CSE 105 Review

Kevin Jacob

Winter 2025

Table Of Contents

1 Regular Expressions

1.1 Definition

Basis steps of recursive definition

- a is a regular expression, for $a \in \Sigma$
- ϵ is a regular expression
- $\emptyset$ is a regular expression

Recursive steps of recursive definition

- $(R_1 \cup R_2)$ is a regular expression when R_1, R_2 are regular expressions
  $L((R_1 \cup R_2)) = L(R_1) \cup L(R_2) = \{w | w \in L(R_1) \vee w \in L(R_2)\}$
- $(R_1 \circ R_2)$ is a regular expression when R_1, R_2 are regular expressions
  $L((R_1 \circ R_2)) = L(R_1) \circ L(R_2) = \{uv | u \in L(R_1) \wedge v \in L(R_2)\}$
- R_1^* is a regular expression when R_1 is a regular expression
  $L((R_2)^*) = (L(R_1))^* = \{w_1, \ldots, w_k | k \geq 0 \text{ and each } w_i \in L(R_1)\}$

1.2 Conventions

Assuming Σ is the alphabet, we use the following conventions

- Σ
  regular expression describing language consisting of all strings of length 1 over Σ
- $*$ then $\circ$ then $\cup$
  precedence order, unless parentheses are used to change it
- $R_1 R_2$
  shorthand for $R_1 \circ R_2$ (concatenation symbol is implicit)
- R^+
  shorthand for $R^* \circ R$
- R^k
  shorthand for R concatenated with itself k times, where k is a (specific) natural number

2 Finite Automata

2.1 Definition

The **formal definition** of a finite automata consists of a 5-tuple $M = (Q, \Sigma, \delta, q_0, F)$ and a finite automata can also be represented by a state diagram

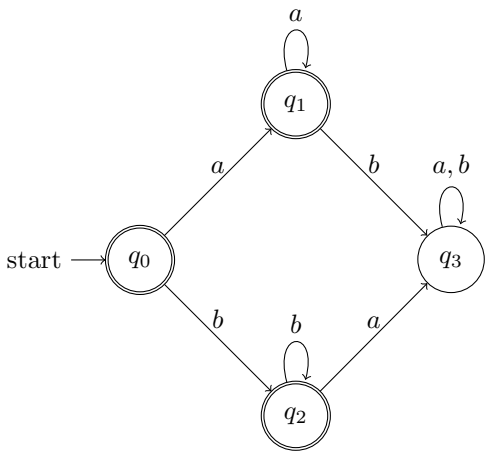
- **Finite set of states** Q can be labeled by any collection of distinct names
- The **alphabet** Σ determines the possible inputs to the automaton. Each input to the automaton is a string over Σ
- The **transition function** δ gives the next state of the automaton based on the current state of the machine and on the next input symbol
- The **start state** q_0 is an element of Q. each computation of the machine starts at the start state.
- The **accept (final) states** F form a subset of the states of the automaton, $F \subseteq Q$. These states flag if a machine accepts or rejects an input string.

The computation of a machine on an input string is a sequence of states in the machine, starting with the start state, determined by transitions of the machine as it reads successive input symbols.

The finite automaton M accepts the given input string exactly when the computation of M on the input string ends in an accept state.

The language of M, $L(M)$, is defined as the set of all strings that are each accepted by the machine M. Each string that is rejected by M is not in $L(M)$.

2.2 Example



Formal Definition: $(\{q_0, q_1, q_2, q_3\}, \{a, b\}, \delta, q_0, \{q_0, q_1, Q_2\})$
where $\delta : Q \times \Sigma \to Q$

Q/Σ	a	b
q_0	q_1	q_2
q_1	q_1	q_3
q_2	q_3	q_2
q_3	q_3	q_3

Language recognized by automaton: $L(a^* \cup b^*)$

3 Nondeterministic Automata

3.1 Definition

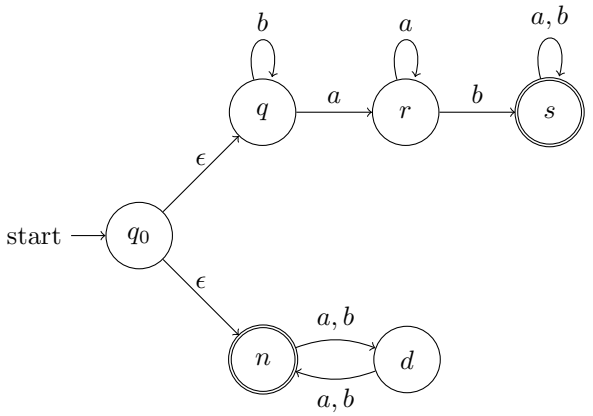
The computation of a **determinstic** finite automaton has exactly one choice for its next step given the current state and character read, but an NFA allows us to have multiple

The **formal definition** of a nondeterministic finite automaton also consists of a 5-tuple $M = (Q, \Sigma, \delta, q_0, F)$ that can also be represented by a state diagram

- **Finite set of states** Q can be labelled by any collection of distinct names
- The **alphabet** Σ where each input to the automaton is a string over Σ
- The **transition function** $\delta : Q \times \Sigma_\epsilon \to \mathcal{P}(Q)$ gives the **set of possible next states** for a transition from the current state upon reading a symbol or spontaneously moving
- The **start state** q_0 is an element of Q. each computation of the machine starts at the start state.
- The **accept (final) states** F form a subset of the states of the automaton, $F \subseteq Q$. These states flag if a machine accepts or rejects an input string.

M accepts the input string $w \in \Sigma^*$ if and only if **there** is a computation of M on w that processes the **whole string** and ends in an accept state

3.2 Example



Formal Definition: $(\{q_0, q, r, s, n, d\}, \{a, b\}, \delta, q_0, \{s, n\})$

where $\delta : Q \times \Sigma_\epsilon \to \mathcal{P}(Q)$

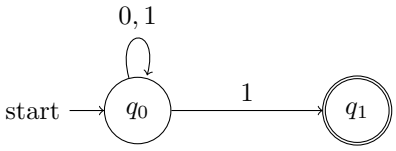
states/labels	a	b	ϵ
q_0	$\emptyset$	$\emptyset$	$\{q, n\}$
q	$\{r\}$	$\{q\}$	$\emptyset$
r	$\{r\}$	$\{s\}$	$\emptyset$
s	$\{s\}$	$\{s\}$	$\emptyset$
n	$\{d\}$	$\{d\}$	$\emptyset$
d	$\{n\}$	$\{n\}$	$\emptyset$

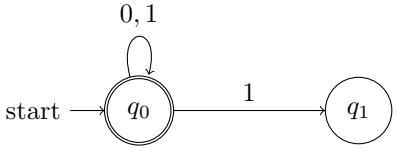
Language recongized by NFA: $\{w \in \{a, b\}^* | w$ has even length or has ab as a substring$\}$

4 Automata Constructions

4.1 Complementation

- The collection of languages that are each recognizable by a DFA is **closed** under complementation. We can flip the accept status of states in a DFA.
- The complementation construction doesn't work with an NFA





The string 1 *is accepted by both NFAs*

4.2 Union

- The collection of languages that are each recognizable by a NFA is **closed** under union. We can add a start state and add transitions of the empty string to the start of different NFAs.
- We can't do the same construction since spontaneous moves are not possible in a DFA

4.3 DFA Constructions for Union and Intersection

4.3.1 Union

Suppose A_1, A_2 are languages over the alphabet Σ. If there is a DFA M_1 such that $L(M_1) = A_1$ and a DFA M_2 such that $L(M_2) = A_2$, then there is a DFA M such that $L(M) = A_1 \cup A_2$.
Formal Construction
$M_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$ and $M_2 = (Q_2, \Sigma, \delta_2, q_2, F_2)$ where $L(M_1) = A_1$ and $L(M_2) = A_2$. We want to build M with $L(M) = A_1 \cup A_2$.
$M = (Q, \Sigma, \delta, q_0, F)$
$Q = \{(q, q') | q \in Q_1 \text{ and } q' \in Q_2\} = Q_1 \times Q_2$
$\delta : Q \times \Sigma \to Q \qquad \delta(((q, q'), x)) = (\delta_1(q, x), \delta_2(q', x))$
$q_0 = (q_1, q_2)$
$F = (F_1 \times Q_2) \cup (Q_1 \times F_2)$

4.3.2 Intersection

Suppose A_1, A_2 are languages over an alphabet Σ. If there is a DFA M_1 such that $L(M_1) = A_1$ and a DFA M_2 such that $L(M_2) = A_2$, then there is a DFA M such that $L(M) = A_1 \cap A_2$
Formal Construction:
$M_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$ and $M_2 = (Q_2, \Sigma, \delta_2, q_2, F_2)$ where $L(M_1) = A_1$ and $L(M_2) = A_2$. We want to build M with $L(M) = A_1 \cap A_2$.
$M = (Q, \Sigma, \delta, q_0, F)$
$Q = \{(q, q') | q \in Q_1 \text{ and } q' \in Q_2\} = Q_1 \times Q_2$
$\delta : Q \times \Sigma \to Q \qquad \delta(((q, q'), x)) = (\delta_1(q, x), \delta_2(q', x))$
$q_0 = (q_1, q_2)$
$F = F_1 \times F_2$

5 Regular Languages

If a language is regular

- there is a regular expression that describes it
- there is a DFA that recognizes it
- there is a NFA that recognizes it

5.1 Concactenation

Suppose A_1, A_2 are languages over an alphabet Σ, If there is a NFA N_1 such that $L(N_1) = A_1$ and NFA N_2 such that $L(N_2) = A_2$, then there is another NFA N such that $L(N) = A_1 \circ A_2$
Formal Construction
$N_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$ and $N_2 = (Q_2, \Sigma, \delta_2, q_2, F_2)$ where $L(N_1) = A_1$ and $L(N_2) = A_2$. We want to build N with $L(N) = A_1 \circ A_2$.
$N = (Q, \Sigma, \delta, q_0, F)$
$Q = Q_1 \cup Q_2$
$q_0 = q_1$
$F = F_2$
$\delta : Q \times \Sigma_\epsilon \to \mathcal{P}(Q)$

$$\delta((q,a)) = \begin{cases} \delta_1((q,a)) & q \in Q_1 \text{ and } q \notin F_1 \\ \delta_1((q,a)) & q \in F_1 \text{ and } a \in \Sigma \\ \delta_1((q,a)) \cup \{q_2\} & q \in F_1 \text{ and } a = \epsilon \\ \delta_2((q,a)) & q \in Q_2 \end{cases}$$

5.2 Kleene Star

Suppose A is a language over alphabet Σ. If there is a NFA N such that $L(N) = A$, then there is another NFA N' such that $L(N') = A^*$.

Formal Construction

$N = (Q, \Sigma, \delta, q_1, F)$ where $q_0 \notin Q$. We want to build N' with $L(N') = A^*$.

$N' = (Q', \Sigma, \delta', q_0, F')$

$Q' = Q \cup \{q_0\}$

$F' = F \cup \{q_0\}$

$\delta' : Q' \times \Sigma_\epsilon \to \mathcal{P}(Q')$

$$\delta'((q,a)) = \begin{cases} \delta((q,a)) & q \in Q \text{ and } q \notin F \\ \delta((q,a)) & q \in F \text{ and } a \in \Sigma \\ \delta((q,a)) \cup \{q_1\} & q \in F \text{ and } a = \epsilon \\ \{q_1\} & q = q_0 \text{ and } a = \epsilon \\ \emptyset & q = q_0 \text{ and } a \in \Sigma \end{cases}$$

5.3 NFA to DFA

Suppose A is a language over alphabet Σ. If there is a NFA N such that $L(N) = A$ then there is a DFA M such that $L(M) = A$

Formal Construction

We can treat states in M as macro-states which are collections of states from N that represent the set of possible states a computation of N might be in.

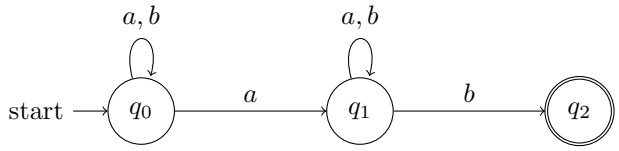
Let $N = (Q, \Sigma, \delta, q_0, F)$. We want to build M with $L(M) = A$.

$M = (\mathcal{P}(Q), \Sigma, \delta', q', \{X \subseteq Q | X \cap F \neq \emptyset\})$

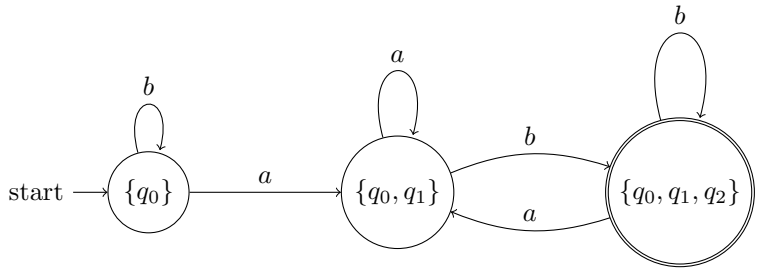
$q' = \{q \in Q | q = q_0$ or is accessible from q_0 by spontaneous moves in $N\}$

$\delta'((X, x)) = \{q \in \delta((r, x))$ for some $r \in X$ or is accessible from such an r by spontaneous moves in $N\}$

NFA



DFA



5.4 Regular Expression to NFA/DFA

5.4.1 Regular Expression to NFA

Suppose A is a language over an alphabet Σ. If there is a regular expression R where $L(R) = A$, then there is a NFA N such that $L(N) = A$.

We can use the NFA constructions for the concatenation and kleene star as shown in the previous sections.

5.4.2 DFA to Regular Expression

Suppose A is a language over an alphabet Σ. If there is a DFA M such that $L(M) = A$, then there is a regular expression R such that $L(R) = A$.

1. Add a new start state with ϵ arrow to old start state
2. Add new accept state with ϵ arrow from old accept states. Make old accept states non-accept.
3. Remove one (of the old) states at a time: modify regular expressions on arrows that went through the removed state to restore language recognized by machine

6 Pumping Lemma

6.1 Countability

Set	Cardinality
$\{0,1\}$	2
$\{0,1\}^*$	Countably Infinite
$\mathcal{P}(\{0,1\})$	4
All languages over $\{0,1\}$ $(\mathcal{P}(\{0,1\}))$	Uncountable
Set of all regular expressions over $\{0,1\}$	Countably Infinite
Set of all regular labguages over $\{0,1\}$	Countably Infinite

Note: The power set of a countably infinite set is uncountable

6.2 Pumping Lemma Definition

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 $i \geq 0, xy^iz \in A$
- $|xy| \leq p$

Example: A pumping length for $A = \{0,1\}^*$ is $p = 5$
$S \in A$ with $|S| \geq 5$
$x = \epsilon$ $y = S_1$ (the first character in S) z is the rest of S
$xy^iz \in \{0,1\}^*$ for all $i \geq 0$

7 Proving Nonregularity

The pumping lemma can't be used to prove that a language is regular, but it can be used to prove that a language is not regular
Proof Strategy to show 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 lengthand therefore is not regular

7.1 Examples

1. $\Sigma = \{0,1\}, L = \{0^n1^n | n \geq 0\}$
   Pick $s = 0^p1^p$
   Suppose $s = xyz$ with $|xy| \leq p$ and $|y| > 0$ where $x = 0^k, y = 0^r, z = 0^{p-k-r}1^p$
   When $i = 0, xy^iz = 0^k0^{p-k-r}1^p$ which is not apart of the language and therefore L doesn't have a pumping length.
2. $\Sigma = \{0,1\}, L = \{0^j1^k | j \geq k \geq 0\}$
   Suppose $s = xyz$ with $|xy| \leq p$ and $|y| > 0$ where $x = 0^k, y = 0^r, z = 0^{p-k-r}1^p$
   When $i = 0, xy^iz = 0^k0^{p-k-r}1^p$ which is not apart of the language since less 0s than 1s and therefore L doesn't have a pumping length.

8 Pushdown Automata

A pushdown automata is like an NFA with access to a stack. At each step transition to new state, read the top of the stack, and possibly push or pop a letter from the stack.

8.1 Definition

A PDA is specified by a 6-tuple $(Q, \Sigma, \Gamma, \delta, q_0, F)$

- Q is the finite set of states
- Σ is the input alphabet
- Γ is the stack alphabet
- $\delta : Q \times \Sigma_\epsilon \times \Gamma_\epsilon \to \mathcal{P}(Q \times \Gamma_\epsilon)$ is the transition function
- $q_0 \in Q$ is the start state
- $F \subseteq Q$ is the set of accept states

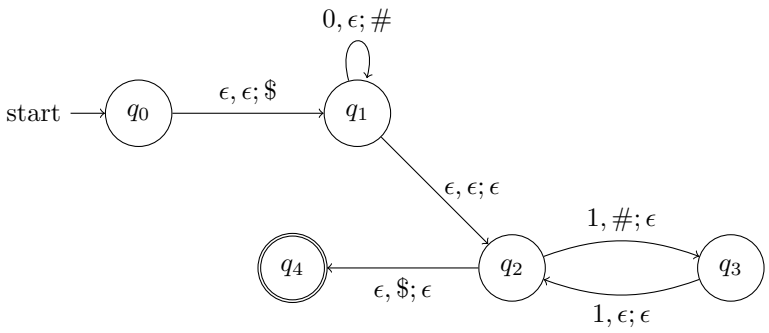
Transition Label For PDA
input character to read or ϵ, top character of the stack to pop or ϵ, what character to push on top of the stack or ϵ

8.2 Examples

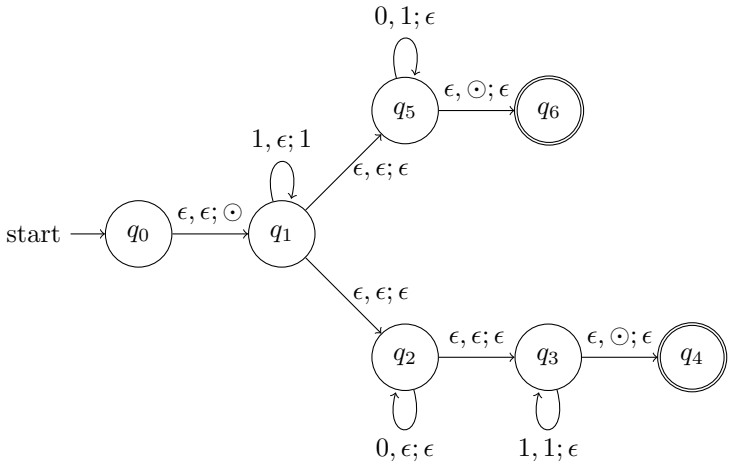
1. Language: $\{0^n 1^{2n} | n \geq 0\}$
   PDA: $(\{q_0, q_1, q_2, q_3, q_4\}, \{0, 1\}, \{\$, \#\}, \delta, q_0, \{q_4\})$

δ

input	output
$(q_0, 0, \$)$	$\emptyset$
$\vdots$	$\vdots$
$(q_0, \epsilon, \epsilon)$	$\{(q_1, \$)\}$
$\vdots$	$\vdots$



2. Language: $\{1^n0^n1^m|n,m \geq 0\} \cup \{1^n0^m1^n|n,m \geq 0\}$



For each language L over Σ, there is an NFA N with $L(N) = L$ then there is a PDA M with $L(M) = L$

9 Context-free Grammars and Languages

9.1 Context-free Grammar

$G = (V, \Sigma, R, S)$

- V is a finite set of symbols that represent phases in the production pattern
- Σ is the alphabet of symbols of strings generated by CFG
- R is the set of rules
- S is the start variable

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

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

$L(G_1) = 0^+$

9.2 Context-free Language

A language that is generated by some context-free grammar is a context-free language.

A language is generated by some context-free grammar if and only if it is recognizable by some push-down automaton.

Every Regular Language is Context Free

10 Language Closure

True/False	Closure Claim
True	The class of regular languages over Σ is closed under complementation
True	The class of regular languages over Σ is closed under union
True	The class of regular languages over Σ is closed under intersection
True	The class of regular languages over Σ is closed under concatenation
True	The class of regular languages over Σ is closed under Kleene star
True	The class of non-regular languages over Σ is closed under complementation
False	The class of non-regular languages over Σ is closed under union
False	The class of non-regular languages over Σ is closed under intersection
Flase	The class of non-regular languages over Σ is closed under concatenation
False	The class of non-regular languages over Σ is closed under Kleene star
False	The class of context-free languages over Σ is closed under complementation
True	The class of context-free languages over Σ is closed under union
False	The class of context-free languages over Σ is closed under intersection
True	The class of context-free languages over Σ is closed un concatenation
True	The class of context-free languages over Σ is closed under Kleene star

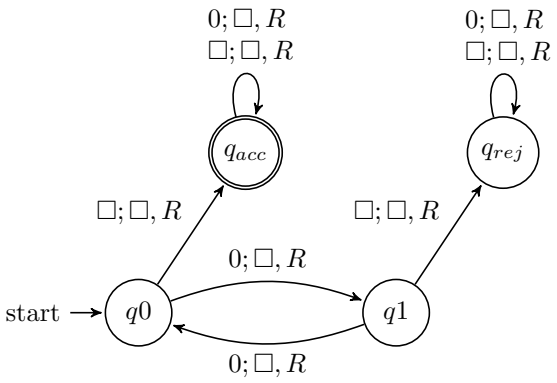
11 Turing Machines

11.1 Definition

The **formal definition** of a Turing Machine is $M = (Q, \Sigma, \Gamma, \delta, q_0, q_{accept}, q_{reject})$

- **Set of states** Q
- The **tape alphabet** Γ with $\textvisiblespace \in \Gamma$ and $\Sigma \subseteq \Gamma$. The blank symbol $\textvisiblespace \notin \Sigma$ and the input string is written on the $|w|$-many leftmost cells of tape where the rest of cells have the blank symbol.
- The **transition function** $\delta : Q \times \Gamma \to Q \times \Gamma \times \{L, R\}$ gives the next state and moves the tape left or right based on the current state and symbol being read at the tape head
- The **start state** q_0 is an element of Q. Each computation starts at this state
- The computation **halts** when the machine enters an **accept** (q_{accept}) or **reject** (q_{reject}) state

11.2 Example



Formal definition:$(\{q_0q_1, q_{acc}, q_{rej}\}, \{0\}, \{\textvisiblespace, 0\}, q_0, q_{acc}, q_{rej})$
Sample computation: $w = 000$

$q0\downarrow$						
0	0	0	␣	␣	␣	␣
	$q1\downarrow$					
␣	0	0	␣	␣	␣	␣
		$q0\downarrow$				
␣	␣	0	␣	␣	␣	␣
			$q1\downarrow$			
␣	␣	␣	␣	␣	␣	␣
				$q_{rej}\downarrow$		
␣	␣	␣	␣	␣	␣	␣

The language recognized by this machine is $\{0^{2i} | i \geq 0\}$
The labels on the diagram consist of three values *tape symbol being scanned*;*tape symbol to write at location of read/write head*,*direction to move read/write head*
An example definition for one of the transtions using the transition function is $\delta((q_0, \textvisiblespace)) = (q_{acc}, \textvisiblespace, R)$

11.3 Describing Turing Machines

- **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.

12 Recognizable and Decidable Languages

12.1 Definition

- A language L is **recognized by** a Turing Machine M means $L = \{w | M \text{ accepts } w\}$. For each string not in the language, M either rejects or doesn't halt
- A language L is **decided by** a Turing Machine M means M is a decider (halts on each computation) and M recognizes L

12.2 Closure

- A **Turing-recognizable** language is a set of strings that is the language recognized by some Turing machine. We also say that such languages are recognizable.
- A **Turing-decidable** language is a set of strings that is the language recognized by some decider. We also say that such languages are decidable.
- An **unrecognizable** language is a language that is not Turing-recognizable.
- An **undecidable** language is a language that is not Turing-decidable.

True/False	Statement
True	Any decidable language is also recognizable.
False	Any recognizable language is also decidable.
False	Any undecidable language is also unrecognizable.
True	Any unrecognizable language is also undecidable.

13 General Constructions with Turing Machines

13.1 Co-Recognizable

Definition: A language L over an alphabet Σ is called **co-recognizable** if its complement, defined as $\Sigma^* \setminus L = \{x \in \Sigma^* \mid x \notin L\}$, is Turing-recognizable.

A language is Turing-decidable if and only if both it and its complement are Turing-recognizable.

Proof, first direction: Suppose a language L is Turing-decidable. WTS that both it and its complement are Turing-recognizable
By definition we have a TM M that decides L; namely for each string W, $W \in L$, M accepts W and if $W \notin L$, M rejects W

1. Build a TM that recognizes L
   We can use M as it is
2. Build a TM that recognized $\bar{L}$
   We can swap the accept and reject stated in M

We will need to use dovetailing to prove the other direction

13.2 Dovetailing

Dovetailing involves interleaving progress on multiple computations by limiting the number of steps each computation makes in each round
Proof, second direction:
By definition, we have a TM M_L with $L(M_L) = L$ and another TM M_C with $L(M_C) = \bar{L}$
Build M_{new} ="On input w

1. For $n = 1, 2, 3, \ldots$
2. Run M_L on w for at most n steps
3. Run M_c on w for at most n steps
4. If M_L accepts w within n steps, accept
5. If M_c accepts w within n steps, reject
6. increment n and continue loop"

It's sufficient to prove that each string in L is accepted by M_{new} and each string not in L is rejected by M_{new}. First let W be an arbitrary string in L. By assumption that M_L recognizes L, we know that M_L accepts W. Let d be the number of steps it takes M_L to halt and accept W. By assumption that M_C recognizes $\bar{L}$, we know that M_C does not accept W. We trace the computation of M_{new} on W: For all iterations of the loop with $n < d$, step 2 and 3 run for at most n steps and the conditions in step 4 and step 5 are not satisfied. At the loop iteration with $n = d$, the subroutine in step 2 ends with M_L accepting W. After the at most d steps of the computation simulated in step 3, in step 4, the condition of the conditional is true, so M_{new} accepts W.
Next let W be an arbitrary string not in L. By assumption that M_C recognizes $\bar{L}$, we know that M_C accpets W. Let d be the number of steps it took M_C to halt and accept W. By assumption that M_L recognizes L, we know that M_L does not accept W. Tracing the computation of M_{new} on W (like before) by definition of d, the computation doesn't halt for loop iterations with $n < d$; and at $n = d$ the subroutine in step 2 doesn't halt and accept but in step3 it does so the condition in step 4 isn't satisfied and the computation continues to step 5 where the condition is satisfied and M_new rejects W.

13.3 Union of Turing Decidable Languages

Claim: If two languages (over a fixed alphabet Σ) are Turing-decidable, then their union is as well
Example: Let L_1, L_2 be arbitrary decidable languages.
Let M_1, M_2 be deciders with $L(M_1) = L_1$ and $L(M_2) = L_2$ guaranteed to exist by definition of L_1 and L_2 being decidable
Define $M =$"On input w

1. Run M_1 on w
2. If M_1 accepts w, accept
3. otherwise run M_2 on w
4. If M_2 accepts w, accept
5. Otherwise, reject"

13.4 Intersection of Turing Recognizable Languages

Claim: If two languages (over a fixed alphabet Σ) are Turing-recognizable, then their union is as well. **Example**: Let L_1, L_2 be arbitrary recognizable languages. Let M_1, M_2 be TMs with $L(M_1) = L_1$ and $L(M_2) = L_2$ guaranteed to exist by definition of L_1, L_2 being recognizable. Goal is to build a TM for $L_1 \cup L_2$ (use dovetailing).
Define $M =$"On input w

1. For $n = 1, 2, \ldots$
2. For M_1 on w for (at most) n steps
3. If M_1 accepts w, accept
4. Otherwise, run M_2 on w for (at most) n steps
5. If M_2 accepts w, accpets
6. Otherwise, continue to next loop iteration

14 Computational Problems

A **computational problem** is decidable iff language encoding its positive problem instances is decidable

Acceptance problem		
...for DFA	A_{DFA}	$\{\langle B, w\rangle \mid B$ is a DFA that accepts input string $w\}$
...for NFA	A_{NFA}	$\{\langle B, w\rangle \mid B$ is a NFA that accepts input string $w\}$
...for regular expressions	A_{REX}	$\{\langle R, w\rangle \mid R$ is a regular expression that generates input string $w\}$
...for CFG	A_{CFG}	$\{\langle G, w\rangle \mid G$ is a context-free grammar that generates input string $w\}$
...for PDA	A_{PDA}	$\{\langle B, w\rangle \mid B$ is a PDA that accepts input string $w\}$
Language emptiness testing		
...for DFA	E_{DFA}	$\{\langle A\rangle \mid A$ is a DFA and $L(A) = \emptyset\}$
...for NFA	E_{NFA}	$\{\langle A\rangle \mid A$ is a NFA and $L(A) = \emptyset\}$
...for regular expressions	E_{REX}	$\{\langle R\rangle \mid R$ is a regular expression and $L(R) = \emptyset\}$
...for CFG	E_{CFG}	$\{\langle G\rangle \mid G$ is a context-free grammar and $L(G) = \emptyset\}$
...for PDA	E_{PDA}	$\{\langle A\rangle \mid A$ is a PDA and $L(A) = \emptyset\}$
Language equality testing		
...for DFA	EQ_{DFA}	$\{\langle A, B\rangle \mid A$ and B are DFAs and $L(A) = L(B)\}$
...for NFA	EQ_{NFA}	$\{\langle A, B\rangle \mid A$ and B are NFAs and $L(A) = L(B)\}$
...for regular expressions	EQ_{REX}	$\{\langle R, R'\rangle \mid R$ and R' are regular expressions and $L(R) = L(R')\}$
...for CFG	EQ_{CFG}	$\{\langle G, G'\rangle \mid G$ and G' are CFGs and $L(G) = L(G')\}$
...for PDA	EQ_{PDA}	$\{\langle A, B\rangle \mid A$ and B are PDAs and $L(A) = L(B)\}$

14.1 A_{TM} Examples

14.1.1 A_{TM} is Turing-recognizable

Define R_{ATM} ="On input x

1. Type check whether $x = \langle M, w\rangle$ where M is a TM and w string. If not, reject
2. Run M on w
3. If M accepts w, accept
4. If M rejects w, reject

WTS $L(R_{ATM}) = A_{TM}$
Take an arbitrary x

1. if $x \neq \langle M, w\rangle$ for any TM M string w. By definition of A_{TM}, $x \notin A_{TM}$ and tracing R_{ATM} on x rejects in type check of x, so R_{ATM} doesn't accept x
2. $x = \langle M, q\rangle$
   (a) $w \in L(M)$
   By def of A_{TM}, $x \in A_{TM}$ and tracing R_{ATM}, x passes type check and R_{ATM} runs M on w as a subroutine. The subroutine halts and accepts, so R_{ATM} accepts x
   (b) $w \notin L(M)$
      i. M rejects w
      By def of A_{TM}, $x \notin A_{TM}$ and tracing R_{ATM}, x passes type check and R_{ATM} runs M on w as a subroutine. The subroutine halts and rejects, so R_{ATM} rejects x
      ii. M loops on w
      By def of A_{TM}, $x \notin A_{TM}$ and tracing R_{ATM}, x passes type check and R_{ATM} runs M on w as a subroutine. The subroutine doesn't halt so R_{ATM} doesn't halt on x, so it doesn't accept x

14.1.2 Other A_{TM} properties

- A_{TM} is not decidable
- $\bar{A_{TM}}$ is not recognizable
- $\bar{A_{TM}}$ is not decidable

15 Computable Functions and Mapping Reduction

For any languages A and B, A is **mapping reducible to** B means there is a computable function $f : \Sigma^* \to \Sigma^*$ such that *for all* strings x in Σ^*,

$$x \in A \qquad \text{if and only if} \qquad f(x) \in B.$$

Notation: when A is mapping reducible to B, we write $A \leq_m B$.
Intuition: $A \leq_m B$ means A is no harder than B, i.e. that the level of difficulty of A is less than or equal the level of difficulty of B.

A function $f : \Sigma^* \to \Sigma^*$ is a **computable function** means there is some Turing machine such that, for each x, on input x the Turing machine halts with exactly $f(x)$ followed by all blanks on the tape

15.1 Mapping Reduction Decidability

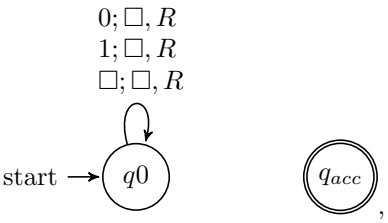
- If $A \leq_m B$ and B is decidable, then A is decidable.
- If $A \leq_m B$ and A is undecidable, then B is undecidable.
- For any language decidable language X and any set Y with at least one string string in Y and at least one string not in Y, $X \leq_m Y$, as witnessed by the identity function (one in and one out case)

15.2 Halting Problem

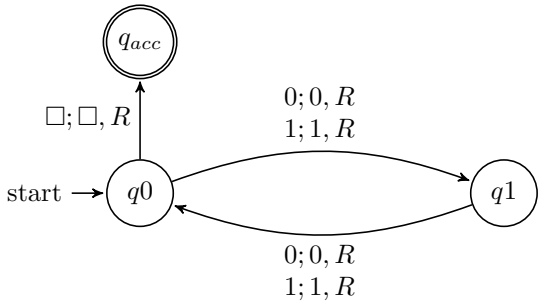
$$HALT_{TM} = \{\langle M, w\rangle \mid M \text{ is a Turing machine, } w \text{ is a string, and } M \text{ halts on } w\}$$

We know A_{TM} is undecidable. If we could prove that $A_{TM} \leq_m HALT_{TM}$ then we could conclude that $HALT_{TM}$ is undecidable too.
Define $F : \Sigma^* \to \Sigma^*$ by

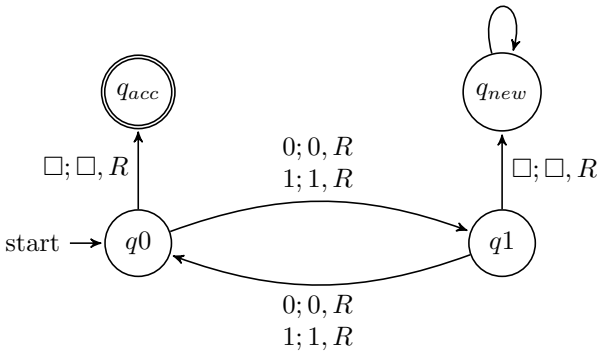
$$F(x) = \begin{cases} const_{out} & \text{if } x \neq \langle M, w\rangle \text{ for any Turing machine } M \text{ and string } w \text{ over the alphabet of } M \\ \langle M'_x, w\rangle & \text{if } x = \langle M, w\rangle \text{ for some Turing machine } M \text{ and string } w \text{ over the alphabet of } M. \end{cases}$$



where $const_{out} = \langle$ $, \varepsilon\rangle$ and M'_x is a Turing machine that computes like M except, if the computation of M ever were to go to a reject state, M'_x loops instead.



$F(\langle$ $, \varepsilon\rangle) = \langle$



$, \epsilon\rangle$

15.3 Mapping Reduction Recognizability

- If $A \leq_m B$ and B is recognizable, then A is recognizable.
- If $A \leq_m B$ and A is unrecognizable, then B is unrecognizable.
- To prove that a recognizable language R is undecidable, prove that $A_{TM} \leq_m R$.
- To prove that a co-recognizable language U is undecidable, prove that $\overline{A_{TM}} \leq_m U$, i.e. that $A_{TM} \leq_m \overline{U}$.

15.4 Computation Decidability Based on Reduction

- A_{TM} is recognizable, undecidable, and not-co-recognizable.
- $\overline{A_{TM}}$ is unrecognizable, undecidable, and co-recognizable.
- $HALT_{TM}$ is recognizable, undecidable, and not-co-recognizable.
- $\overline{HALT_{TM}}$ is unrecognizable, undecidable, and co-recognizable.
- E_{TM} is unrecognizable, undecidable, and co-recognizable.
- $\overline{E_{TM}}$ is recognizable, undecidable, and not-co-recognizable.
- EQ_{TM} is unrecognizable, undecidable, and not co-recognizable
- $\overline{EQ_{TM}}$ is unrecognizable, undecidable, and not co-recognizable

16 Other Models of Computation

Church-Turing Thesis: 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.

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

16.1 May-stay 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: never use the direction S!

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

16.2 Multitape Turing machine

A multitape Turing machine with k tapes can be formally represented 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.