Week5 monday

Three different CFGs that each generate the language $\{abba\}$

$$(\{S,T,V,W\},\{a,b\},\{S\to aT, T\to bV, V\to bW, W\to a\},S)$$

$$(\{Q\},\{a,b\},\{Q\to abba\},Q)$$

$$(\{X,Y\},\{a,b\},\{X\to aYa, Y\to bb\},X)$$

Design a CFG to generate the language $\{a^nb^n \mid n\geq 0\}$

Sample derivation:

Design a CFG to generate the language $\{a^ib^j \mid j\geq i\geq 0\}$

Sample derivation:

Theorem 2.20: A language is generated by some context-free grammar if and only if it is recognized by some push-down automaton.

Definition: a language is called **context-free** if it is the language generated by a context-free grammar. The class of all context-free language over a given alphabet Σ is called **CFL**.

Consequences:

- Quick proof that every regular language is context free
- To prove closure of the class of context-free languages under a given operation, we can choose either of two modes of proof (via CFGs or PDAs) depending on which is easier
- To fully specify a PDA we could give its 6-tuple formal definition or we could give its input alphabet, stack alphabet, and state diagram. An informal description of a PDA is a step-by-step description of how its computations would process input strings; the reader should be able to reconstruct the state diagram or formal definition precisely from such a descripton. The informal description of a PDA can refer to some common modules or subroutines that are computable by PDAs:
  - PDAs can "test for emptiness of stack" without providing details. *How?* We can always push a special end-of-stack symbol, \$, at the start, before processing any input, and then use this symbol as a flag.
  - PDAs can "test for end of input" without providing details. *How?* We can transform a PDA to one where accepting states are only those reachable when there are no more input symbols.

Over $\Sigma = \{a, b\}$, let $L = \{a^n b^m \mid n \neq m\}$. **Goal**: Prove L is context-free.

Suppose L_1 and L_2 are context-free languages over Σ. **Goal**: $L_1 \cup L_2$ is also context-free.

Approach 1: with PDAs

Let $M_1 = (Q_1, \Sigma, \Gamma_1, \delta_1, q_1, F_1)$ and $M_2 = (Q_2, \Sigma, \Gamma_2, \delta_2, q_2, F_2)$ be PDAs with $L(M_1) = L_1$ and $L(M_2) = L_2$.

Define $M =$

Approach 2: with CFGs

Let $G_1 = (V_1, \Sigma, R_1, S_1)$ and $G_2 = (V_2, \Sigma, R_2, S_2)$ be CFGs with $L(G_1) = L_1$ and $L(G_2) = L_2$.

Define $G =$

Suppose L_1 and L_2 are context-free languages over Σ. **Goal**: $L_1 \circ L_2$ is also context-free.

Approach 1: with PDAs

Let $M_1 = (Q_1, \Sigma, \Gamma_1, \delta_1, q_1, F_1)$ and $M_2 = (Q_2, \Sigma, \Gamma_2, \delta_2, q_2, F_2)$ be PDAs with $L(M_1) = L_1$ and $L(M_2) = L_2$.

Define $M =$

Approach 2: with CFGs

Let $G_1 = (V_1, \Sigma, R_1, S_1)$ and $G_2 = (V_2, \Sigma, R_2, S_2)$ be CFGs with $L(G_1) = L_1$ and $L(G_2) = L_2$.

Define $G =$

Summary

Over a fixed alphabet Σ, a language L is **regular**

iff it is described by some regular expression
iff it is recognized by some DFA
iff it is recognized by some NFA

Over a fixed alphabet Σ, a language L is **context-free**

iff it is generated by some CFG
iff it is recognized by some PDA

Fact: Every regular language is a context-free language.

Fact: There are context-free languages that are not nonregular.

Fact: There are countably many regular languages.

Fact: There are countably inifnitely many context-free languages.

Consequence: Most languages are **not** context-free!

Examples of non-context-free languages

$$\{a^n b^n c^n \mid 0 \leq n, n \in \mathbb{Z}\}$$
$$\{a^i b^j c^k \mid 0 \leq i \leq j \leq k, i \in \mathbb{Z}, j \in \mathbb{Z}, k \in \mathbb{Z}\}$$
$$\{ww \mid w \in \{0,1\}^*\}$$

(Sipser Ex 2.36, Ex 2.37, 2.38)

There is a Pumping Lemma for CFL that can be used to prove a specific language is non-context-free: If A is a context-free language, there there is a number p where, if s is any string in A of length at least p, then s may be divided into five pieces $s = uvxyz$ where (1) for each $i \geq 0$, $uv^ixy^iz \in A$, (2) $|uv| > 0$, (3) $|vxy| \leq p$. *We will not go into the details of the proof or application of Pumping Lemma for CFLs this quarter.*

Week5 wednesday

A set X is said to be **closed** under an operation OP if, for any elements in X, applying OP to them gives an element in X.

True/False	Closure claim
True	The set of integers is closed under multiplication. $\forall x \forall y \, (\, (x \in \mathbb{Z} \land y \in \mathbb{Z}) \to xy \in \mathbb{Z} \,)$
True	For each set A, the power set of A is closed under intersection. $\forall A_1 \forall A_2 \, (\, (A_1 \in \mathcal{P}(A) \land A_2 \in \mathcal{P}(A) \in \mathbb{Z}) \to A_1 \cap A_2 \in \mathcal{P}(A) \,)$
	The class of regular languages over Σ is closed under complementation.
	The class of regular languages over Σ is closed under union.
	The class of regular languages over Σ is closed under intersection.
	The class of regular languages over Σ is closed under concatenation.
	The class of regular languages over Σ is closed under Kleene star.
	The class of context-free languages over Σ is closed under complementation.
	The class of context-free languages over Σ is closed under union.
	The class of context-free languages over Σ is closed under intersection.
	The class of context-free languages over Σ is closed under concatenation.
	The class of context-free languages over Σ is closed under Kleene star.

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

Σ^*	Regular	/	nonregular and context-free	/	not context-free
$\{0^i \# 1^j \mid i \geq 0, j \geq 0\}$	Regular	/	nonregular and context-free	/	not context-free
$\{0^i 1^j \# 1^j 0^i \mid i \geq 0, j \geq 0\}$	Regular	/	nonregular and context-free	/	not context-free
$\{0^i 1^j \# 0^i 1^j \mid i \geq 0, j \geq 0\}$	Regular	/	nonregular and context-free	/	not context-free

Turing machines: unlimited read + write memory, unlimited time (computation can proceed without "consuming" input and can re-read symbols of input)

- Division betweeen program (CPU, state diagram) and data
- Unbounded memory gives theoretical limit to what modern computation (including PCs, supercomputers, quantum computers) can achieve
- State diagram formulation is simple enough to reason about (and diagonalize against) while expressive enough to capture modern computation

For Turing machine $M = (Q, \Sigma, \Gamma, \delta, q_0, q_{accept}, q_{reject})$ the **computation** of M on a string w over Σ is:

- Read/write head starts at leftmost position on tape.
- Input string is written on $|w|$-many leftmost cells of tape, rest of the tape cells have the blank symbol. **Tape alphabet** is Γ with $\sqcup \in \Gamma$ and $\Sigma \subseteq \Gamma$. The blank symbol $\sqcup \notin \Sigma$.
- Given current state of machine and current symbol being read at the tape head, the machine transitions to next state, writes a symbol to the current position of the tape head (overwriting existing symbol), and moves the tape head L or R (if possible). Formally, **transition function** is

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

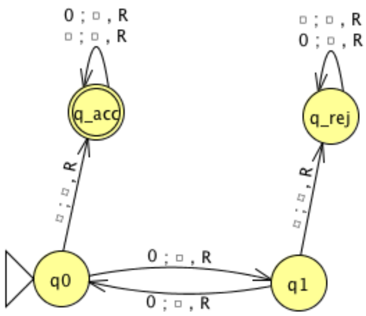
- Computation ends if and when machine enters either the accept or the reject state. This is called **halting**. Note: $q_{accept} \neq q_{reject}$.

The **language recognized by the Turing machine** M, is

$\{w \in \Sigma^* \mid \text{computation of } M \text{ on } w \text{ halts after entering the accept state}\} = \{w \in \Sigma^* \mid w \text{ is accepted by } M\}$

An example Turing machine: $\Sigma =$ $\qquad\qquad\qquad$ $, \Gamma =$

$$\delta((q0, 0)) =$$



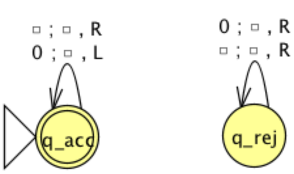
Formal definition:

Sample computation:

$q0 \downarrow$						
0	0	0	␣	␣	␣	␣

The language recognized by this machine is . . .

Extra practice:



Formal definition:

Sample computation:

Week4 monday

Recap so far: In DFA, the only memory available is in the states. Automata can only "remember" finitely far in the past and finitely much information, because they can have only finitely many states. If a computation path of a DFA visits the same state more than once, the machine can't tell the difference between the first time and future times it visits this state. Thus, if a DFA accepts one long string, then it must accept (infinitely) many similar strings.

Definition A positive integer p is a **pumping length** of a language L over Σ means that, for each string $s \in \Sigma^*$, if $|s| \geq p$ and $s \in L$, then there are strings x, y, z such that

$$s = xyz$$

and

$$|y| > 0, \qquad \text{for each } i \geq 0,\ xy^iz \in L, \qquad \text{and} \qquad |xy| \leq p.$$

Negation: A positive integer p is **not a pumping length** of a language L over Σ iff

$$\exists s \left(\ |s| \geq p \land s \in L \land \forall x \forall y \forall z \left(\ (s = xyz \land |y| > 0 \land |xy| \leq p\ ) \to \exists i (i \geq 0 \land xy^iz \notin L) \right) \ \right)$$

Informally:

Restating **Pumping Lemma**: If L is a regular language, then it has a pumping length.

Contrapositive: If L has no pumping length, then it is nonregular.

The Pumping Lemma *cannot* be used to prove that a language *is* regular.

The Pumping Lemma **can** be used to prove that a language *is not* regular.

Extra practice: Exercise 1.49 in the book.

Proof strategy: To prove that a language L is **not** regular,

- Consider an arbitrary positive integer p
- Prove that p is not a pumping length for L
- Conclude that L does not have *any* pumping length, and therefore it is not regular.

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

Fix p an arbitrary positive integer. List strings that are in L and have length greater than or equal to p:

Pick $s =$

Suppose $s = xyz$ with $|xy| \leq p$ and $|y| > 0$.

Then when $i =$, $xy^i z =$

Example: $\Sigma = \{0, 1\}$, $L = \{ww^{\mathcal{R}} \mid w \in \{0, 1\}^*\}$.

Fix p an arbitrary positive integer. List strings that are in L and have length greater than or equal to p:

Pick $s =$

Suppose $s = xyz$ with $|xy| \leq p$ and $|y| > 0$.

Then when $i =$, $xy^iz =$

Example: $\Sigma = \{0, 1\}$, $L = \{0^j1^k \mid j \geq k \geq 0\}$.

Fix p an arbitrary positive integer. List strings that are in L and have length greater than or equal to p:

Pick $s =$

Suppose $s = xyz$ with $|xy| \leq p$ and $|y| > 0$.

Then when $i =$, $xy^iz =$

Example: $\Sigma = \{0, 1\}$, $L = \{0^n1^m0^n \mid m, n \geq 0\}$.

Fix p an arbitrary positive integer. List strings that are in L and have length greater than or equal to p:

Pick $s =$

Suppose $s = xyz$ with $|xy| \leq p$ and $|y| > 0$.

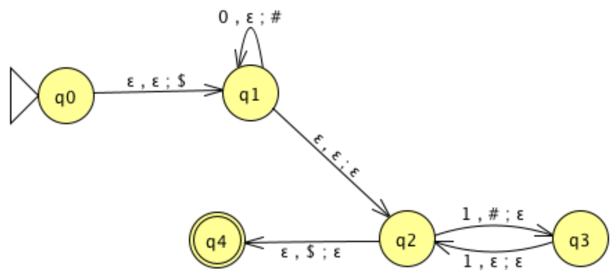
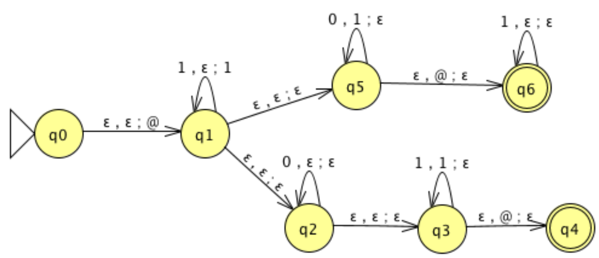
Then when $i =$, $xy^iz =$

Extra practice:

Language	$s \in L$	$s \notin L$	Is the language regular or nonregular?
$\{a^n b^n \mid 0 \leq n \leq 5\}$			
$\{b^n a^n \mid n \geq 2\}$			
$\{a^m b^n \mid 0 \leq m \leq n\}$			
$\{a^m b^n \mid m \geq n+3, n \geq 0\}$			
$\{b^m a^n \mid m \geq 1, n \geq 3\}$			
$\{w \in \{a,b\}^* \mid w = w^{\mathcal{R}}\}$			
$\{ww^{\mathcal{R}} \mid w \in \{a,b\}^*\}$			

Week4 friday

For the PDA state diagrams below, $\Sigma = \{0, 1\}$.

<table>
<tr><th>Mathematical description of language</th><th>State diagram of PDA recognizing language</th></tr>
<tr><td></td><td>$\Gamma = \{\$, \#\}$

</td></tr>
<tr><td></td><td>$\Gamma = \{@, 1\}$

</td></tr>
<tr><td>$\{0^i 1^j 0^k \mid i, j, k \geq 0\}$</td><td></td></tr>
</table>

Big picture: PDAs were motivated by wanting to add some memory of unbounded size to NFA. How do we accomplish a similar enhancement of regular expressions to get a syntactic model that is more expressive?

DFA, NFA, PDA: Machines process one input string at a time; the computation of a machine on its input string reads the input from left to right.

Regular expressions: Syntactic descriptions of all strings that match a particular pattern; the language described by a regular expression is built up recursively according to the expression's syntax

Context-free grammars: Rules to produce one string at a time, adding characters from the middle, beginning, or end of the final string as the derivation proceeds.

Term	Typical symbol	Definition
Context-free grammar (CFG)	G	$G = (V, \Sigma, R, S)$
Variables	V	Finite set of symbols that represent phases in production pattern
Terminals	Σ	Alphabet of symbols of strings generated by CFG $V \cap \Sigma = \emptyset$
Rules	R	Each rule is $A \to u$ with $A \in V$ and $u \in (V \cup \Sigma)^*$
Start variable	S	Usually on LHS of first / topmost rule
Derivation		Sequence of substitutions in a CFG
	$S \implies \cdots \implies w$	Start with start variable, apply one rule to one occurrence of a variable at a time
Language generated by the CFG G	$L(G)$	$\{w \in \Sigma^* \mid \text{there is derivation in } G \text{ that ends in } w\} = \{w \in \Sigma^* \mid S \implies ^* w\}$
Context-free language		A language that is the language generated by some CFG
Sipser pages 102-103		

Examples of context-free grammars, derivations in those grammars, and the languages generated by those grammars

$G_1 = (\{S\}, \{0\}, R, S)$ with rules

$$S \to 0S$$
$$S \to 0$$

In $L(G_1)$ ...

Not in $L(G_1)$ ...

$G_2 = (\{S\}, \{0,1\}, R, S)$

$$S \to 0S \mid 1S \mid \varepsilon$$

In $L(G_2)$ ...

Not in $L(G_2)$ ...

$(\{S,T\}, \{0,1\}, R, S)$ with rules

$$S \to T1T1T1T$$
$$T \to 0T \mid 1T \mid \varepsilon$$

In $L(G_3)$ ...

Not in $L(G_3)$ ...

$G_4 = (\{A, B\}, \{0, 1\}, R, A)$ with rules

$$A \to 0A0 \mid 0A1 \mid 1A0 \mid 1A1 \mid 1$$

In $L(G_4)$ ...

Not in $L(G_4)$ ...

Extra practice: Is there a CFG G with $L(G) = \emptyset$?

Week6 monday

For Turing machine $M = (Q, \Sigma, \Gamma, \delta, q_0, q_{accept}, q_{reject})$ where δ is the **transition function**

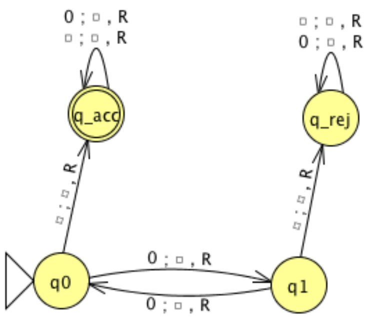
$$\delta : Q \times \Gamma \to Q \times \Gamma \times \{L, R\}$$

the **computation** of M on a string w over Σ is:

- Read/write head starts at leftmost position on tape.
- Input string is written on $|w|$-many leftmost cells of tape, rest of the tape cells have the blank symbol. **Tape alphabet** is Γ with $\textvisiblespace \in \Gamma$ and $\Sigma \subseteq \Gamma$. The blank symbol $\textvisiblespace \notin \Sigma$.
- Given current state of machine and current symbol being read at the tape head, the machine transitions to next state, writes a symbol to the current position of the tape head (overwriting existing symbol), and moves the tape head L or R (if possible).
- Computation ends **if and when** machine enters either the accept or the reject state. This is called **halting**. Note: $q_{accept} \neq q_{reject}$.

The **language recognized by the Turing machine** M, is $L(M) = \{w \in \Sigma^* \mid w \text{ is accepted by } M\}$, which is defined as

$$\{w \in \Sigma^* \mid \text{computation of } M \text{ on } w \text{ halts after entering the accept state}\}$$



Formal definition:

Sample computation:

$q0 \downarrow$						
0	0	0	␣	␣	␣	␣

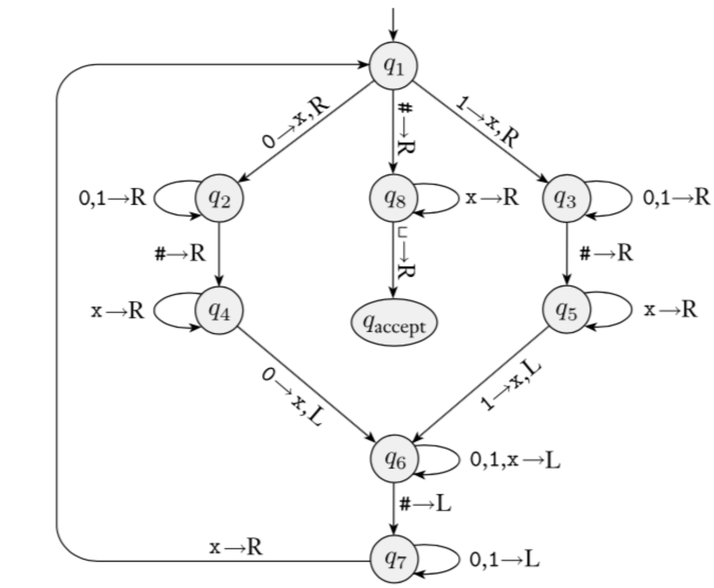
The language recognized by this machine is ...

To define a Turing machine, we could give a

- **Formal definition**, namely the 7-tuple of parameters including set of states, input alphabet, tape alphabet, transition function, start state, accept state, and reject state; or,
- **Implementation-level definition**: English prose that describes the Turing machine head movements relative to contents of tape, and conditions for accepting / rejecting based on those contents.

Conventions for drawing state diagrams of Turing machines: (1) omit the reject state from the diagram (unless it's the start state), (2) any missing transitions in the state diagram have value $(q_{reject}, \textvisiblespace, R)$.

Sipser Figure 3.10



Computation on input string 01#01

$q_1 \downarrow$						
0	1	#	0	1	␣	␣

Implementation level description of this machine:

Zig-zag across tape to corresponding positions on either side of # to check whether the characters in these positions agree. If they do not, or if there is no #, reject. If they do, cross them off.

Once all symbols to the left of the # are crossed off, check for any un-crossed-off symbols to the right of #; if there are any, reject; if there aren't, accept.

The language recognized by this machine is

$$\{w\#w \mid w \in \{0,1\}^*\}$$

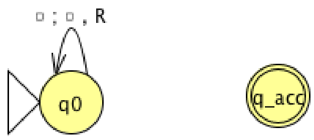
Extra practice

Computation on input string 01#1

$q_1 \downarrow$						
0	1	#	1	␣	␣	␣

Week6 wednesday

Fix $\Sigma = \{0,1\}$, $\Gamma = \{0,1,\text{\textvisiblespace}\}$ for the Turing machines with the following state diagrams:

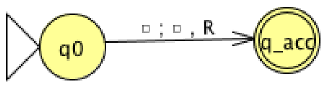


Implementation level description:

Example of string accepted:
Example of string rejected:

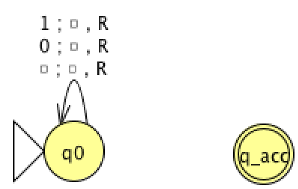
Implementation level description:

Example of string accepted:
Example of string rejected:



Implementation level description:

Example of string accepted:
Example of string rejected:



Implementation level description:

Example of string accepted:
Example of string rejected:

Two models of computation are called **equally expressive** when every language recognizable with the first model is recognizable with the second, and vice versa.

True / False: NFAs and PDAs are equally expressive.

True / False: Regular expressions and CFGs are equally expressive.

To say a language is **Turing-recognizable** means that there is some Turing machine that recognizes it.

Some examples of models that are **equally expressive** *with deterministic Turing machines:*

$\boxed{\textbf{May-stay}\text{ machines}}$ The May-stay machine model is the same as the usual Turing machine model, except that on each transition, the tape head may move L, move R, or Stay.

Formally: $(Q, \Sigma, \Gamma, \delta, q_0, q_{accept}, q_{reject})$ where

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

Claim: Turing machines and May-stay machines are equally expressive. *To prove ...*

To translate a standard TM to a may-stay machine:

To translate one of the may-stay machines to standard TM: any time TM would Stay, move right then left.

Formally: suppose $M_S = (Q, \Sigma, \Gamma, \delta, q_0, q_{acc}, q_{rej})$ has $\delta : Q \times \Gamma \to Q \times \Gamma \times \{L, R, S\}$. Define the Turing-machine

$$M_{new} = (\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad)$$

$\boxed{\textbf{Multitape Turing machine}}$ A multitape Turing macihne with k tapes can be formally representated as $(Q, \Sigma, \Gamma, \delta, q_0, q_{acc}, q_{rej})$ where Q is the finite set of states, Σ is the input alphabet with $\sqcup \notin \Sigma$, Γ is the tape alphabet with $\Sigma \subsetneq \Gamma$, $\delta : Q \times \Gamma^k \to Q \times \Gamma^k \times \{L, R\}^k$ (where k is the number of states)

If M is a standard TM, it is a 1-tape machine.

To translate a k-tape machine to a standard TM: Use a new symbol to separate the contents of each tape and keep track of location of head with special version of each tape symbol. Sipser Theorem 3.13

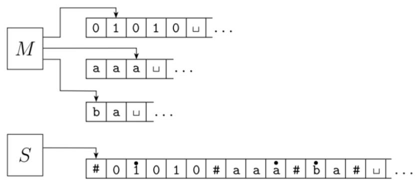


FIGURE 3.14
Representing three tapes with one

Extra practice: $\boxed{\textbf{Wikipedia Turing machine}}$ Define a machine $(Q, \Gamma, b, \Sigma, q_0, F, \delta)$ where Q is the finite set of states Γ is the tape alphabet, $b \in \Gamma$ is the blank symbol, $\Sigma \subsetneq \Gamma$ is the input alphabet, $q_0 \in Q$ is the start state, $F \subseteq Q$ is the set of accept states, $\delta : (Q \setminus F) \times \Gamma \not\to Q \times \Gamma \times \{L, R\}$ is a partial transition function If computation enters a state in F, it accepts If computation enters a configuration where δ is not defined, it rejects . Hopcroft and Ullman, cited by Wikipedia

Enumerators Enumerators give a different model of computation where a language is **produced, one string at a time**, rather than recognized by accepting (or not) individual strings.

Each enumerator machine has finite state control, unlimited work tape, and a printer. The computation proceeds according to transition function; at any point machine may "send" a string to the printer.

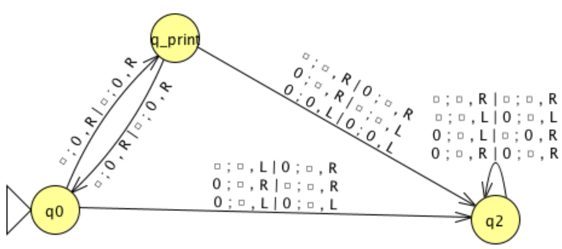
$$E = (Q, \Sigma, \Gamma, \delta, q_0, q_{print})$$

Q is the finite set of states, Σ is the output alphabet, Γ is the tape alphabet ($\Sigma \subsetneq \Gamma, \llcorner\!\lrcorner \in \Gamma \setminus \Sigma$),

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

where in state q, when the working tape is scanning character x and the printer tape is scanning character y, $\delta((q, x, y)) = (q', x', y', d_w, d_p)$ means transition to control state q', write x' on the working tape, write y' on the printer tape, move in direction d_w on the working tape, and move in direction d_p on the printer tape. The computation starts in q_0 and each time the computation enters q_{print} the string from the leftmost edge of the printer tape to the first blank cell is considered to be printed.

The language **enumerated** by E, $L(E)$, is $\{w \in \Sigma^* \mid E$ eventually, at finite time, prints $w\}$.



$q0$						
␣ *	␣	␣	␣	␣	␣	␣
␣ *	␣	␣	␣	␣	␣	␣

Theorem 3.21 A language is Turing-recognizable iff some enumerator enumerates it.

Proof:

Assume L is enumerated by some enumerator, E, so $L = L(E)$. We'll use E in a subroutine within a high-level description of a new Turing machine that we will build to recognize L.

Goal: build Turing machine M_E with $L(M_E) = L(E)$.

Define M_E as follows: $M_E =$ "On input w,

1. Run E. For each string x printed by E.
2. Check if $x = w$. If so, accept (and halt); otherwise, continue."

Assume L is Turing-recognizable and there is a Turing machine M with $L = L(M)$. We'll use M in a subroutine within a high-level description of an enumerator that we will build to enumerate L.

Goal: build enumerator E_M with $L(E_M) = L(M)$.

Idea: check each string in turn to see if it is in L.

How? Run computation of M on each string. *But*: need to be careful about computations that don't halt.

Recall String order for $\Sigma = \{0, 1\}$: $s_1 = \varepsilon$, $s_2 = 0$, $s_3 = 1$, $s_4 = 00$, $s_5 = 01$, $s_6 = 10$, $s_7 = 11$, $s_8 = 000$, ...

Define E_M as follows: $E_M =$ " *ignore any input.* Repeat the following for $i = 1, 2, 3, \ldots$

1. Run the computations of M on s_1, s_2, ..., s_i for (at most) i steps each
2. For each of these i computations that accept during the (at most) i steps, print out the accepted string."

Week6 friday

Nondeterministic Turing machine

At any point in the computation, the nondeterministic machine may proceed according to several possibilities: $(Q, \Sigma, \Gamma, \delta, q_0, q_{acc}, q_{rej})$ where

$$\delta : Q \times \Gamma \to \mathcal{P}(Q \times \Gamma \times \{L, R\})$$

The computation of a nondeterministic Turing machine is a tree with branching when the next step of the computation has multiple possibilities. A nondeterministic Turing machine accepts a string exactly when some branch of the computation tree enters the accept state.

Given a nondeterministic machine, we can use a 3-tape Turing machine to simulate it by doing a breadth-first search of computation tree: one tape is "read-only" input tape, one tape simulates the tape of the nondeterministic computation, and one tape tracks nondeterministic branching. Sipser page 178

Two models of computation are called **equally expressive** when every language recognizable with the first model is recognizable with the second, and vice versa.

Church-Turing Thesis (Sipser p. 183): The informal notion of algorithm is formalized completely and correctly by the formal definition of a Turing machine. In other words: all reasonably expressive models of computation are equally expressive with the standard Turing machine.

A language L is **recognized by** a Turing machine M means

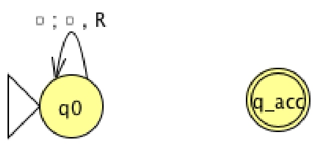
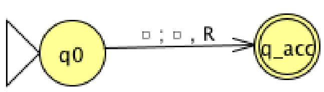
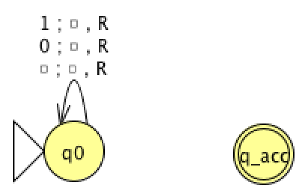
A Turing machine M **recognizes** a language L if means

A Turing machine M is a **decider** means

A language L is **decided by** a Turing machine M means

A Turing machine M **decides** a language L means

Fix $\Sigma = \{0, 1\}$, $\Gamma = \{0, 1, \textvisiblespace\}$ for the Turing machines with the following state diagrams:

 Decider? Yes / No	 Decider? Yes / No
 Decider? Yes / No	 Decider? Yes / No

Claim: If two languages (over a fixed alphabet Σ) are Turing-recognizable, then their union is as well.

Proof using Turing machines:

Proof using nondeterministic Turing machines:

Proof using enumerators:

Describing Turing machines (Sipser p. 185)

To define a Turing machine, we could give a

- **Formal definition**: the 7-tuple of parameters including set of states, input alphabet, tape alphabet, transition function, start state, accept state, and reject state; or,
- **Implementation-level definition**: English prose that describes the Turing machine head movements relative to contents of tape, and conditions for accepting / rejecting based on those contents.
- **High-level description**: description of algorithm (precise sequence of instructions), without implementation details of machine. As part of this description, can "call" and run another TM as a subroutine.

The Church-Turing thesis posits that each algorithm can be implemented by some Turing machine

High-level descriptions of Turing machine algorithms are written as indented text within quotation marks.

Stages of the algorithm are typically numbered consecutively.

The first line specifies the input to the machine, which must be a string. This string may be the encoding of some object or list of objects.

Notation: $\langle O \rangle$ is the string that encodes the object O. $\langle O_1, \ldots, O_n \rangle$ is the string that encodes the list of objects $O_1, \ldots, O_n$.

Assumption: There are Turing machines that can be called as subroutines to decode the string representations of common objects and interact with these objects as intended (data structures).

For example, since there are algorithms to answer each of the following questions, by Church-Turing thesis, there is a Turing machine that accepts exactly those strings for which the answer to the question is "yes"

- Does a string over $\{0, 1\}$ have even length?
- Does a string over $\{0, 1\}$ encode a string of ASCII characters?[1]
- Does a DFA have a specific number of states?
- Do two NFAs have any state names in common?
- Do two CFGs have the same start variable?

[1] An introduction to ASCII is available on the w3 tutorial here.

Week3 friday

Theorem: For an alphabet Σ, For each language L over Σ,

$$\begin{gathered} L \text{ is recognized by some DFA} \\ \text{iff} \\ L \text{ is recognized by some NFA} \\ \text{iff} \\ L \text{ is described by some regular expression} \end{gathered}$$

If (any, hence all) these conditions apply, L is called **regular**.

Prove or Disprove: There is some alphabet Σ for which there is some language recognized by an NFA but not by any DFA.

Prove or Disprove: There is some alphabet Σ for which there is some finite language not described by any regular expression over Σ.

Prove or Disprove: If a language is recognized by an NFA then the complement of this language is not recognized by any DFA.

Set	Cardinality
$\{0, 1\}$	
$\{0, 1\}^*$	
$\mathcal{P}(\{0, 1\})$	
The set of all languages over $\{0, 1\}$	
The set of all regular expressions over $\{0, 1\}$	
The set of all regular languages over $\{0, 1\}$	

Pumping Lemma (Sipser Theorem 1.70): If A is a regular language, then there is a number p (a *pumping length*) where, if s is any string in A of length at least p, then s may be divided into three pieces, $s = xyz$ such that

- $|y| > 0$
- for each $i \geq 0$, $xy^iz \in A$
- $|xy| \leq p$.

True or False: A pumping length for $A = \{0,1\}^*$ is $p = 5$.

True or False: A pumping length for $A = \{1, 01, 001, 0001, 00001\}$ is $p = 4$.

True or False: A pumping length for $A = \{0^j 1 \mid j \geq 0\}$ is $p = 3$.

True or False: For any language A, if p is a pumping length for A and $p' > p$, then p' is also a pumping length for A.