closed under union)

New Proof (sketch): Given DFAs M_1 and M_2 recognizing A_1 and A_2
Construct NFA M recognizing $A_1 \cup A_2$

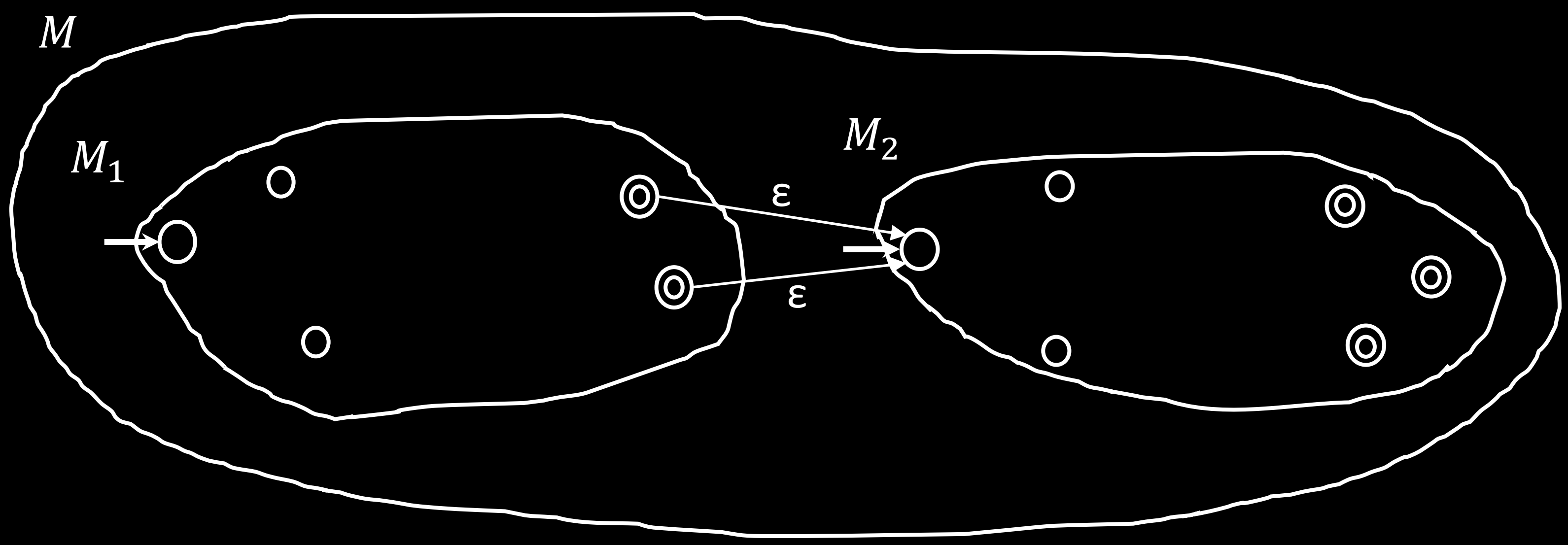


<u>Nondeterminism</u>
parallelism
vs
guessing

Closure under ∘ (concatenation)

Theorem: If A_1, A_2 are regular languages, so is A_1A_2

Proof sketch: Given DFAs M_1 and M_1 recognizing A_1 and A_2
Construct NFA M recognizing A_1A_2



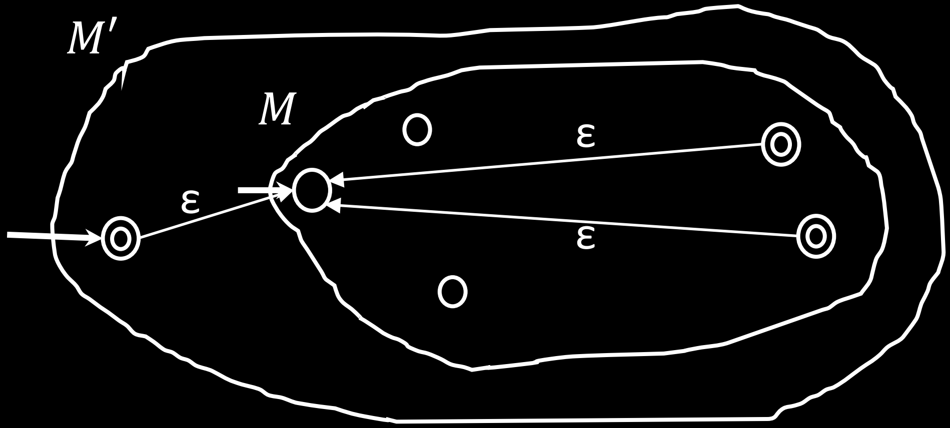
M should accept input w
if $w = xy$ where
M_1 accepts x and M_2 accepts y.

Nondeterministic M' has <u>the option</u>
to jump to M_2 when M_1 accepts.

Closure under $*$ (star)

Theorem: If A is a regular language, so is A^*

Proof sketch: Given DFA M recognizing A
Construct NFA M' recognizing A^*



Make sure M' accepts ε

Check-in 2.3

If M has n states, how many states does M' have by this construction?

(a) n
(b) $n + 1$
(c) $2n$

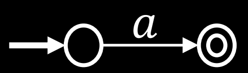
Regular Expressions → NFA

Theorem: If R is a regular expr and $A = L(R)$ then A is regular

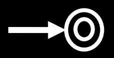
Proof: Convert R to equivalent NFA M:

If R is <u>atomic:</u> Equivalent M is:

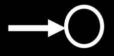
$R = a$ for $a \in \Sigma$



$R = \varepsilon$



$R = \emptyset$



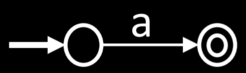
If R is <u>composite:</u>

$$\left.\begin{array}{l} R = R_1 \cup R_2 \\ R = R_1 \circ R_2 \\ R = R_1^* \end{array}\right\} \text{Use closure constructions}$$

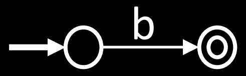
Example:

Convert $(\mathrm{a} \cup \mathrm{ab})^*$ to equivalent NFA

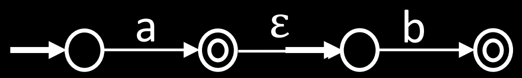
a:



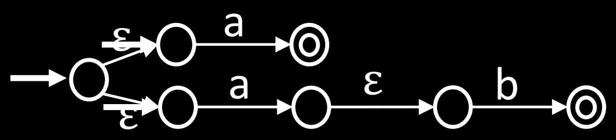
b:



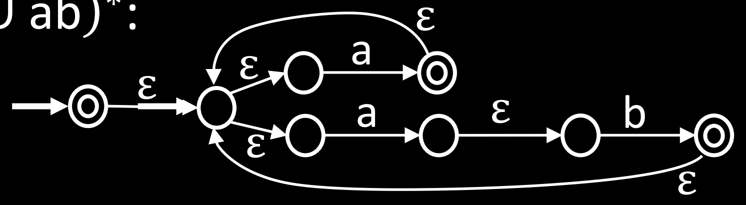
ab:



a ∪ ab:



$(\mathrm{a} \cup \mathrm{ab})^*$:



DFAs → Regular Expressions

Recall Theorem: If R is a regular expressipn and $A = L(R)$ then A is regular

Proof: Conversion $R \rightarrow$ NFA $M \rightarrow$ DFA M'

Regular expression R ⇨ M Finite automaton

Recall: we did $(\mathrm{a} \cup \mathrm{ab})^*$ as an example

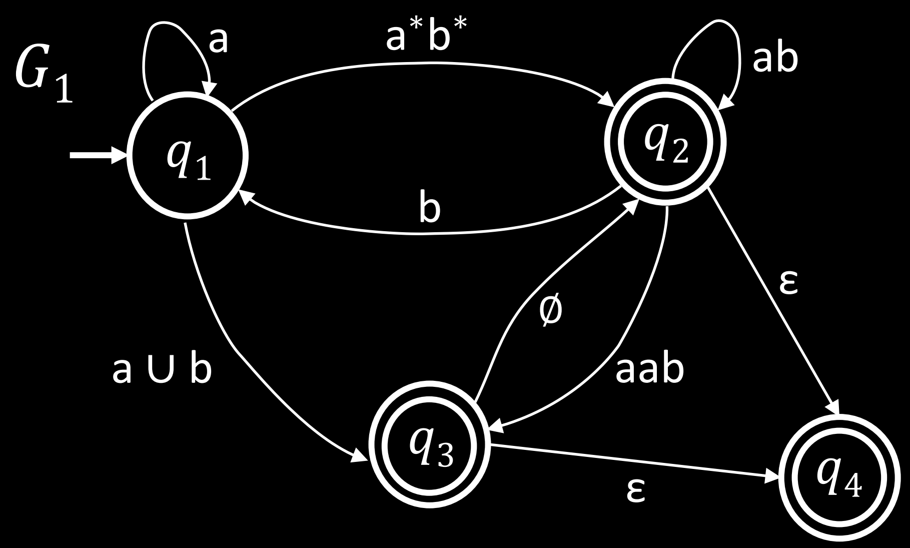
Today's Theorem: If A is regular then $A = L(R)$ for some regular expr R

Proof: Give conversion DFA $M \rightarrow R$

WAIT! Need new concept first.

Generalized NFA

Defn: A <u>Generalized Nondeterministic Finite Automaton</u> (GNFA) is similar to an NFA, but allows regular expressions as transition labels



For convenience we will assume:

- One accept state, separate from the start state
- One arrow from each state to each state, except
  a) only exiting the start state
  b) only entering the accept state

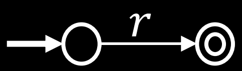
We can easily modify a GNFA to have this <u>special form</u>.

GNFA → Regular Expressions

Lemma: Every GNFA G has an equivalent regular expression R

Proof: By induction on the number of states k of G

<u>*Basis*</u> $(k = 2)$:

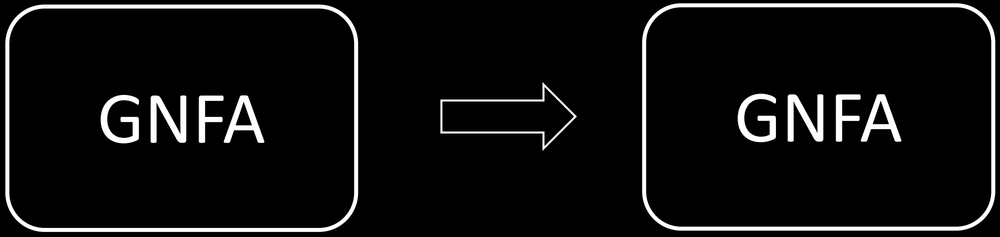
$G =$ 

Remember: G is in special form

Let $R = r$

Induction step $(k > 2)$: Assume Lemma true for $k - 1$ states and prove for k states

IDEA: Convert k-state GNFA to equivalent $(k - 1)$ -state GNFA



k states $\quad$ $k - 1$ states

k-state GNFA $\rightarrow$ $(k-1)$-state GNFA

Check-in 3.1

We just showed how to convert GNFAs to regular expressions but our goal was to show that how to convert DFAs to regular expressions. How do we finish our goal?

(a) Show how to convert DFAs to GNFAs

(b) Show how to convert GNFAs to DFAs

(c) We are already done. DFAs are a type of GNFAs.

Thus DFAs and regular expressions are equivalent.

1. Pick any state x except the start and accept states.
2. Remove x.
3. Repair the damage by recovering all paths that went through x.
4. Make the indicated change for each pair of states q_i, q_j .

Check-in 3.1

Non-Regular Languages

How do we show a language is not regular?

- Remember, to show a language *is* regular, we give a DFA.
- <u>To show a language is *not* regular, we must give a proof.</u>
- It is not enough to say that you couldn't find a DFA for it, therefore the language isn't regular.

Two examples: Here $\Sigma = \{0,1\}$.

1. Let $B = \{w |\ w$ has equal numbers of 0s and 1s$\}$

Intuition: B is not regular because DFAs cannot count unboundedly.

2. Let $C = \{w |\ w$ has equal numbers of 01 and 10 substrings$\}$

Intuition: C is not regular because DFAs cannot count unboundedly.
However C is regular!

Moral: You need to give a proof.

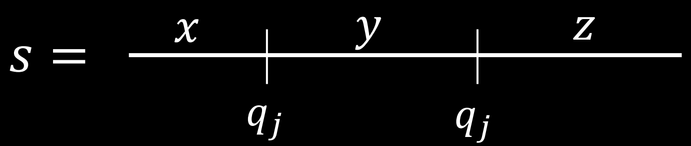
Method for Proving Non-regularity

Pumping Lemma: For every regular language A, there is a number p (the "pumping length") such that if $s \in A$ and $|s| \geq p$ then $s = xyz$ where

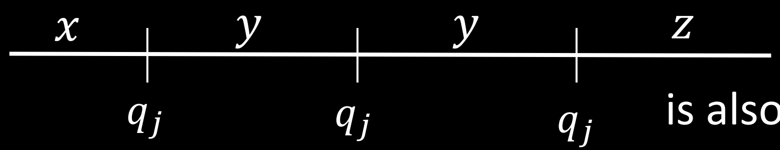
1) $xy^iz \in A$ for all $i \geq 0$ $\qquad y^i = \underbrace{yy\cdots y}_{i}$
2) $y \neq \varepsilon$
3) $|xy| \leq p$

Informally: A is regular → every long string in A can be pumped and the result stays in A.

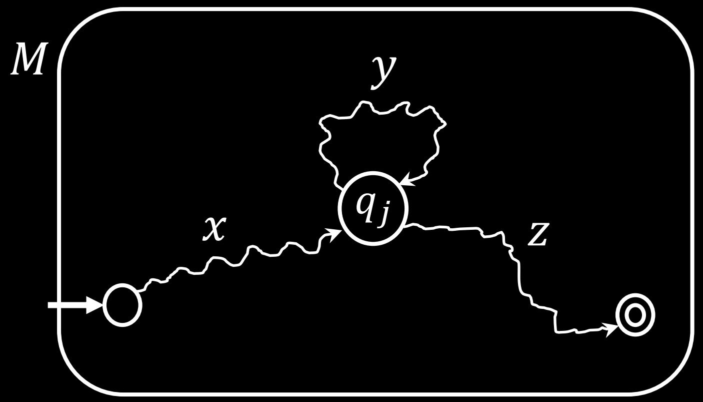
Proof: Let DFA M recognize A. Let p be the number of states in M. Pick $s \in A$ where $|s| \geq p$.



M will repeat a state q_j when reading s because s is so long.



is also accepted



The path that M follows when reading s.

Example 1 of Proving Non-regularity

Pumping Lemma: For every regular language A, there is a p such that if $s \in A$ and $|s| \geq p$ then $s = xyz$ where

1) $xy^iz \in A$ for all $i \geq 0$ $\qquad y^i = yy \cdots y$
2) $y \neq \varepsilon$
3) $|xy| \leq p$

Let $D = \{0^k 1^k | \, k \geq 0\}$

Show: D is not regular

Proof by Contradiction:

Assume (to get a contradiction) that D is regular.

The pumping lemma gives p as above. Let $s = 0^p 1^p \in D$.

Pumping lemma says that can divide $s = xyz$ satisfying the 3 conditions.

$$s = \underbrace{000 \cdots 000}_{x \,|\, y \;\leftarrow \leq p \rightarrow}\underbrace{111 \cdots 111}_{z}$$

But $xyyz$ has excess 0s and thus $xyyz \notin D$ contradicting the pumping lemma.

Therefore our assumption (D is regular) is false. We conclude that D is not regular.

Example 2 of Proving Non-regularity

Pumping Lemma: For every regular language A, there is a p such that if $s \in A$ and $|s| \geq p$ then $s = xyz$ where

1) $xy^iz \in A$ for all $i \geq 0$ $\quad$ $y^i = yy\cdots y$
2) $y \neq \varepsilon$
3) $|xy| \leq p$

Let $F = \{ww \mid w \in \Sigma^*\}$. Say $\Sigma^* = \{0,1\}$.

Show: F is not regular

Proof by Contradiction:

Assume (for contradiction) that F is regular.

The pumping lemma gives p as above. Need to choose $s \in F$. Which s?

Try $s = 0^p0^p \in F$.

$$s = \underbrace{000\cdots 0}_{x}\underbrace{00}_{y}\underbrace{0000\cdots 000}_{z}, \quad |xy| \leq p$$

$y = 00$

Try $s = 0^p10^p1 \in F$. Show cannot be pumped $s = xyz$ satisfying the 3 conditions.

$$s = \underbrace{000\cdots 0}_{x}\underbrace{0}_{y}\underbrace{01000\cdots 001}_{z}, \quad |xy| \leq p$$

$xyyz \notin F$ Contradiction! Therefore F is not regular.

Example 3 of Proving Non-regularity

Variant: Combine closure properties with the Pumping Lemma.

Let $B = \{w \mid w \text{ has equal numbers of 0s and 1s}\}$

Show: B is not regular

Proof by Contradiction:

Assume (for contradiction) that B is regular.

We know that 0^*1^* is regular so $B \cap 0^*1^*$ is regular (closure under intersection).

But $D = B \cap 0^*1^*$ and we already showed D is not regular. Contradiction!

Therefore our assumption is false, so B is not regular.

Context Free Grammars

G_1

$$\left.\begin{array}{l} S \rightarrow 0S1 \\ S \rightarrow R \\ R \rightarrow \varepsilon \end{array}\right\} \text{(Substitution) Rules}$$

Rule: Variable → string of variables and terminals
Variables: Symbols appearing on left-hand side of rule
Terminals: Symbols appearing only on right-hand side
Start Variable: Top left symbol

Grammars generate strings

1. Write down start variable
2. Replace any variable according to a rule
   Repeat until only terminals remain
3. Result is the generated string
4. $L(G)$ is the language of all generated strings.

Check-in 3.3

G_2

$S \rightarrow RR$
$R \rightarrow 0R1$
$R \rightarrow \varepsilon$

Check all of the strings that are in $L(G_2)$:

(a) 001101
(b) 000111
(c) 1010
(d) ε

Check-in 3.3

Context Free Grammars (CFGs)

G_1

S → 0S1

S → R

R → ε

Shorthand:

S → 0S1 | R

R → ε

Recall that a CFG has terminals, variables, and rules.

Grammars generate strings

1. Write down start variable
2. Replace any variable according to a rule
   Repeat until only terminals remain
3. Result is the generated string
4. $L(G)$ is the language of all generated strings
5. We call $L(G)$ a Context Free Language.

Example of G_1 generating a string

Tree of substitutions "parse tree"

S

S

Resulting string

$\in L(G_1)$

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

CFG – Formal Definition

Defn: A Context Free Grammar (CFG) G is a 4-tuple (

V finite set of variables

Σ finite set of terminal symbols

R finite set of rules (rule form: $V \to (V \cup \Sigma)^*$)

S start variable

For $u, v \in (V \cup \Sigma)^*$ write

1) $u \Rightarrow v$ if can go from u to v with one substitution step in G

2) $u \overset{*}{\Rightarrow} v$ if can go from u to v with some number of substitution steps in G

$u \Rightarrow u_1 \Rightarrow u_2 \Rightarrow \cdots \Rightarrow u_k = v$ is called a derivation of v from u.

If $u = S$ then it is a derivation of v.

$L(G) = \{w |\ w \in \Sigma^* \text{ and } S \overset{*}{\Rightarrow} w\}$

Defn: A is a Context Free Language (CFL) if $A = L(G)$ for some CFG G.

Check-in 4.1

Which of these are valid CFGs?

C_1: B → 0B1 | ε
B1 → 1B
0B → 0B

C_2: S → 0S | S1
R → RR

a) C_1 only
b) C_2 only
c) Both C_1 and C_2
d) Neither

Check-in 4.1

Ambiguity

G_2

E → E+T | T

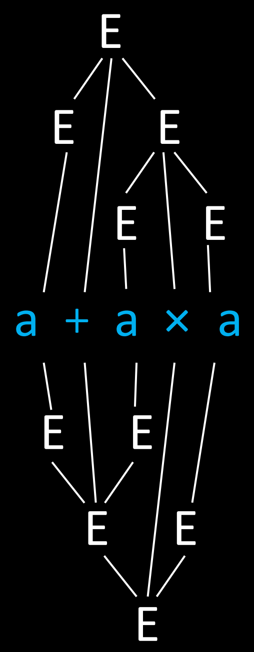
T → T×F | F

F → (E) | a

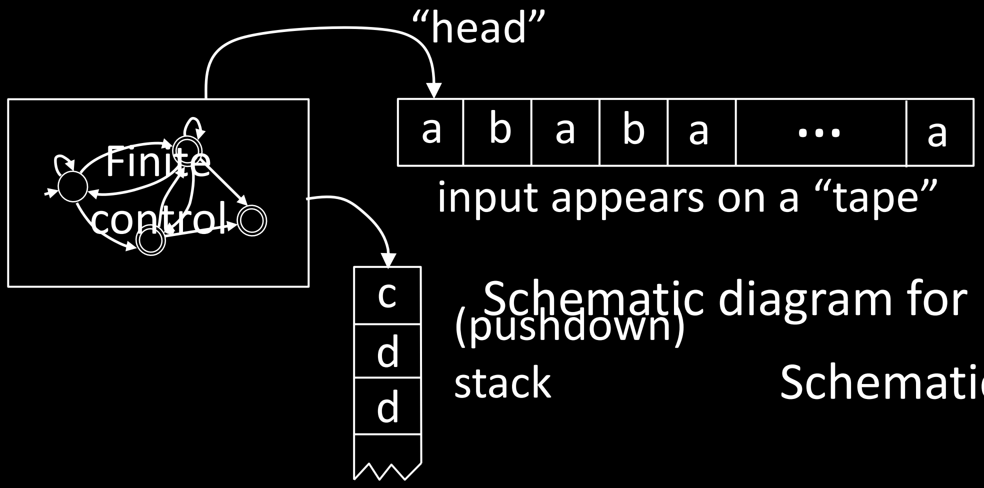
G_3

E → E+E | E×E | (E) | a

Both G_2 and G_3 recognize the same language, i.e., $L(G_2) = L(G_3)$.
However G_2 is an unambiguous CFG and G_3 is ambiguous.



Pushdown Automata (PDA)



Schematic diagram for DFA or NFA

Schematic diagram for PDA

Operates like an NFA except can <u>write-add</u> or <u>read-remove</u> symbols from the top of stack.

push

pop

Example: PDA for $D = \{0^k 1^k \mid k \geq 0\}$

1) Read 0s from input, push onto stack until read 1.
2) Read 1s from input, while popping 0s from stack.
3) Enter accept state if stack is empty. (note: acceptance only at end of input)

PDA – Formal Definition

Defn: A Pushdown Automaton (PDA) is a 6-tuple $(Q, \Sigma, \Gamma, \delta, q_0, F)$

Σ input alphabet

Γ stack alphabet

δ: $\mathrm{Q} \times \Sigma_\varepsilon \times \Gamma_\varepsilon \rightarrow \mathcal{P}(Q \times \Gamma_\varepsilon)$

$\delta(q, \mathrm{a}, \mathrm{c}) = \{(r_1, \mathrm{d}),\ (r_2, \mathrm{e})\}$

Accept if some thread is in the accept state at the end of the input string.

Example: PDA for $B = \{ww^{\mathcal{R}} \mid w \in \{0,1\}^*\}$ Sample input:

0	1	1	1	1	0

1) Read and push input symbols.
   Nondeterministically either repeat or go to (2).
2) Read input symbols and pop stack symbols, compare.
   If ever $\neq$ then thread rejects.
3) Enter accept state if stack is empty. (do in “software”)

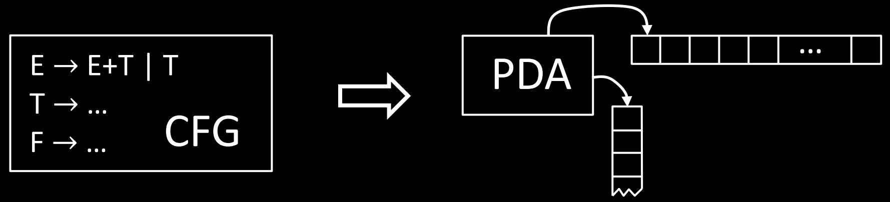
The nondeterministic forks replicate the stack.

This language requires nondeterminism.
Our PDA model is nondeterministic.

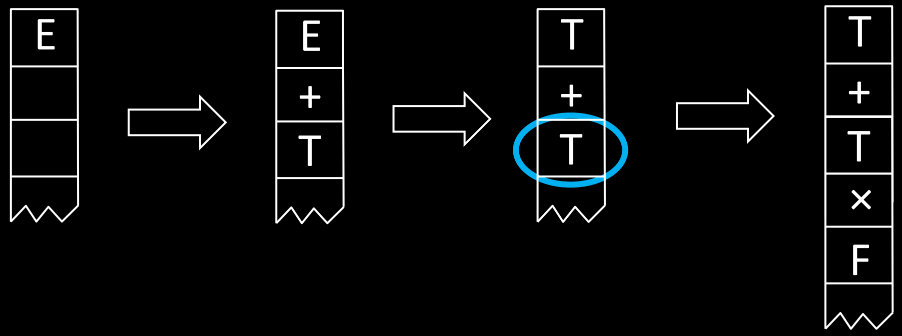
Converting CFGs to PDAs

Theorem: If A is a CFL then some PDA recognizes A

Proof: Convert A's CFG to a PDA



IDEA: PDA begins with starting variable and guesses substitutions.
It keeps intermediate generated strings on stack. When done, compare with input.



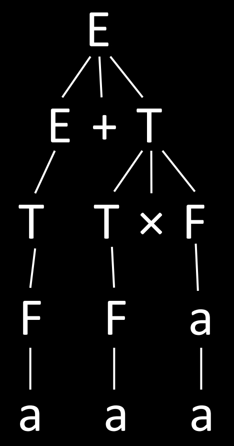
Input:

a	+	a	×	a

G_2

E → E+T | T
T → T×F | F
F → (E) | a

E
E+T
T+T×F
F+F×a
a+a×a



Problem! Access below the top of stack is cheating!

Instead, only substitute variables when on the top of stack.

If a terminal is on the top of stack, pop it and compare with input. Reject if $\neq$.

Converting CFGs to PDAs (contd)

G_2
E → E+T | T
T → T×F | F
F → (E) | a

Theorem: If A is a CFL then some PDA recognizes A

Proof construction: Convert the CFG for A to the following PDA.

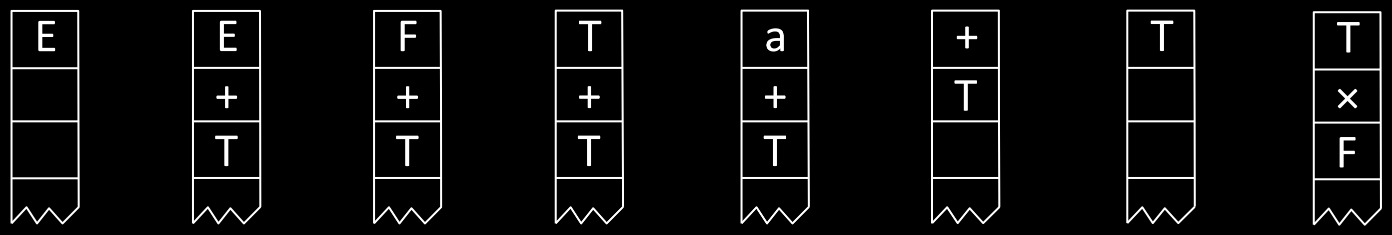
1) Push the start symbol on the stack.
2) If the top of stack is

   Variable: replace with right hand side of rule (nondet choice).

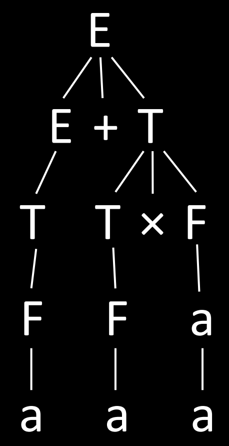
   Terminal: pop it and match with next input symbol.
3) If the stack is empty, *accept.*

Example:

a	+	a	×	a



E
E+T
T+T×F
F+F×a
a+a×a



Equivalence of CFGs and PDAs

Theorem: A is a CFL iff* some PDA recognizes A

$\longleftrightarrow$ Done.

In book. You are responsible for knowing it is true, but not for knowing the proof.

* "iff" = "if an only if" means the implication goes both ways.
So we need to prove both directions: forward ($\rightarrow$) and reverse ($\leftarrow$).

Check-in 4.3

Is every Regular Language also a Context Free Language?

(a) Yes

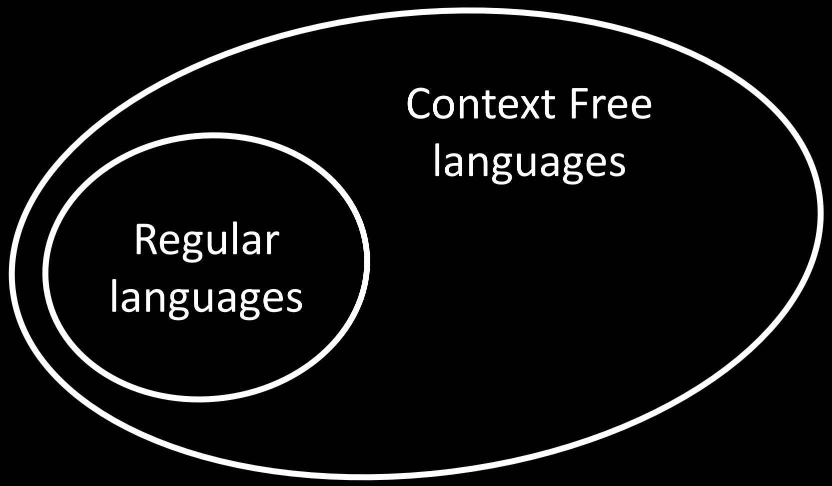
(b) No

(c) Not sure

Check-in 4.3

Recap

	Recognizer	Generator
Regular language	DFA or NFA	Regular expression
Context Free language	PDA	Context Free Grammar



Equivalence of CFGs and PDAs

Recall Theorem: A is a CFL iff some PDA recognizes A

$\longrightarrow$ Done.

$\longleftarrow$ Need to know the fact, not the proof

Corollaries:

1) Every regular language is a CFL.
2) If A is a CFL and B is regular then $A \cap B$ is a CFL.

Proof sketch of (2):
While reading the input, the finite control of the PDA for A simulates the DFA for B.

Note 1: If A and B are CFLs then $A \cap B$ may not be a CFL (will show today).
Therefore the class of CFLs is not closed under $\cap$.

Note 2: The class of CFLs is closed under $\cup, \circ, *$ (see Pset 2).

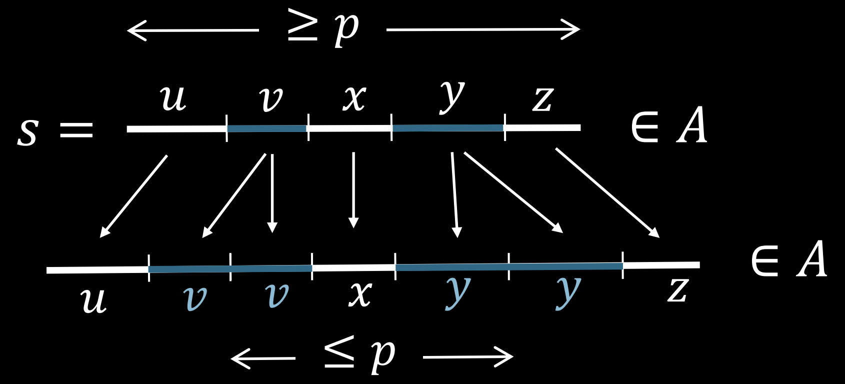
Proving languages not Context Free

Let $B = \{0^k 1^k 2^k \mid k \geq 0\}$. We will show that B isn't a CFL.

Pumping Lemma for CFLs: For every CFL A, there is a p such that if $s \in A$ and $|s| \geq p$ then $s = uvxyz$ where

1) $uv^i xy^i z \in A$ for all $i \geq 0$
2) $vy \neq \varepsilon$
3) $|vxy| \leq p$

Informally: All long strings in A are pumpable and stay in A.

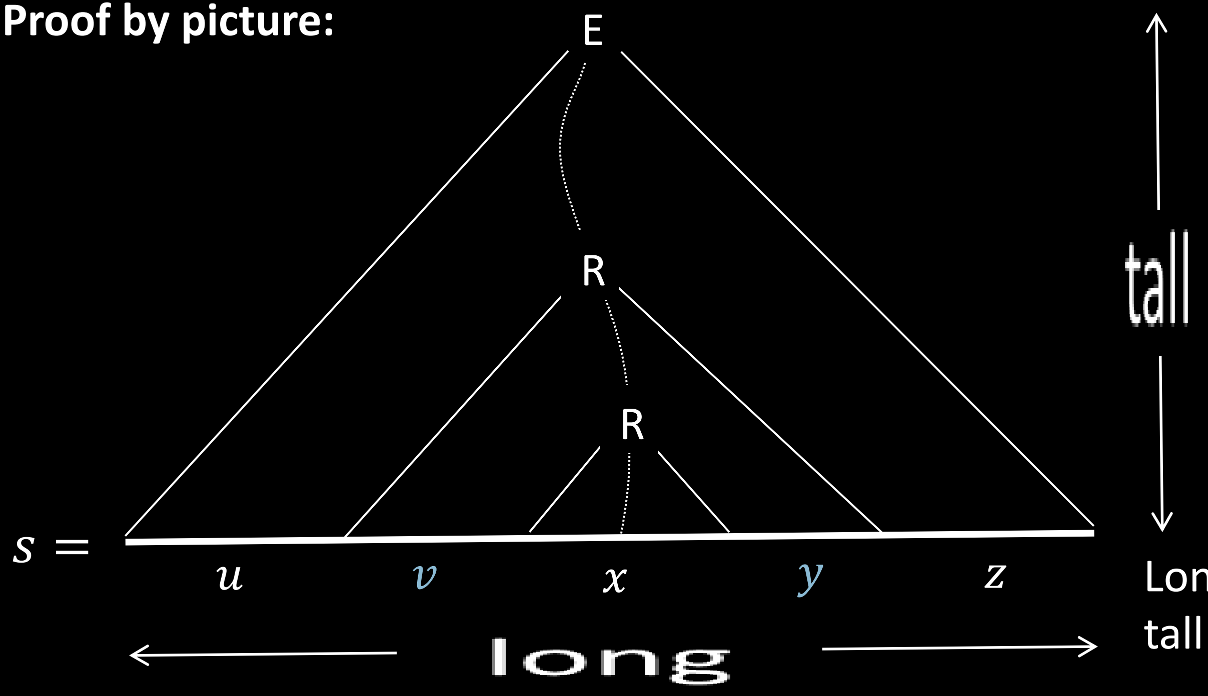


Pumping Lemma – Proof

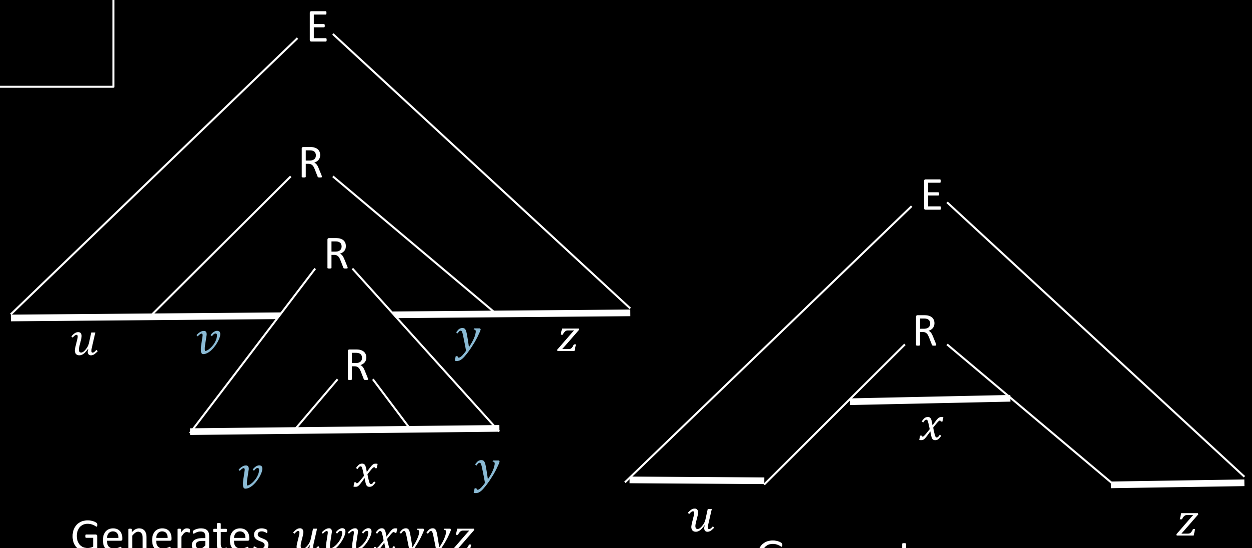
Pumping Lemma for CFLs: For every CFL A, there is a p such that if $s \in A$ and $|s| \geq p$ then $s = uvxyz$ where

1) $uv^i xy^i z \in A$ for all $i \geq 0$
2) $vy \neq \varepsilon$
3) $|vxy| \leq p$

Proof by picture:



Long $s \rightarrow$
tall parse tree



Generates $uvvxyyz$
$= uv^2xy^2z$

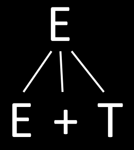
Generates uxz
$= uv^0xy^0z$

"cutting and pasting" argument

Pumping Lemma – Proof details

For $s \in A$ where $|s| \geq p$, we have $s = uvxyz$ where:
1) $uv^ixy^iz \in A$ for all $i \geq 0$
2) $vy \neq \varepsilon$
3) $|vxy| \leq p$

Let $b =$ the length of the longest right hand side of a rule (E → E+T)
$=$ the max branching of the parse tree



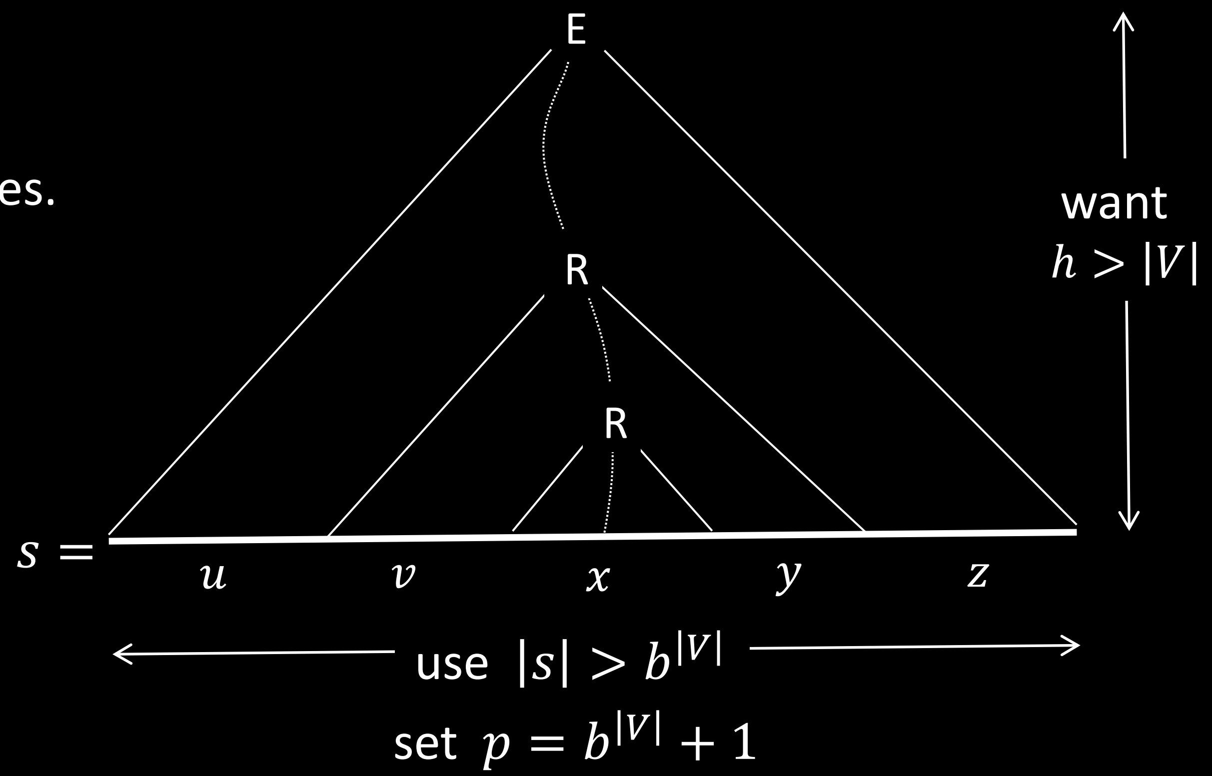
Let $h =$ the height of the parse tree for s.

A tree of height h and max branching b has at most b^h leaves.
So $|s| \leq b^h$.

Let $p = b^{|V|} + 1$ where $|V| =$ # variables in the grammar.

So if $|s| \geq p > b^{|V|}$ then $|s| > b^{|V|}$ and so $h > |V|$.

Thus at least $|V| + 1$ variables occur in the longest path.
So some variable R must repeat on a path.



Example 1 of Proving Non-CF

Pumping Lemma for CFLs: For every CFL A, there is a p such that if $s \in A$ and $|s| \geq p$ then $s = uvxyz$ where

1) $uv^ixy^iz \in A$ for all $i \geq 0$
2) $vy \neq \varepsilon$
3) $|vxy| \leq p$

Let $B = \{0^k1^k2^k \mid k \geq 0\}$

Show: B is not a CFL

Check-in 5.1

Let $A_1 = \{0^k1^k2^l \mid k,l \geq 0\}$ (equal #s of 0s and 1s)

Let $A_2 = \{0^l1^k2^k \mid k,l \geq 0\}$ (equal #s of 1s and 2s)

Observe that PDAs can recognize A_1 and A_2. What can we now conclude?

a) The class of CFLs is not closed under intersection.

b) The Pumping Lemma shows that $A_1 \cup A_2$ is not a CFL .

c) The class of CFLs is closed under complement.

$$s = \underbrace{00\cdots0011\cdots1122\cdots22}_{u\ |\ v\ |\ x\ |\ y\ |\ z}$$

$\leftarrow \leq p \rightarrow$

Example 2 of Proving Non-CF

Pumping Lemma for CFLs: For every CFL A, there is a p such that if $s \in A$ and $|s| \geq p$ then $s = uvxyz$ where

1) $uv^ixy^iz \in A$ for all $i \geq 0$
2) $vy \neq \varepsilon$
3) $|vxy| \leq p$

Let $F = \{ww \mid w \in \Sigma^*\}$. $\Sigma = \{0,1\}$.

Show: F is not a CFL.

Assume (for contradiction) that F is a CFL.

The CFL pumping lemma gives p as above. Need to choose $s \in F$. Which s?

Try $s_1 = 0^p10^p1 \in F$.

$$s_1 = 000\cdots001000\cdots001$$

u | v | x | y | z

$\leftarrow \leq p \rightarrow$

Try $s_2 = 0^p1^p0^p1^p \in F$.

$$s_2 = 0\cdots01\cdots10\cdots01\cdots1$$

u | v | x | y | z

$\leftarrow \leq p \rightarrow$

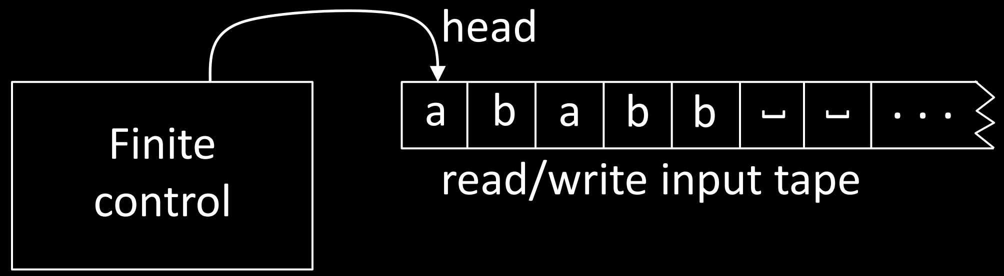
Show s_2 cannot be pumped $s_2 = uvxyz$ satisfying the 3 conditions.

Condition 3 implies that vxy does not overlap two runs of 0s or two runs of 1s.

Therefore, in uv^2xy^2z, two runs of 0s or two runs of 1s have unequal length.

So $uv^2xy^2z \notin F$ violating Condition 1. Contradiction! Thus F is not a CFL.

Turing Machines (TMs)

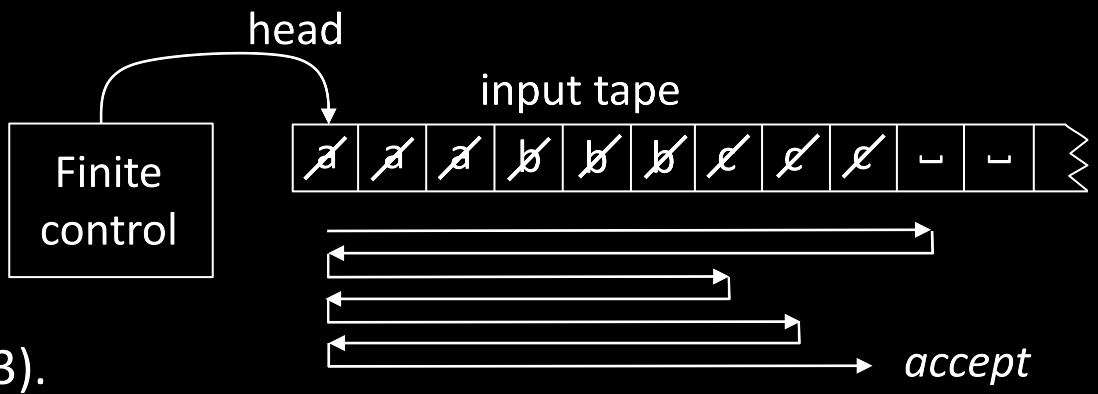


1) Head can read and write
2) Head is two way (can move left or right)
3) Tape is infinite (to the right)
4) Infinitely many blanks "␣" follow input
5) Can accept or reject any time (not only at end of input)

TM – example

TM recognizing $B = \{\mathrm{a}^k\mathrm{b}^k\mathrm{c}^k \mid k \geq 0\}$

1) Scan right until $\sqcup$ while checking if input is in $\mathrm{a}^*\mathrm{b}^*\mathrm{c}^*$, *reject* if not.
2) Return head to left end.
3) Scan right, crossing off single a, b, and c.
4) If the last one of each symbol, *accept*.
5) If the last one of some symbol but not others, *reject*.
6) If all symbols remain, return to left end and repeat from (3).



Check-in 5.2

How do we get the effect of "crossing off" with a Turing machine?

a) We add that feature to the model.

b) We use a tape alphabet $\Gamma = \{\mathrm{a}, \mathrm{b}, \mathrm{c}, \cancel{\mathrm{a}}, \cancel{\mathrm{b}}, \cancel{\mathrm{c}}, \sqcup\}$.

c) All Turing machines come with an eraser.

Check-in 5.2

TM – Formal Definition

Defn: A Turing Machine (TM) is a 7-tuple $(Q, \Sigma, \Gamma, \delta, q_0, q_{\text{acc}}, q_{\text{rej}})$

Σ input alphabet

Γ tape alphabet ($\Sigma \subseteq \Gamma$)

δ: $\mathrm{Q} \times \Gamma \to Q \times \Gamma \times \{\mathrm{L}, \mathrm{R}\}$ (L = Left, R = Right)

$\delta(q, \mathrm{a}) = (r, \mathrm{b}, \mathrm{R})$

On input w a TM M may halt (enter q_{acc} or q_{rej}) or M may run forever ("loop").

So M has 3 possible outcomes for each input w:

1. *Accept* w (enter q_{acc})
2. *Reject* w by halting (enter q_{rej})
3. *Reject* w by looping (running forever)

Check-in 5.3

This Turing machine model is deterministic.
How would we change it to be nondeterministic?

a) Add a second transition function.

b) Change δ to be δ: $\mathrm{Q} \times \Gamma \to \mathcal{P}(Q \times \Gamma \times \{\mathrm{L}, \mathrm{R}\})$

c) Change the tape alphabet Γ to be infinite.

TM Recognizers and Deciders

Let M be a TM. Then $L(M) = \{w |\ M \text{ accepts } w\}$.

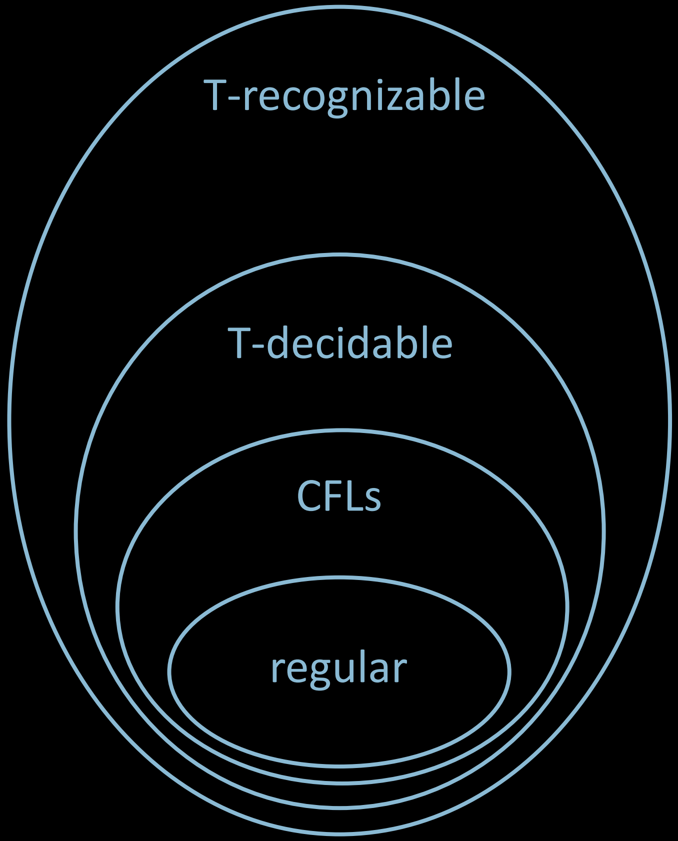
Say that M recognizes A if $A = L(M)$.

Defn: A is Turing-recognizable if $A = L(M)$ for some TM M.

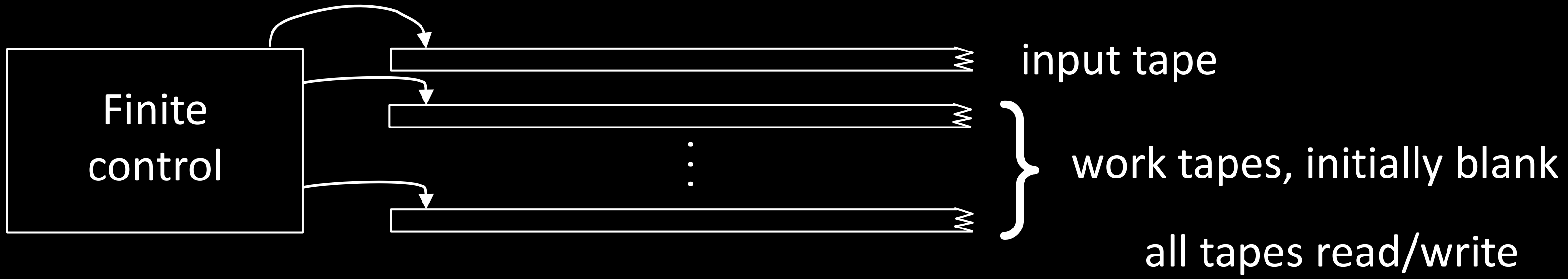
Defn: TM M is a decider if M halts on all inputs.

Say that M decides A if $A = L(M)$ and M is a decider.

Defn: A is Turing-decidable if $A = L(M)$ for some TM decider M.

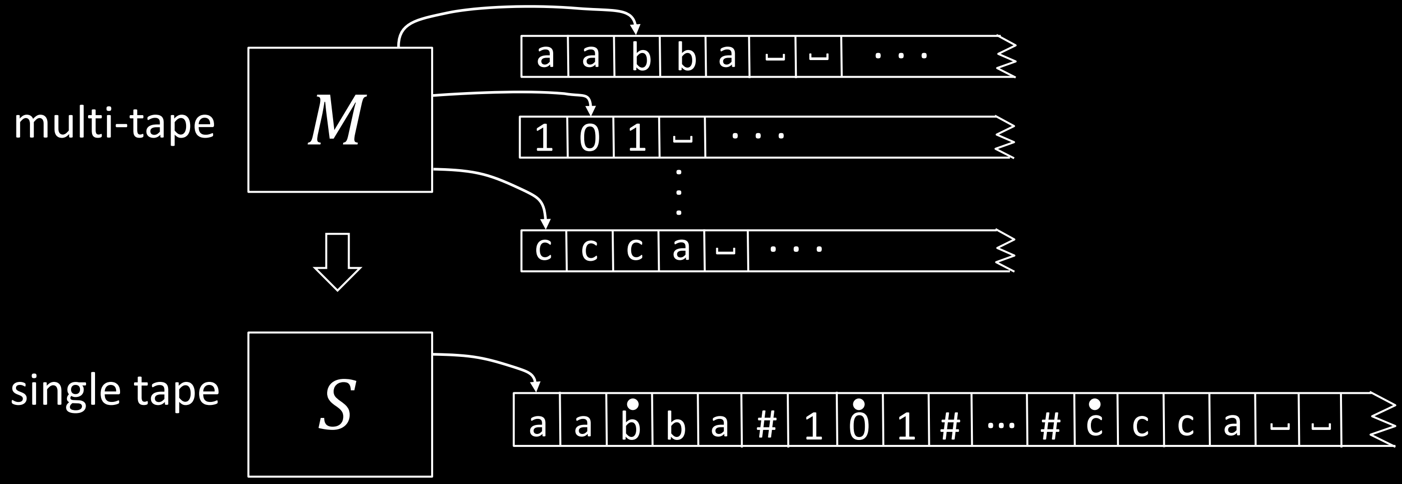


Multi-tape Turing machines



Theorem: A is T-recognizable iff some multi-tape TM recognizes A

Proof: ($\rightarrow$) immediate. ($\leftarrow$) convert multi-tape to single tape:



S simulates M by storing the contents of multiple tapes on a single tape in “blocks”. Record head positions with dotted symbols.

Some details of S:

1) To simulate each of M’s steps
   a. Scan entire tape to find dotted symbols.
   b. Scan again to update according to M’s δ.
   c. Shift to add room as needed.
2) Accept/reject if M does.

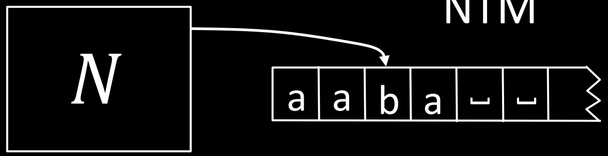
Nondeterministic Turing machines

A <u>Nondeterministic TM</u> (NTM) is similar to a Deterministic TM except for its transition function $\delta\colon \mathrm{Q} \times \Gamma \to \mathcal{P}(\,Q \times \Gamma \times \{\mathrm{L}, \mathrm{R}\}\,)$.

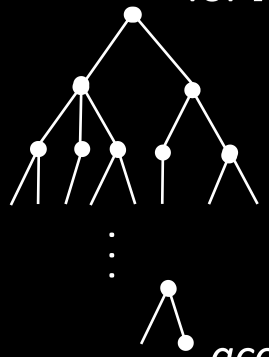
Theorem: A is T-recognizable iff some NTM recognizes A

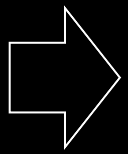
Proof: $(\rightarrow)$ immediate. $(\leftarrow)$ convert NTM to Deterministic TM.

NTM

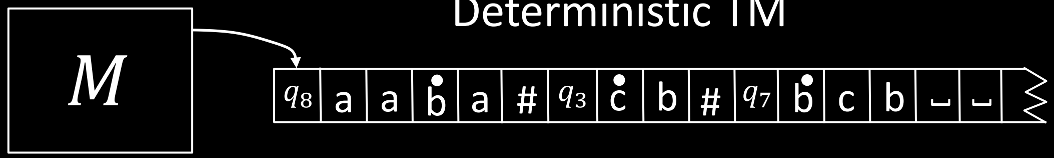


Nondeterministic computation tree for N on input w.





Deterministic TM

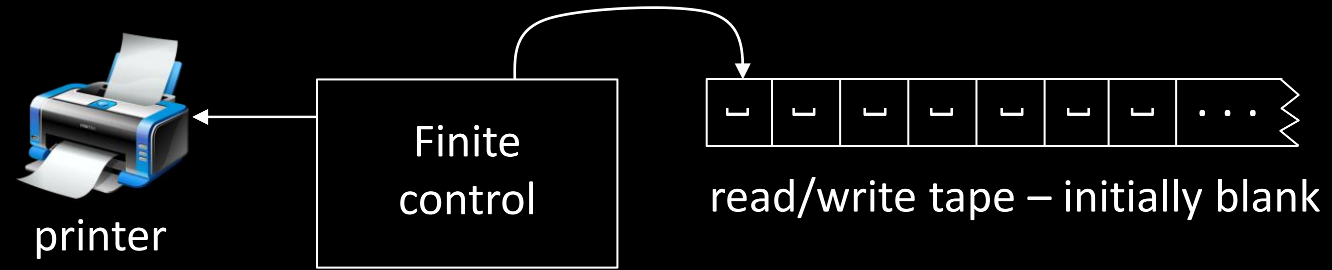


M simulates N by storing each thread's tape in a separate "block" on its tape.
Also need to store the head location,
and the state for each thread, in the block.

If a thread forks, then M copies the block.

If a thread accepts then M accepts.

Turing Enumerators



Defn: A <u>Turing Enumerator</u> is a deterministic TM with a printer.

It starts on a blank tape and it can print strings $w_1, w_2, w_3, \ldots$ possibly going forever.

Its language is the set of all strings it prints. It is a generator, not a recognizer.

For enumerator E we say $L(E) = \{w \mid E \text{ prints } w\}$.

Theorem: A is T-recognizable iff $A = L(E)$ for some T-enumerator E.

Proof: $(\rightarrow)$ Convert TM M to equivalent enumerator E.

$E =$ Simulate M on each w_i in $\Sigma^* = \{\varepsilon, 0,1,00,01,10, \ldots\}$

If M accepts w_i then print w_i .

Continue with next w_i .

Problem: What if M on w_i loops?

Fix: Simulate M on $w_1, w_2, \ldots, w_i$ for i steps, for $i = 1,2, \ldots$

Print those w_i which are accepted.

Check-in 6.1

When converting TM M to enumerator E, does E always print the strings in ***string order***?

a) Yes.

b) No.

Check-in 6.1

Church-Turing Thesis ~1936

Alonzo Church
1903–1995

Algorithm	=	Turing machine
Intuitive		Formal

Instead of Turing machines,
can use any other “reasonable” model
of unrestricted computation:
λ-calculus, random access machine,
your favorite programming language, …

Big impact on mathematics.

Alan Turing
1912–1954

Check-in 6.2

Which is the following is true about Alan Turing?
Check all that apply.

a) Broke codes for England during WW2.
b) Worked in AI.
c) Worked in Biology.
d) Was imprisoned for being gay.
e) Appears on a British banknote.

Hilbert's 10^{th} Problem

In 1900 David Hilbert posed 23 problems

#1) Problem of the continuum (Does set A exist where $|\mathbb{N}| < |A| < |\mathbb{R}|$?).

#2) Prove that the axioms of mathematics are consistent.

#10) Give an algorithm for solving *Diophantine equations.*

Diophantine equations:

Equations of polynomials where solutions must be integers.

Example: $3x^2 - 2xy - y^2z = 7$ solution: $x = 1,\ y = 2,\ z = -2$

Let $D = \{p |$ polynomial $p(x_1, x_2, \ldots, x_k) = 0$ has a solution in integers)

Hilbert's 10^{th} problem: Give an algorithm to decide D.

Matiyasevich proved in 1970: D is not decidable.

Note: D is T-recognizable.

David Hilbert
1862—1943

Notation for encodings and TMs

Notation for encoding objects into strings

- If O is some object (e.g., polynomial, automaton, graph, etc.), we write $\langle O \rangle$ to be an encoding of that object into a string.

- If $O_1, O_2, \ldots, O_k$ is a list of objects then we write $\langle O_1, O_2, \ldots, O_k \rangle$ to be an encoding of them together into a single string.

Notation for writing Turing machines

We will use high-level English descriptions of algorithms when we describe TMs, knowing that we could (in principle) convert those descriptions into states, transition function, etc. Our notation for writing a TM M is

$M =$ "On input w

[English description of the algorithm]"

Check-in 6.3

If x and y are strings, would xy be a good choice for their encoding $\langle x, y \rangle$ into a single string?

a) Yes.

b) No.

Check-in 6.3

TM – example revisited

TM M recognizing $B = \{\mathrm{a}^k\mathrm{b}^k\mathrm{c}^k \mid k \geq 0\}$

$M =$ “On input w

1. Check if $w \in \mathrm{a}^*\mathrm{b}^*\mathrm{c}^*$, *reject* if not.
2. Count the number of a’s, b’s, and c’s in w.
3. *Accept* if all counts are equal; *reject* if not.”

High-level description is ok.
You do not need to manage tapes, states, etc…

TMs and Encodings – review

A TM has 3 possible outcomes for each input w:

1. *Accept* w (enter q_{acc})
2. *Reject* w by halting (enter q_{rej})
3. *Reject* w by looping (running forever)

A is T-recognizable if $A = L(M)$ for some TM M.

A is T-decidable if $A = L(M)$ for some TM decider M.

halts on all inputs

$\langle O_1, O_2, \dots, O_k \rangle$ encodes objects $O_1, O_2, \dots, O_k$ as a single string.

Notation for writing a TM M is

$M =$ "On input w

[English description of the algorithm]"

Acceptance Problem for DFAs

Let $A_{\mathrm{DFA}} = \{\langle B, w\rangle \mid B \text{ is a DFA and } B \text{ accepts } w\}$

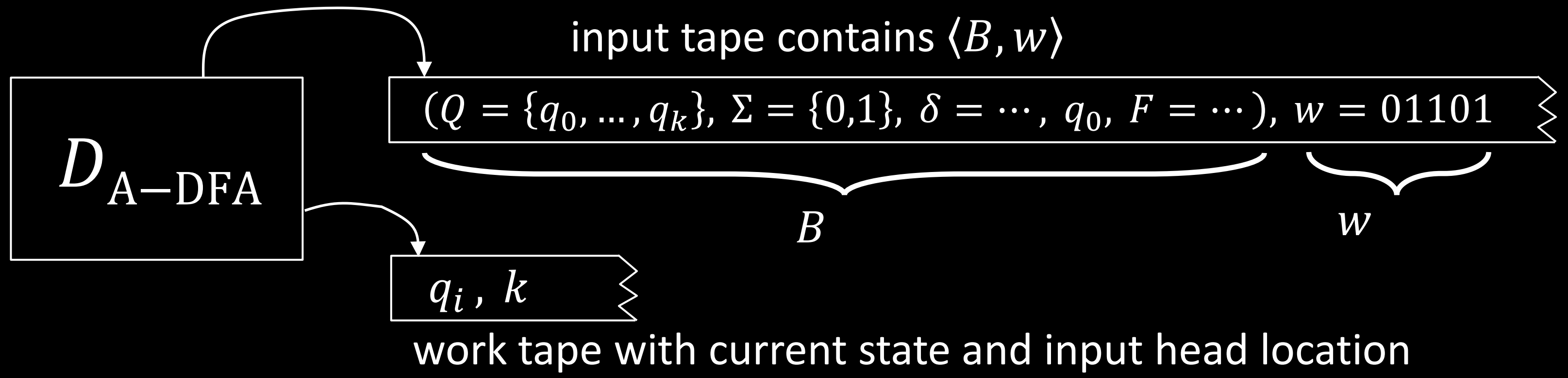
Theorem: A_{DFA} is decidable

Proof: Give TM $D_{\mathrm{A-DFA}}$ that decides A_{DFA}.

$D_{\mathrm{A-DFA}}$ = "On input s

1. Check that s has the form $\langle B, w\rangle$ where B is a DFA and w is a string; *reject* if not.
2. Simulate the computation of B on w.
3. If B ends in an accept state then *accept*. If not then *reject*."

Shorthand:
On input $\langle B, w\rangle$



Acceptance Problem for NFAs

Let $A_{\text{NFA}} = \{\langle B, w\rangle \mid B \text{ is a NFA and } B \text{ accepts } w\}$

Theorem: A_{NFA} is decidable

Proof: Give TM $D_{\text{A-NFA}}$ that decides A_{NFA}.

$D_{\text{A-NFA}} =$ "On input $\langle B, w\rangle$

1. Convert NFA B to equivalent DFA B'.
2. Run TM $D_{\text{A-DFA}}$ on input $\langle B', w\rangle$. [Recall that $D_{\text{A-DFA}}$ decides A_{DFA}]
3. *Accept* if $D_{\text{A-DFA}}$ accepts.
   Reject if not."

New element: Use conversion construction and previously constructed TM as a subroutine.

Emptiness Problem for DFAs

Let $E_{\text{DFA}} = \{\langle B\rangle \mid B \text{ is a DFA and } L(B) = \emptyset\}$

Theorem: E_{DFA} is decidable

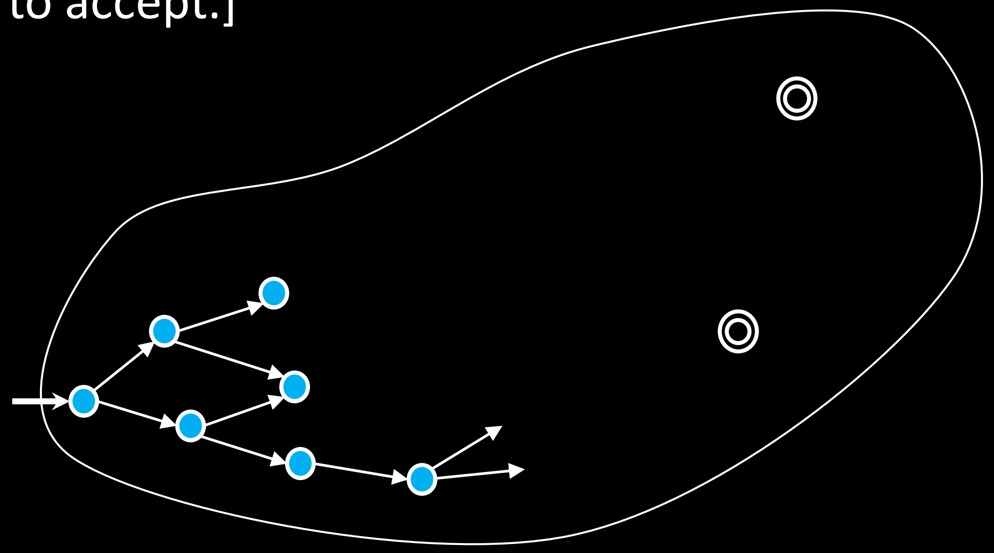
Proof: Give TM $D_{\text{E-DFA}}$ that decides E_{DFA} .

$D_{\text{E-DFA}}$ = "On input $\langle B\rangle$ [IDEA: Check for a path from start to accept.]

1. Mark start state.
2. Repeat until no new state is marked:

   Mark every state that has an incoming arrow from a previously marked state.

3. *Accept* if no accept state is marked.
   Reject if some accept state is marked."



Equivalence problem for DFAs

Let $EQ_{\mathrm{DFA}} = \{\langle A, B\rangle | \ A \text{ and } B \text{ are DFAs and } L(A) = L(B)\}$

Theorem: EQ_{DFA} is decidable

Proof: Give TM $D_{\mathrm{EQ-DFA}}$ that decides EQ_{DFA} .

Check-in 7.1

Let $EQ_{\mathrm{REX}} = \{\langle R_1, R_2\rangle | \ R_1 \text{ and } R_2 \text{ are regular expressions and } L(R_1) = L(R_2)\}$

Can we now conclude that EQ_{REX} is decidable?

a) Yes, it follows immediately from things we've already shown.

b) Yes, but it would take significant additional work.

c) No, intersection is not a regular operation.

Acceptance Problem for CFGs

Let $A_{\text{CFG}} = \{\langle G, w\rangle |\ G$ is a CFG and $w \in L(G)\}$

Theorem: A_{CFG} is decidable

Proof: Give TM $D_{\text{A-CFG}}$ that decides A_{CFG} .

$D_{\text{A-CFG}} =$ "On input $\langle G, w\rangle$

1. Convert G into CNF.
2. Try all derivations of length $2|w| - 1$.
3. *Accept* if any generate w.
   Reject if not.

Recall Chomsky Normal Form (CNF) only allows rules:

A → BC
B → b

Lemma 1: Can convert every CFG into CNF.
Proof and construction in book.

Lemma 2: If H is in CNF and $w \in L(H)$ then every derivation of w has $2|w| - 1$ steps.
Proof: exercise.

Check-in 7.2

Can we conclude that A_{PDA} is decidable?

a) Yes.
b) No, PDAs may be nondeterministic.
c) No, PDAs may not halt.

Check-in 7.2

Emptiness Problem for CFGs

Let $E_{\text{CFG}} = \{\langle G\rangle |\ G$ is a CFG and $L(G) = \emptyset\ \}$

Theorem: E_{CFG} is decidable

Proof:

$D_{\text{E-CFG}} =$ "On input $\langle G\rangle$ [IDEA: work backwards from terminals]

1. Mark all occurrences of terminals in G.
2. Repeat until no new variables are marked
   Mark all occurrences of variable A if
   $\mathrm{A} \to \mathrm{B}_1\mathrm{B}_2 \cdots \mathrm{B}_k$ is a rule and all B_i were already marked.
3. *Reject* if the start variable is marked.
   Accept if not."

S → RTa
R → Tb
T → a

Equivalence Problem for CFGs

Let $EQ_{\mathrm{CFG}} = \{\langle G, H\rangle |\ G, H$ are CFGs and $L(G) = L(H)\ \}$

Theorem: EQ_{CFG} is NOT decidable

Proof: Next week.

Let $AMBIG_{\mathrm{CFG}} = \{\langle G\rangle |\ G$ is an ambiguous CFG $\}$

Check-in 7.3

Why can't we use the same technique we used to show EQ_{DFA} is decidable to show that EQ_{CFG} is decidable?

a) Because CFGs are generators and DFAs are recognizers.

b) Because CFLs are closed under union.

c) Because CFLs are not closed under complementation and intersection.

Check-in 7.3

Acceptance Problem for TMs

Let $A_{\mathrm{TM}} = \{\langle M, w\rangle \mid M \text{ is a TM and } M \text{ accepts } w\}$

Theorem: A_{TM} is not decidable
Proof: Thursday.

Theorem: A_{TM} is T-recognizable
Proof: The following TM U recognizes A_{TM}

$U =$ "On input $\langle M, w\rangle$

1. Simulate M on input w.
2. *Accept* if M halts and accepts.
3. *Reject* if M halts and rejects.
4. ~~*Reject* if M never halts.~~" Not a legal TM action.

Turing's original "Universal Computing Machine"

Von Neumann said U inspired the concept of a stored program computer.

Recall: Acceptance Problem for TMs

Let $A_{\mathrm{TM}} = \{\langle M, w\rangle \mid M \text{ is a TM and } M \text{ accepts } w\}$

Today's Theorem: A_{TM} is not decidable

Proof uses the diagonalization method,
so we will introduce that first.

The Size of Infinity

How to compare the relative sizes of infinite sets?

Cantor (~1890s) had the following idea.

Defn: Say that set A and B <u>have the same size</u> if there is
a $\underbrace{\text{one-to-one}}_{\substack{x \neq y \rightarrow \\ f(x) \neq f(y) \\ \text{“injective”}}}$ and $\underbrace{\text{onto}}_{\substack{\text{Range } (f) = B \\ \text{“surjective”}}}$ function $f: A \rightarrow B$

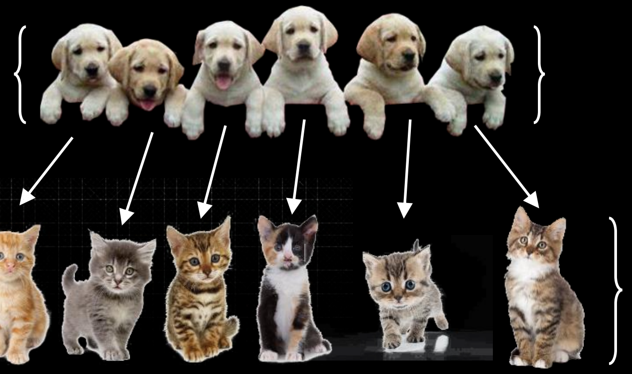
We call such an f a <u>1-1 correspondence</u>

Informally, two sets have the same size if we can pair up their members.

This definition works for finite sets.

Apply it to infinite sets too.

$f:$



Countable Sets

Let $\mathbb{N} = \{1,2,3,\dots\}$ and let $\mathbb{Z} = \{\dots,-2,-1,0,1,2,\dots\}$

Show $\mathbb{N}$ and $\mathbb{Z}$ have the same size

n	$f(n)$
$\mathbb{N}$	$\mathbb{Z}$

Let $\mathbb{Q}^+ = \{\, {}^{m}/_{n} \mid m,n \in \mathbb{N}\}$

Show $\mathbb{N}$ and $\mathbb{Q}^+$ have the same size

$\mathbb{Q}^+$	1	2	3	4	…
1	1/1	1/2	1/3	1/4	
2	2/1	2/2	2/3	2/4	…
3	3/1	3/2	3/3	3/4	
4	4/1	4/2	4/3	4/4	
⋮		⋮			

n	$f(n)$
$\mathbb{N}$	$\mathbb{Q}^+$

Defn: A set is <u>countable</u> if it is finite or it has the same size as $\mathbb{N}$.

Both $\mathbb{Z}$ and $\mathbb{Q}^+$ are countable.

$\mathbb{R}$ is Uncountable – Diagonalization

Let $\mathbb{R}$ = all real numbers (expressible by infinite decimal expansion)

Theorem: $\mathbb{R}$ is uncountable

Proof by contradiction via diagonalization: Assume $\mathbb{R}$ is countable

So there is a 1-1 correspondence $f: \mathbb{N} \to \mathbb{R}$

n	$f(n)$
1	
2	
3	
4	
5	
6	
7	
$\vdots$	

Diagonalization

Demonstrate a number $x \in \mathbb{R}$ that is missing from the list.

$$x = 0\,.$$

differs from the n^{th} number in the n^{th} digit
so cannot be the n^{th} number for any n.

Hence x is not paired with any n. It is missing from the list.

Therefore f is not a 1-1 correspondence.

$\mathbb{R}$ is Uncountable – Corollaries

Let $\mathcal{L}$ = all languages

Corollary 1: $\mathcal{L}$ is uncountable

Proof: There's a 1-1 correspondence from $\mathcal{L}$ to $\mathbb{R}$ so they are the same size.

Observation: $\Sigma^* = \{\varepsilon, 0,1,00,01,10,11,000, \ldots\}$ is countable.

Let $\mathcal{M}$ = all Turing machines
Observation: $\mathcal{M}$ is countable.
Because $\{\langle M \rangle |\ M$ is a TM$\} \subseteq \Sigma^*$.

Corollary 2: Some language is not decidable.

Because there are more languages than TMs.

We will show some specific language A_{TM} is not decidable.

Check-in 8.1

Hilbert's 1st question asked if there is a set of intermediate size between $\mathbb{N}$ and $\mathbb{R}$. Gödel and Cohen showed that we cannot answer this question by using the standard axioms of mathematics. How can we interpret their conclusion?

(a) We need better axioms to describe reality.

(b) Infinite sets have no mathematical reality so Hilbert's 1st question has no answer.

Check-in 8.1

A_{TM} is undecidable

Recall $A_{\mathrm{TM}} = \{\langle M, w\rangle \mid M$ is a TM and M accepts $w\}$

Theorem: A_{TM} is not decidable

Proof by contradiction: Assume some TM H decides A_{TM}.

So H on $\langle M, w\rangle = \begin{cases} \textit{Accept} & \text{if } M \text{ accepts } w \\ \textit{Reject} & \text{if not} \end{cases}$

Use H to construct TM D

$D =$ "On input $\langle M\rangle$

1. Simulate H on input $\langle M, \langle M\rangle\rangle$
2. *Accept* if H rejects. *Reject* if H accepts."

D accepts $\langle M\rangle$ iff M doesn't accept $\langle M\rangle$.

D accepts $\langle D\rangle$ iff D doesn't accept $\langle D\rangle$.

Contradiction.

Why is this proof a diagonalization?

All TMs ⇩	All TM descriptions: $\langle M_1\rangle$	$\langle M_2\rangle$	$\langle M_3\rangle$	$\langle M_4\rangle$	. . .	$\langle D\rangle$
M_1	acc					
M_2		rej				
M_3			acc			
M_4				acc		
$\vdots$						
D						

Check-in 8.2

Recall the Queue Automaton (QA) defined in Pset 2.
It is similar to a PDA except that it is deterministic
and it has a queue instead of a stack.

Let $A_{\mathrm{QA}} = \{\langle B, w\rangle |\ B$ is a QA and B accepts $w\}$

Is A_{QA} decidable?

(a) Yes, because QA are similar to PDA and A_{PDA} is decidable.

(b) No, because "yes" would contradict results we now know.

(c) We don't have enough information to answer this question.

$\overline{A_{\mathrm{TM}}}$ is T-unrecognizable

Theorem: If A and $\overline{A}$ are T-recognizable then A is decidable

Proof: Let TM M_1 and M_2 recognize A and $\overline{A}$.

Construct TM T deciding A.

T = "On input w

1. Run M_1 and M_2 on w in parallel until one accepts.
2. If M_1 accepts then *accept*.
   If M_2 accepts then *reject*."

Corollary: $\overline{A_{\mathrm{TM}}}$ is T-unrecognizable

Proof: A_{TM} is T-recognizable but also undecidable

Check-in 8.3

From what we've learned, which closure properties can we prove for the class of T-recognizable languages?

Choose all that apply.

(a) Closed under union.

(b) Closed under intersection.

(c) Closed under complement.

(d) Closed under concatenation.

(e) Closed under star.

The Reducibility Method

Use our knowledge that A_{TM} is undecidable to show other problems are undecidable.

Defn: $HALT_{\mathrm{TM}} = \{\langle M, w\rangle \mid M \text{ halts on input } w\}$

Theorem: $HALT_{\mathrm{TM}}$ is undecidable
Proof by contradiction, showing that A_{TM} is reducible to $HALT_{\mathrm{TM}}$:

Assume that $HALT_{\mathrm{TM}}$ is decidable and show that A_{TM} is decidable (false!).
Let TM R decide $HALT_{\mathrm{TM}}$.
Construct TM S deciding A_{TM}.

$S =$ "On input $\langle M, w\rangle$

1. Use R to test if M on w halts. If not, reject.
2. Simulate M on w until it halts (as guaranteed by R).
3. If M has accepted then *accept*.
   If M has rejected then *reject*.

TM S decides A_{TM}, a contradiction. Therefore $HALT_{\mathrm{TM}}$ is undecidable.

The Reducibility Method

If we know that some problem (say A_{TM}) is undecidable,
we can use that to show other problems are undecidable.

Defn: $HALT_{\mathrm{TM}} = \{\langle M, w\rangle \mid M \text{ halts on input } w\}$

Recall Theorem: $HALT_{\mathrm{TM}}$ is undecidable
Proof by contradiction, showing that A_{TM} is reducible to $HALT_{\mathrm{TM}}$:

Assume that $HALT_{\mathrm{TM}}$ is decidable and show that A_{TM} is decidable (false!).
Let TM R decide $HALT_{\mathrm{TM}}$.
Construct TM S deciding A_{TM}.

$S =$ "On input $\langle M, w\rangle$

1. Use R to test if M on w halts. If not, *reject*.
2. Simulate M on w until it halts (as guaranteed by R).
3. If M has accepted then *accept*.
   If M has rejected then *reject*.

TM S decides A_{TM}, a contradiction. Therefore $HALT_{\mathrm{TM}}$ is undecidable.

Reducibility – Concept

If we have two languages (or problems) A and B, then
A is reducible to B means that we can use B to solve A.

Example 1: Measuring the area of a rectangle
is reducible to measuring the lengths of its sides.

Example 2: We showed that A_{NFA} is reducible to A_{DFA} .

Example 3: From Pset 2, *PUSHER* is reducible to E_{CFG} .
(Idea- Convert push states to accept states.)

If A is reducible to B then solving B gives a solution to A.
- then B is easy $\rightarrow$ A is easy.
- then A is hard $\rightarrow$ B is hard.

this is the form we will use

Check-in 9.1

Is Biology reducible to Physics?

(a) Yes, all aspects of the physical world may be explained in terms of Physics, at least in principle.

(b) No, some things in the world, maybe life, the brain, or consciousness, are beyond the realm pf Physics.

(c) I'm on the fence on this question!

Check-in 9.1

E_{TM} is undecidable

Let $E_{\mathrm{TM}} = \{\langle M\rangle |\ M$ is a TM and $L(M) = \emptyset\}$

Theorem: E_{TM} is undecidable

Proof by contradiction. Show that A_{TM} is reducible to E_{TM}.

Assume that E_{TM} is decidable and show that A_{TM} is decidable (false!).
Let TM R decide E_{TM}.
Construct TM S deciding A_{TM}.

$S =$ "On input $\langle M, w\rangle$

1. Transform M to new TM $M_w =$ "On input x
    1. If $x \neq w$, *reject*.
    2. else run M on w
    3. *Accept* if M accepts."
2. Use R to test whether $L(M_w) = \emptyset$
3. If YES [so M rejects w] then *reject*.
   If NO [so M accepts w] then *accept*.

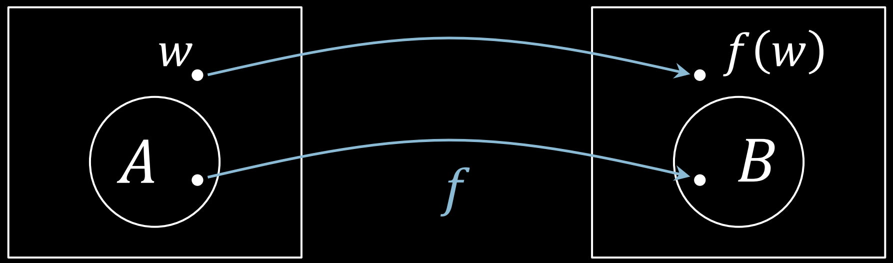
M_w works like M except that it always rejects strings x where $x \neq w$.

So $L(M_w) = \begin{cases} \{w\} & \text{if } M \text{ accepts } w \\ \emptyset & \text{if } M \text{ rejects } w \end{cases}$

Mapping Reducibility

Defn: Function $f: \Sigma^* \to \Sigma^*$ is <u>computable</u> if there is a TM F where F on input w halts with $f(w)$ on its tape, for all strings w.

Defn: <u>A is mapping-reducible to B</u> ($A \leq_{\mathrm{m}} B$) if there is a computable function f where $w \in A$ iff $f(w) \in B$.



Example: $A_{\mathrm{TM}} \leq_{\mathrm{m}} \overline{E_{\mathrm{TM}}}$

The computable reduction function f is $f(\langle M, w\rangle) = \langle M_w \rangle$

Because $\langle M, w\rangle \in A_{\mathrm{TM}}$ iff $\langle M_w \rangle \in \overline{E_{\mathrm{TM}}}$

(M accepts w iff $L(\langle M_w \rangle) \neq \emptyset$)

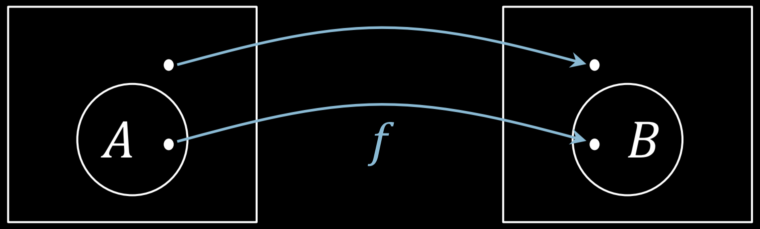
Recall TM M_w = "On input x
1. If $x \neq w$, *reject*.
2. else run M on w
3. *Accept* if M accepts."

Mapping Reductions - properties

Theorem: If $A \leq_{\mathrm{m}} B$ and B is decidable then so is A

Proof: Say TM R decides B.

Construct TM S deciding A:



S = "On input w

1. Compute $f(w)$
2. Run R on $f(w)$ to test if $f(w) \in B$
3. If R halts then output same result."

Corollary: If $A \leq_{\mathrm{m}} B$ and A is undecidable then so is B

Theorem: If $A \leq_{\mathrm{m}} B$ and B is T-recognizable then so is A

Proof: Same as above.

Corollary: If $A \leq_{\mathrm{m}} B$ and A is T-unrecognizable then so is B

Check-in 9.2

Suppose $A \leq_{\mathrm{m}} B$.

What can we conclude?

Check all that apply.

(a) $B \leq_{\mathrm{m}} A$

(b) $\overline{A} \leq_{\mathrm{m}} \overline{B}$

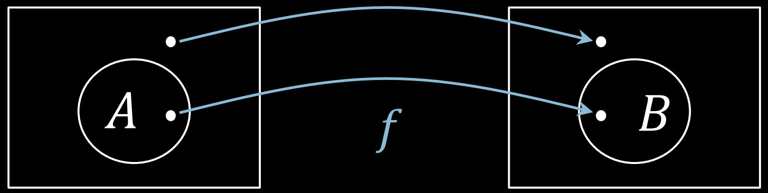
(c) None of the above

Check-in 9.2

Mapping vs General Reducibility

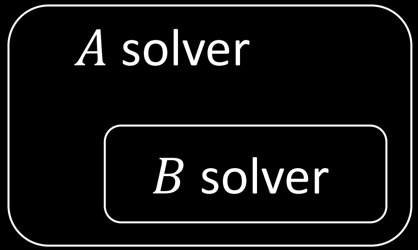
Mapping Reducibility of A to B: Translate A-questions to B-questions.

- A special type of reducibility
- Useful to prove T-unrecognizability



(General) Reducibility of A to B: Use B solver to solve A.

- May be conceptually simpler
- Useful to prove undecidability



Noteworthy difference:

- A is reducible to $\overline{A}$
- A may not be mapping reducible to $\overline{A}$.

  For example $\overline{A_{\mathrm{TM}}} \not\leq_{\mathrm{m}} A_{\mathrm{TM}}$

Check-in 9.3

We showed that if $A \leq_{\mathrm{m}} B$ and B is T-recognizable then so is A.

Is the same true if we use general reducibility instead of mapping reducibility?

(a) Yes

(b) No

Check-in 9.3

Reducibility – Templates

To prove B is undecidable:

- Show undecidable A is reducible to B. (often A is A_{TM})
- Template: Assume TM R decides B.
  Construct TM S deciding A. Contradiction.

To prove B is T-unrecognizable:

- Show T-unrecognizable A is mapping reducible to B. (often A is $\overline{A_{\mathrm{TM}}}$)
- Template: give reduction function f.

E_{TM} is T-unrecognizable

Recall $E_{\text{TM}} = \{\langle M\rangle \mid M \text{ is a TM and } L(M) = \emptyset\}$

Theorem: E_{TM} is T-unrecognizable

Proof: Show $\overline{A_{\text{TM}}} \leq_{\text{m}} E_{\text{TM}}$

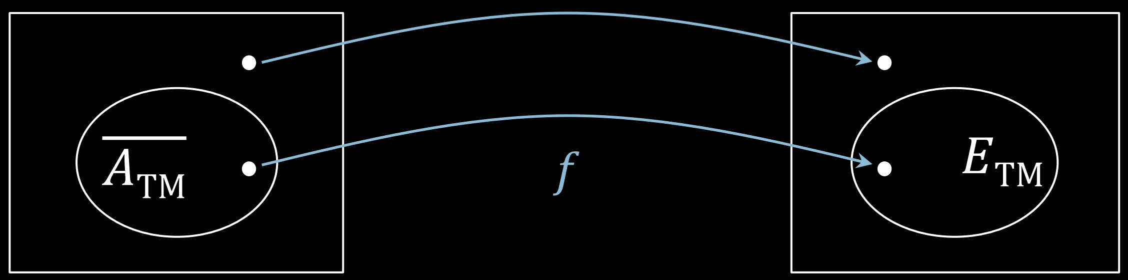
Reduction function: $f(\langle M, w\rangle) = \langle M_w\rangle$

Explanation: $\langle M, w\rangle \in \overline{A_{\text{TM}}}$ iff $\langle M_w\rangle \in E_{\text{TM}}$

M rejects w iff $L(\langle M_w\rangle) = \emptyset$

Recall TM M_w = "On input x

1. If $x \neq w$, *reject*.
2. else run M on w
3. *Accept* if M accepts."



EQ_{TM} and $\overline{EQ_{\text{TM}}}$ are T-unrecognizable

$EQ_{\text{TM}} = \{\langle M_1, M_2\rangle |\ M_1$ and M_2 are TMs and $L(M_1) = L(M_2)\ \}$

Theorem: Both EQ_{TM} and $\overline{EQ_{\text{TM}}}$ are T-unrecognizable

Proof: (1) $\overline{A_{\text{TM}}} \leq_{\text{m}} EQ_{\text{TM}}$

(2) $\overline{A_{\text{TM}}} \leq_{\text{m}} \overline{EQ_{\text{TM}}}$

For any w let $T_w =$ "On input x

1. Ignore x.
2. Simulate M on w."

T_w acts on all inputs the way M acts on w.

(1) Here we give f which maps $\overline{A_{\text{TM}}}$ problems (of the form $\langle M, w\rangle$) to EQ_{TM} problems (of the form $\langle T_1, T_2\rangle$).

$f(\langle M, w\rangle) = \langle T_w, T_{\text{reject}}\rangle$ $\qquad T_{\text{reject}}$ is a TM that always rejects.

(2) Similarly $f(\langle M, w\rangle) = \langle T_w, T_{\text{accept}}\ \rangle$ $\qquad T_{\text{accept}}$ always accepts.

Reducibility terminology

Why do we use the term "reduce"?

When we reduce A to B, we show how to solve A by using B
and conclude that A is no harder than B. (suggests the $\leq_{\mathrm{m}}$ notation)

Possibility 1: We bring A's difficulty down to B's difficulty.

Possibility 2: We bring B's difficulty up to A's difficulty.

Remember

To prove some language B is undecidable, show that A_{TM} (or any known undecidable language) is reducible to B.

Revisit Hilbert's 10th Problem

Recall $D = \{\langle p\rangle |$ polynomial $p(x_1, x_2, \dots, x_k) = 0$ has integer solution)

Hilbert's 10th problem (1900): Is D decidable?

Theorem (1971): No

Proof: Show A_{TM} is reducible to D. [would take entire semester]

Do toy problem instead which has a similar proof method.

Toy problem: The Post Correspondence Problem.

Method: The Computation History Method.

Post Correspondence Problem

Given a collection of pairs of strings as dominoes:

$$P = \left\{ \left[\frac{t_1}{b_1} \right], \left[\frac{t_2}{b_2} \right], \ldots, \left[\frac{t_k}{b_k} \right] \right\}$$

a <u>match</u> is a finite sequence of dominos in P (repeats allowed) where the concatenation of the t's = the concatenation of the b's.

Match = $\left[\frac{t_{i_1}}{b_{i_1}} \right] \left[\frac{t_{i_2}}{b_{i_2}} \right] \cdots \left[\frac{t_{i_l}}{b_{i_l}} \right]$ where $t_{i_1} t_{i_2} \cdots t_{i_l} = b_{i_1} b_{i_2} \cdots b_{i_l}$

Example: $P = \left\{ \left[\frac{\text{ab}}{\text{aba}} \right], \left[\frac{\text{aa}}{\text{aba}} \right], \left[\frac{\text{ba}}{\text{aa}} \right], \left[\frac{\text{abab}}{\text{b}} \right] \right\}$

Match: ✓

Check-in 10.1

Let $P_1 = \left\{ \left[\frac{\text{aa}}{\text{aaba}} \right], \left[\frac{\text{ba}}{\text{ab}} \right], \left[\frac{\text{ab}}{\text{ba}} \right] \right\}$

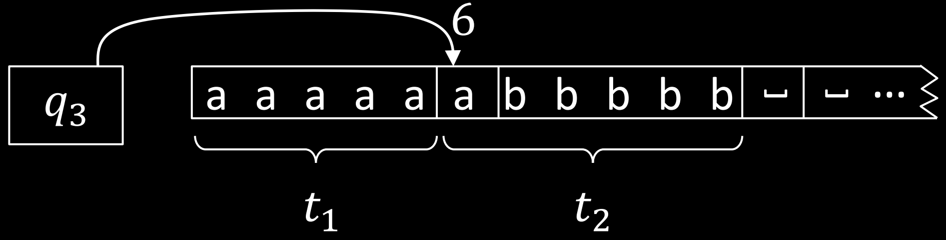
Does P_1 have a match?

(a) Yes.

(b) No.

TM Configurations

Defn: A configuration of a TM is a triple (q, p, t) where
q = the state,
p = the head position,
t = tape contents
representing a snapshot of the TM at a point in time.



Configuration: $(q_3, 6, \text{aaaaaabbbbb})$
Encoding as a string: aaaaa$q_3$abbbbb

Encode configuration (q, p, t) as the string $t_1 q t_2$ where
$t = t_1 t_2$ and the head position is on the first symbol of t_2.

TM Computation Histories

Defn: An <u>(accepting) computation history</u> for TM M on input w is a sequence of configurations $C_1, C_2, \dots, C_{\text{accept}}$ that M enters until it accepts.

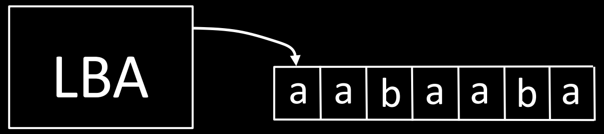
Encode a computation history $C_1, C_2, \dots, C_{\text{accept}}$ as the string $C_1 \# C_2 \# \cdots \# C_{\text{accept}}$ where each configuration C_i is encoded as a string.

A computation history for M on $w = w_1 w_2 \cdots w_n$.
Here say $\delta(q_0, w_1) = (q_7, \text{a}, \text{R})$ and $\delta(q_7, w_2) = (q_8, \text{c}, R)$.

$$\overbrace{q_0 w_1 w_2 \cdots w_n}^{C_1} \;\#\; \overbrace{\text{a} q_7 w_2 \cdots w_n}^{C_2} \;\#\; \overbrace{\text{ac} q_8 w_3 \cdots w_n}^{C_3} \;\#\; \cdots \;\#\; \overbrace{\cdots\; q_{\text{accept}} \;\cdots}^{C_{\text{accept}}}$$

Linearly Bounded Automata

Defn: A linearly bounded automaton (LBA) is a 1-tape TM that cannot move its head off the input portion of the tape.



Tape size adjusts to length of input.

Let $A_{\text{LBA}} = \{\langle B, w\rangle |$ LBA B accepts w $\}$

Theorem: A_{LBA} is decidable

Proof: (idea) If B on w runs for long, it must be cycling.

Claim: For inputs of length n, an LBA can have only $|Q| \times n \times |\Gamma|^n$ different configurations.

Therefore, if an LBA runs for longer, it must repeat some configuration and thus will never halt.

Decider for A_{LBA}:

$D_{\text{A-LBA}} =$ "On input $\langle B, w\rangle$

1. Let $n = |w|$.
2. Run B on w for $|Q| \times n \times |\Gamma|^n$ steps.
3. If has accepted, *accept*.
4. If it has rejected or $\underbrace{\text{is still running}}_{\text{must be looping}}$, *reject*."

E_{LBA} is undecidable

Let $E_{\text{LBA}} = \{\langle B \rangle \mid B \text{ is an LBA and } L(B) = \emptyset\}$

Theorem: E_{LBA} is undecidable

Proof: Show A_{TM} is reducible to E_{LBA}. Uses the computation history method.

Assume that TM R decides E_{LBA}

Construct TM S deciding A_{TM}

$S =$ "on input $\langle M, w \rangle$

1. Construct LBA $B_{M,w}$ which tests whether its input x is an accepting computation history for M on w, and only accepts x if it is.
2. Use R to determine whether $L(B_{M,w}) = \emptyset$.
3. *Accept* if no. *Reject* if yes."

$B_{M,w}$

$$\underbrace{q_0 w_1 w_2 \cdots w_n}_{C_1} \ \# \ \underbrace{\mathrm{a} q_7 w_2 \cdots w_n}_{C_2} \ \# \ \underbrace{\mathrm{ac} q_8 w_3 \cdots w_n}_{C_3} \ \# \ \cdots \ \# \ \underbrace{\cdots \ q_{\text{accept}} \ \cdots}_{C_{\text{accept}}}$$

N

Check-in 10.2

What do you think of the Computation History Method? Check all that apply.

(a) Cool !

(b) Just another theorem.

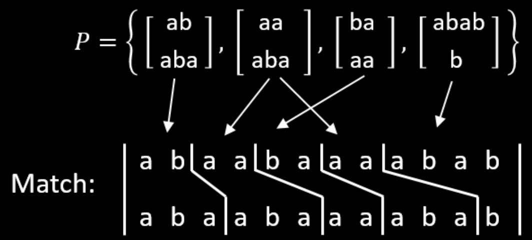
(c) I'm baffled.

(d) I wish I was in 6.046.

Check-in 10.2

PCP is undecidable

Recall $PCP = \{\langle P \rangle |\ P$ has a match $\}$



Theorem: PCP is undecidable

Proof: Show A_{TM} is reducible to PCP. Uses the computation history method.

Technical assumption: Match must start with $\begin{bmatrix} t_1 \\ b_1 \end{bmatrix}$. Can fix this assumption.

Assume that TM R decides PCP

Construct TM S deciding A_{TM}

$S =$ "on input $\langle M, w \rangle$

1. Construct PCP instance $P_{M,w}$ where a match corresponds to a computation history for M on w.
2. Use R to determine whether $P_{M,w}$ has a match.
3. *Accept* if yes. *Reject* if no."

Constructing $P_{M,w}$

Make $P_{M,w}$ where a match is a computation history for M on w.

$$\begin{bmatrix} u_1 \\ v_1 \end{bmatrix} = \begin{bmatrix} \# \\ \#q_0 w_1 \cdots w_n \# \end{bmatrix}$$ (starting domino)

For each $a, b \in \Gamma$ and $q, r \in Q$ where $\delta(q, a) = (r, b, \mathrm{R})$

put $\begin{bmatrix} q & a \\ b & r \end{bmatrix}$ in $P_{M,w}$

(Handles right moves. Similar for left moves.)

Ending dominos to allow a match if M accepts:

$$\begin{bmatrix} a\ q_{\text{accept}} \\ q_{\text{accept}} \end{bmatrix} \quad \begin{bmatrix} q_{\text{accept}}\ a \\ q_{\text{accept}} \end{bmatrix}$$

Check-in 10.3

What else can we now conclude?

Choose all that apply.

(a) PCP is T-unrecognizable.

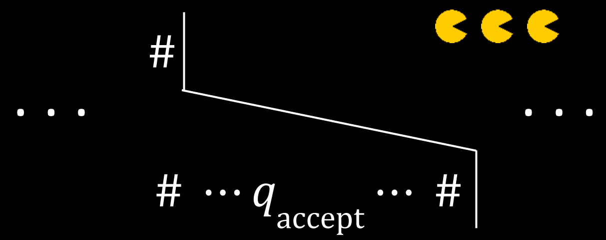
(b) $\overline{PCP}$ is T-unrecognizable.

(c) Neither of the above.

Illustration:

$w = 223$

$\delta(q_0, 2) = (q_7, 4, \mathrm{R})$



Match completed!

… one detail needed.

Check-in 10.3

ALL_{CFG} is undecidable

Let $ALL_{\mathrm{CFG}} = \{\langle G \rangle \mid G$ is a CFG and $L(G) = \Sigma^*\}$

Theorem: ALL_{CFG} is undecidable

Proof: Show A_{TM} is reducible to ALL_{PDA} via the computation history method.

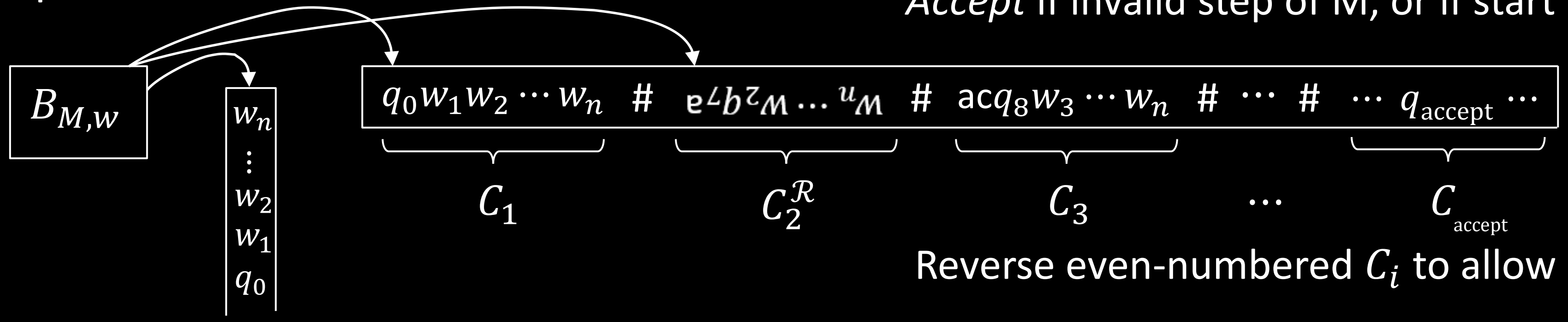
Assume TM R decides ALL_{PDA} and construct TM S deciding A_{TM}.

$S =$ "On input $\langle M, w \rangle$

1. Construct PDA $B_{M,w}$ which tests whether its input x is an accepting computation history for M on w, and only accepts x if it is NOT.
2. Use R to determine whether $L(B_{M,w}) = \Sigma^*$.
3. *Accept* if no. *Reject* if yes."

Nondeterministically push some C_i and pop to compare with C_{i+1}.
Accept if invalid step of M, or if start wrong, or if end isn't accepting.

$B_{M,w}$ operation:



Reverse even-numbered C_i to allow comparing with C_{i+1} via stack.

Computation History Method - recap

Computation History Method is useful for showing the undecidability of problems involving testing for the existence of some object.

D Is there an integral solution (to the polynomial equation)?

E_{LBA} Is there some accepted string (for the LBA)?

PCP Is there a match (for the given dominos)?

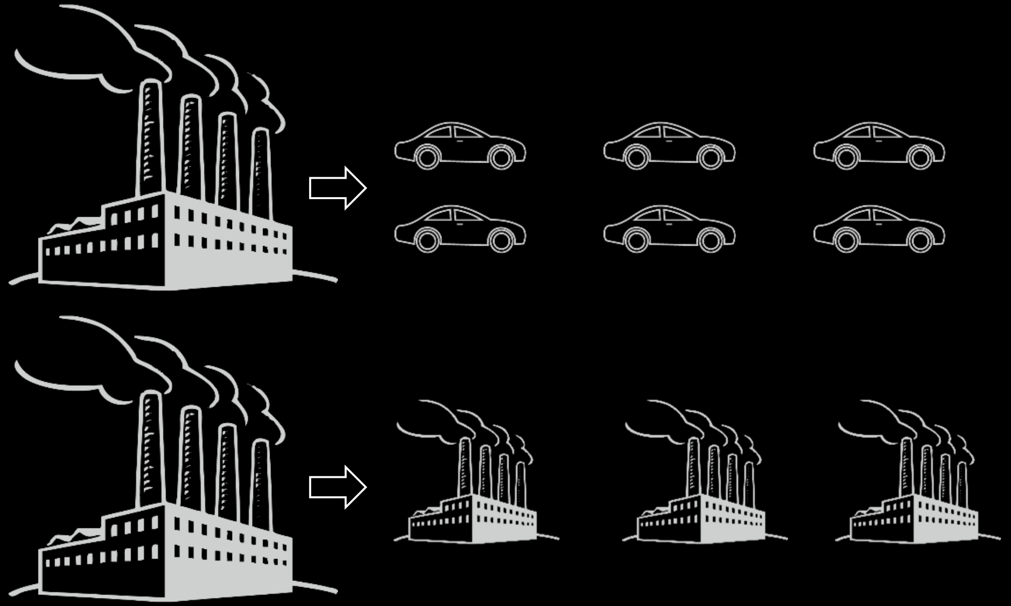
ALL_{CFG} Is there some rejected string (for the CFG)?

In each case, the object is the computation history in some form.

Self-reproduction Paradox

Suppose a Factory makes Cars
- Complexity of Factory > Complexity of Car
  (because Factory needs instructions for Car + robots, tools, …)

Can a Factory make Factories?
- Complexity of Factory > Complexity of Factory?
- Seems impossible to have a self-reproducing machine



But, living things self-reproduce

How to resolve this paradox?

Self-reproducing machines are possible!

A Self-Reproducing TM

Theorem: There is a TM $SELF$ which (on any input) halts with $\langle SELF \rangle$ on the tape.

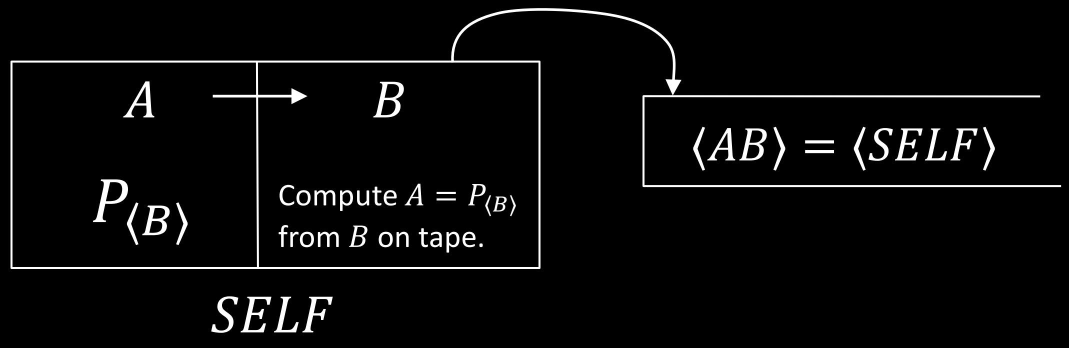
Lemma: There is a computable function $q: \Sigma^* \to \Sigma^*$ such that $q(w) = \langle P_w \rangle$ for every w, where P_w is the TM $P_w =$ “Print w on the tape and halt”.
Proof: Straightforward.

Proof of Theorem: $SELF$ has two parts, A and B.

$A = P_{\langle B \rangle}$

$B = P_{\langle A \rangle}$?

$B =$ “1. Compute q(tape contents) to get A.
2. Combine with B to get $AB = SELF$.
3. Halt with $\langle SELF \rangle$ on tape.”



Can implement in any programming language.

English Implementation

Check-in 11.1

Implementations of the Recursion Theorem have two parts, a <u>Template</u> and an <u>Action</u>. In the TM and English implementations, which is the <u>Action</u> part?

(a) A and the upper phrase

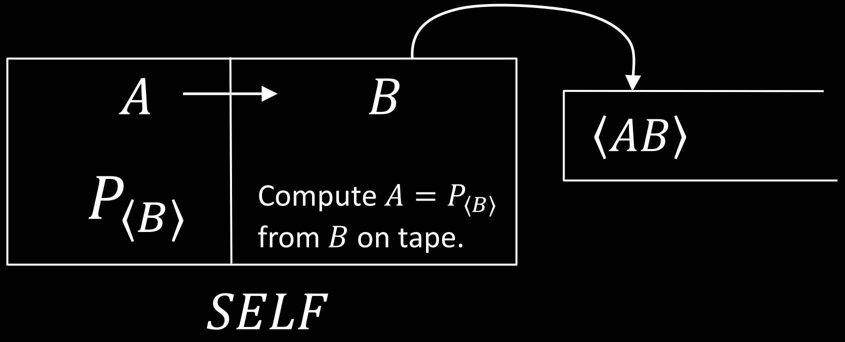
(b) A and the lower phrase

(c) B and the upper phrase

(d) B and the lower phrase.

Write the following twice, the second time in quotes
"Write the following twice, the second time in quotes"

Write the following twice, the second time in quotes
"Write the following twice, the second time in quotes"



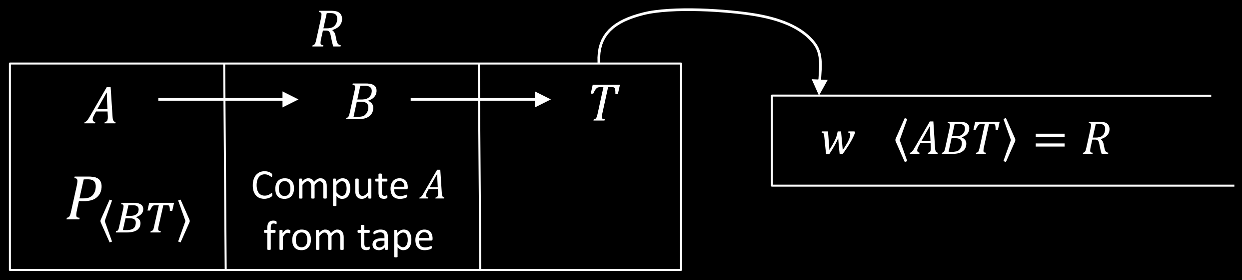
Note on Pset Problem 6: Don't need to worry about quoting.

The Recursion Theorem

A compiler which implements “compute your own description” for a TM.

Theorem: For any TM T there is a TM R where for all w
R on input w operates in the same way as T on input $\langle w, R\rangle$.

Proof of Theorem: R has three parts: A, B, and T.
T is given
$A = P_{\langle BT\rangle}$
$B =$ “1. Compute q(tape contents after w) to get A.
2. Combine with BT to get $ABT = R$.
3. Pass control to T on input $\langle w, R\rangle$.”



Check-in 11.2

Can we use the Recursion Theorem to design a TM T where $L(T) = \{\langle T\rangle\}$?

(a) Yes.

(b) No.

Moral: You can use “compute your own description” in describing TMs.

Check-in 11.2

Ex 1: A_{TM} is undecidable - new proof

Theorem: A_{TM} is not decidable
Proof by contradiction: Assume some TM H decides A_{TM}.

Consider the following TM R:

$R =$ "On input w

1. Get own description $\langle R \rangle$.
2. Use H on input $\langle R, w \rangle$ to determine whether R accepts w.
3. Do the opposite of what H says."

Ex 2: Fixed-point Theorem

Theorem: For any computable function $f : \Sigma^* \rightarrow \Sigma^*$,
there is a TM R such that $L(R) = L(S)$ where $f(\langle R \rangle) = \langle S \rangle$.

In other words, consider f to be a program transformation function.
Then for some program R, its behavior is unchanged by f.

Proof: Let R be the following TM.

$R =$ "On input w

1. Get own description $\langle R \rangle$.
2. Compute $f(\langle R \rangle)$ and call the result $\langle S \rangle$.
3. Simulate S on w."

Ex 3: MIN_{TM} is T-unrecognizable

Defn: M is a minimal TM if $|\langle M'\rangle| < |\langle M\rangle| \rightarrow L(M') \neq L(M)$.
Thus, a minimal TM has the shortest description among all equivalent TMs.

Let $MIN_{\mathrm{TM}} = \{\langle M\rangle |\ M$ is a minimal TM $\}$.
Theorem: MIN_{TM} is T-unrecognizable.
Proof by contradiction: Assume some TM E enumerates

Consider the following TM R:
$R =$ "On input w
1. Get own description $\langle R\rangle$.
2. Run enumerator E until some TM B appears, where $|\langle R\rangle| < |\langle B\rangle|$.
3. Simulate B on w."

Thus $L(R) = L(B)$ and $|\langle R\rangle| < |\langle B\rangle|$ so B isn't minimal, but $\langle B\rangle \in L(E)$, contradiction.

Check-in 11.3

Let A be an infinite subset of MIN_{TM} .
Is it possible that A is T-recognizable?

(a) Yes.

(b) No.

Check-in 11.3

Other applications

1. Computer viruses.
2. A true but unprovable mathematical statement due to Kurt Gödel:
   "This statement is unprovable."

Intro to Mathematical Logic

Goal: A mathematical study of mathematical reasoning itself.

Formally defines the language of mathematics, mathematical truth, and provability.

Gödel's First Incompleteness Theorem:

In any reasonable formal system, some true statements are not provable.

Proof: We use two properties of formal proofs:

1) Soundness: If ϕ has a proof π then ϕ is true.
2) Checkability: The language $\{\langle \pi, \phi \rangle |\ \pi$ is a proof of statement $\phi\}$ is decidable.

Checkability implies the set of provable statements $\{\langle \phi \rangle |\ \phi$ has a proof$\}$ is T-recognizable.

SImilarly, if we can always prove $\langle M, w \rangle \in \overline{A_{\mathrm{TM}}}$ when it is true, then $\overline{A_{\mathrm{TM}}}$ is T-recognizable (false!).

Therefore, some true statements of the form $\langle M, w \rangle \in \overline{A_{\mathrm{TM}}}$ are unprovable.

Next, we use the Recursion Theorem to give a specific example of a true but unprovable statement.

A True but Unprovable Statement

Implement Gödel statement "This statement is unprovable."

Let ϕ_U be the statement $\langle R, 0\rangle \in \overline{A_{\mathrm{TM}}}$ where R is the following TM:

$R =$ "On any input

1. Obtain $\langle R\rangle$ and use it to obtain ϕ_U.
2. For each possible proof $\pi = \pi_1, \pi_2, \ldots$

   Test if π is a proof that ϕ_U is true.

   If yes, then *accept*. Otherwise, continue."

Theorem: (1) ϕ_U has no proof

(2) ϕ_U is true

ϕ_U

Proof:

(1) If ϕ_U has a proof

(2) If ϕ_U is false