Practice Problems for Final Exam: Solutions
CS 341: Foundations of Computer Science II
Prof. Marvin K. Nakayama

1. Short answers:

   (a) Define the following terms and concepts:

      i. Union, intersection, set concatenation, Kleene-star, set subtraction, complement
      Answer: Union: $S \cup T = \{ x \mid x \in S \text{ or } x \in T\}$
      Intersection: $S \cap T = \{ x \mid x \in S \text{ and } x \in T\}$
      Concatenation: $S \circ T = \{ xy \mid x \in S, y \in T\}$
      Kleene-star: $S^* = \{ w_1 w_2 \cdots w_k \mid k \geq 0, w_i \in S \ \forall\ i = 1, 2, \ldots, k\}$
      Subtraction: $S - T = \{ x \mid x \in S, x \notin T\}$
      Complement: $\overline{S} = \{ x \in \Omega \mid x \notin S\} = \Omega - S$, where Ω is the universe of all elements under consideration.

      ii. A set S is closed under an operation f
      Answer: S is closed under f if applying f to members of S always returns a member of S.

      iii. Regular language
      Answer: A regular language is defined by a DFA.

      iv. Kleene's theorem
      Answer: A language is regular if and only if it has a regular expression.

      v. Context-free language
      Answer: A CFL is defined by a CFG.

      vi. Chomsky normal form
      Answer: A CFG is in Chomsky normal form if each of its rules has one of 3 forms: $A \to BC$, $A \to x$, or $S \to \varepsilon$, where A, B, C are variables, B and C are not the start variable, x is a terminal, and S is the start variable.

      vii. Church-Turing Thesis
      Answer: The informal notion of algorithm corresponds exactly to a Turing machine that always halts (i.e., a decider).

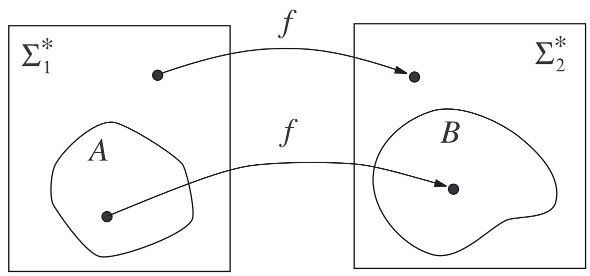
      viii. Turing-decidable language
      Answer: A language A that is decided by a Turing machine; i.e., there is a Turing machine M such that M halts and accepts on any input $w \in A$, and M halts and rejects on input input $w \notin A$; i.e., looping cannot happen.

      ix. Turing-recognizable language
      Answer: A language A that is recognized by a Turing machine; i.e., there is a Turing machine M such that M halts and accepts on any input $w \in A$, and M rejects or loops on any input $w \notin A$.

      x. co-Turing-recognizable language
      Answer: A language whose complement is Turing-recognizable.

xi. Countable and uncountable sets
Answer: A set S is countable if it is finite or we can define a correspondence between S and the positive integers. In other words, we can create a list of all the elements in S and each specific element will eventually appear in the list. An uncountable set is a set that is not countable. A common approach to prove a set is uncountable is by using a diagonalization argument.

xii. Language A is mapping reducible to language B, $A \leq_{\mathrm{m}} B$
Answer: Suppose A is a language defined over alphabet Σ_1, and B is a language defined over alphabet Σ_2. Then $A \leq_{\mathrm{m}} B$ means there is a computable function $f : \Sigma_1^* \to \Sigma_2^*$ such that $w \in A$ if and only if $f(w) \in B$. Thus, if $A \leq_{\mathrm{m}} B$, we can determine if a string w belongs to A by checking if $f(w)$ belongs to B.



$$w \in A \iff f(w) \in B$$

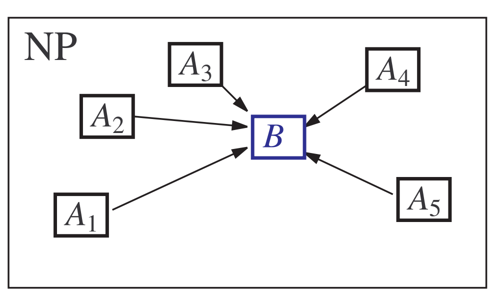
$$\text{YES instance for problem } A \iff \text{YES instance for problem } B$$

xiii. Function $f(n)$ is $O(g(n))$
Answer: There exist constants c and n_0 such that $|f(n)| \leq c \cdot g(n)$ for all $n \geq n_0$.

xiv. Classes P and NP
Answer: P is the class of languages that can be **decided** by a **deterministic** Turing machine in **polynomial time**. NP is the class of languages that can be **verified** in (deterministic) **polynomial time**. Equivalently, NP is the class of languages that can be **decided** by a **nondeterministic** Turing machine in **polynomial time**.

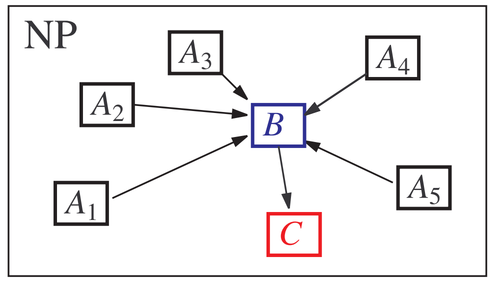
xv. Language A is polynomial-time mapping reducible to language B, $A \leq_{\mathrm{P}} B$.
Answer: Suppose A is a language defined over alphabet Σ_1, and B is a language defined over alphabet Σ_2. Then $A \leq_{\mathrm{P}} B$ means there is a polynomial-time computable function $f : \Sigma_1^* \to \Sigma_2^*$ such that $w \in A$ if and only if $f(w) \in B$.

xvi. NP-complete
Answer: Language B is NP-Complete if $B \in$ NP, and for every language $A \in$ NP, we have $A \leq_{\mathrm{P}} B$.



The typical approach to proving a language C is NP-Complete is as follows:

- First show $C \in$ NP by giving a deterministic polynomial-time verifier for C. (Alternatively, we can show $C \in$ NP by giving a nondeterministic polynomial-time decider for C.)
- Next show that a known NP-Complete language B can be reduced to C in polynomial time; i.e., $B \leq_{\mathrm{P}} C$.



Note that the second step implies that $A \leq_{\mathrm{P}} C$ for each $A \in$ NP Because we can first reduce A to B in polynomial time because B is NP-Complete, and then we can reduce B to C in polynomial time, so the entire reduction of A to C takes polynomial time.

xvii. NP-hard

Answer: Language B is NP-hard if for every language $A \in$ NP, we have $A \leq_{\mathrm{P}} B$.

(b) Give the transition functions δ (i.e., specify the domains and ranges) of a DFA, NFA, PDA, Turing machine and nondeterministic Turing machine.

Answer:

- DFA, $\delta : Q \times \Sigma \to Q$, where Q is the set of states and Σ is the alphabet.
- NFA, $\delta : Q \times \Sigma_\varepsilon \to \mathcal{P}(Q)$, where $\Sigma_\varepsilon = \Sigma \cup \{\varepsilon\}$ and $\mathcal{P}(Q)$ is the power set of Q
- PDA, $\delta : Q \times \Sigma_\varepsilon \times \Gamma_\varepsilon \to \mathcal{P}(Q \times \Gamma_\varepsilon)$, where Γ is the stack alphabet and $\Gamma_\varepsilon = \Gamma \cup \{\varepsilon\}$.
- Turing machine, $\delta : Q \times \Gamma \to Q \times \Gamma \times \{L, R\}$, where Γ is the tape alphabet, L means move tape head one cell left, and R means move tape head one cell right.
- Nondeterministic Turing machine, $\delta : Q \times \Gamma \to \mathcal{P}(Q \times \Gamma \times \{L, R\})$, where Γ is the tape alphabet, L means move tape head one cell left, and R means move tape head one cell right.

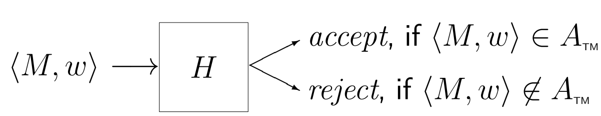
(c) Explain the "P vs. NP" problem.

Answer: P is the class of languages that can be solved in polynomial time, and NP is the class of languages that can be verified in polynomial time. We know that P $\subseteq$ NP. To see why, note that each TM is also an NTM, so deciding in deterministic polynomial time is a special case of deciding in nondeterministic polynomial time, making P $\subseteq$ NP. But it is currently unknown if P $=$ NP or P $\neq$ NP.

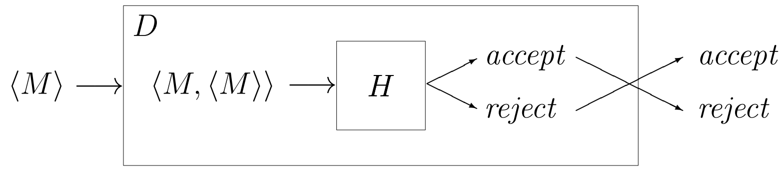
2. Recall that $A_{\mathrm{TM}} = \{ \langle M, w \rangle \mid M \text{ is a TM that accepts string } w \}$.

(a) Prove that A_{TM} is undecidable. You may not cite any theorems or corollaries in your proof.

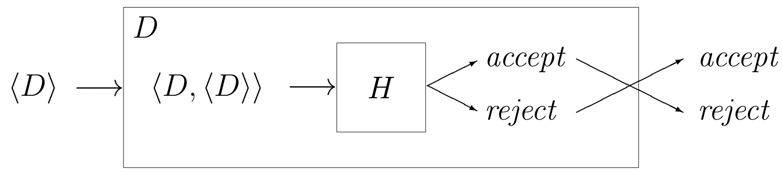
Overview of Proof: We use a proof by contradiction. Suppose A_{TM} is decided by some TM H, so H accepts $\langle M, w \rangle$ if TM M accepts w, and H rejects $\langle M, w \rangle$ if TM M doesn't accept w.



Define another TM D using H as a subroutine.



So D takes as input any encoded TM $\langle M\rangle$, then feeds $\langle M, \langle M\rangle\rangle$ as input into H, and finally outputs the opposite of what H outputs. Because D is a TM, we can feed $\langle D\rangle$ as input into D. What happens when we run D with input $\langle D\rangle$?



Note that D accepts $\langle D\rangle$ iff D doesn't accept $\langle D\rangle$, which is impossible. Thus, A_{TM} must be undecidable.

Complete Proof: Suppose there exists a TM H that decides A_{TM}. TM H takes input $\langle M, w\rangle$, where M is a TM and w is a string. If TM M accepts string w, then $\langle M, w\rangle \in A_{\text{TM}}$ and H accepts input $\langle M, w\rangle$. If TM M does not accept string w, then $\langle M, w\rangle \notin A_{\text{TM}}$ and H rejects input $\langle M, w\rangle$. Consider the language $L = \{\, \langle M\rangle \mid M \text{ is a TM that does not accept } \langle M\rangle \,\}$. Now construct a TM D for L using TM H as a subroutine:

D = "On input $\langle M\rangle$, where M is a TM:
1. Run H on input $\langle M, \langle M\rangle\rangle$.
2. If H accepts, *reject*. If H rejects, *accept*."

If we run TM D on input $\langle D\rangle$, then D accepts $\langle D\rangle$ if and only if D doesn't accept $\langle D\rangle$. Because this is impossible, TM H must not exist, so A_{TM} is undecidable.

(b) Show that A_{TM} is Turing-recognizable.

Answer: The universal TM U recognizes A_{TM}, where U is defined as follows:

U = "On input $\langle M, w\rangle$, where M is a TM and w is a string:
1. Run M on w.
2. If M accepts w, *accept*; if M rejects w, *reject*."

Note that U only recognizes A_{TM} and does not decide A_{TM} Because when we run M on w, there is the possibility that M neither accepts nor rejects w but rather loops on w.

3. Each of the languages below in parts (a), (b), (c), (d) is of one of the following types:

Type REG. It is regular.
Type CFL. It is context-free, but not regular.
Type DEC. It is Turing-decidable, but not context-free.

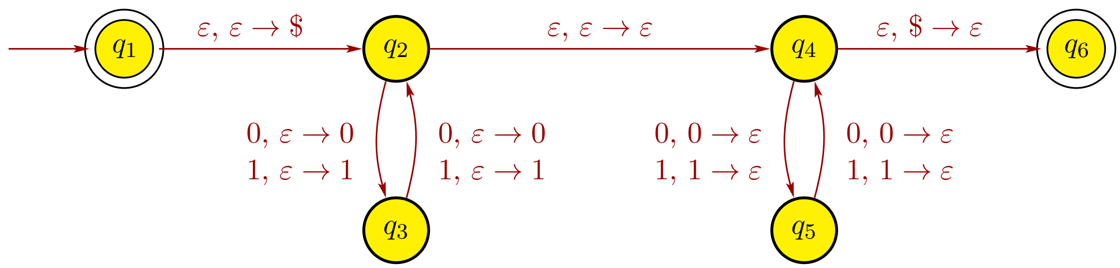
For each of the following languages, specify which type it is. Also, follow these instructions:

- If a language L is of Type REG, give a regular expression **and** a DFA (5-tuple) for L.
- If a language L is of Type CFL, give a context-free grammar (4-tuple) **and** a PDA (6-tuple) for L. **Also, prove that L is not regular.**
- If a language L is of Type DEC, give a description of a Turing machine that decides L. **Also, prove that L is not context-free.**

(a) $A = \{ w \in \Sigma^* \mid w = \text{reverse}(w) \text{ and the length of } w \text{ is divisible by } 4 \}$, where $\Sigma = \{0, 1\}$.

Circle one type: REG CFL DEC

Answer: A is of type CFL. A CFG $G = (V, \Sigma, R, S)$ for A has $V = \{S\}$, $\Sigma = \{0, 1\}$, starting variable S, and rules $R = \{ S \to 00S00 \mid 01S10 \mid 10S01 \mid 11S11 \mid \varepsilon \}$. A PDA for A is as follows:



The above PDA has 6-tuple $(Q, \Sigma, \Gamma, \delta, q_1, F)$, with $Q = \{q_1, q_2, \ldots, q_6\}$, $\Sigma = \{0, 1\}$, $\Gamma = \{0, 1, \$\}$, starting state q_1, $F = \{q_1, q_6\}$, and transition function $\delta : Q \times \Sigma_\varepsilon \times \Gamma_\varepsilon \to \mathcal{P}(Q \times \Gamma_\varepsilon)$ defined by

Input:	0				1				ε			
Stack:	0	1	\$	ε	0	1	\$	ε	0	1	\$	ε
q_1												$\{ (q_2, \$) \}$
q_2				$\{ (q_3, 0) \}$				$\{ (q_3, 1) \}$				$\{ (q_4, \varepsilon) \}$
q_3				$\{ (q_2, 0) \}$				$\{ (q_2, 1) \}$				
q_4	$\{ (q_5, \varepsilon) \}$					$\{ (q_5, \varepsilon) \}$					$\{(q_6, \varepsilon)\}$	
q_5	$\{ (q_4, \varepsilon) \}$					$\{ (q_4, \varepsilon) \}$						
q_6												

Blank entries are $\emptyset$.

We now prove that A is not regular by contradiction. Suppose that A is regular. Let $p \geq 1$ be the pumping length of the pumping lemma (Theorem 1.I). Consider string $s = 0^p\ 1^{2p}\ 0^p \in A$, and note that $|s| = 4p > p$, so the conclusions of the pumping lemma must hold. Thus, we can split $s = xyz$ satisfying conditions (1) $xy^iz \in A$ for all $i \geq 0$, (2) $|y| > 0$, and (3) $|xy| \leq p$. Because all of the first p symbols of s are 0s, (3) implies that x and y must only consist of 0s. Also, z must consist of the rest of the 0s at the beginning, followed by $1^{2p}0^p$. Hence, we can write $x = 0^j$, $y = 0^k$, $z = 0^m\ 1^{2p}\ 0^p$, where $j + k + m = p$ because $s = 0^p1^{2p}0^p = xyz = 0^j\ 0^k\ 0^m\ 1^{2p}\ 0^p$. Moreover, (2) implies that $k > 0$. Finally, (1) states that $xyyz$ must belong to A. However,

$$xyyz = 0^j\ 0^k\ 0^k\ 0^m\ 1^{2p}\ 0^p = 0^{p+k}\ 1^{2p}\ 0^p$$

because $j + k + m = p$. But, $k > 0$ implies reverse$(xyyz) \neq xyyz$, which means $xyyz \notin A$, which contradicts (1). Therefore, A is a nonregular language.

(b) $B = \{ b^n a^n b^n \mid n \geq 0 \}$.

Circle one type: REG CFL DEC

Answer: B is of type DEC. Below is a description of a Turing machine that decides B.

M = "On input string $w \in \{a, b\}^*$:

1. Scan the input from left to right to make sure that it is a member of $b^* a^* b^*$, and *reject* if it isn't.
2. Return tape head to left-hand end of tape.
3. Repeat the following until there are no more bs left on the tape.
4. Replace the leftmost b with x.
5. Scan right until an a occurs. If there are no a's, *reject.*
6. Replace the leftmost a with x.
7. Scan right until a b occurs. If there are no b's, *reject.*
8. Replace the leftmost b (after the a's) with x.
9. Return tape head to left-hand end of tape, and go to stage 3.
10. If the tape contains any a's, *reject.* Otherwise, *accept.*"

We now prove that B is not context-free by contradiction. Suppose that B is context-free. Let p be the pumping length of the pumping lemma for CFLs (Theorem 2.D), and consider string $s = b^p a^p b^p \in B$. Note that $|s| = 3p > p$, so the pumping lemma will hold. Thus, we can split $s = b^p a^p b^p = uvxyz$ satisfying $uv^ixy^iz \in B$ for all $i \geq 0$, $|vy| \geq 1$, and $|vxy| \leq p$. We now consider all of the possible choices for v and y:

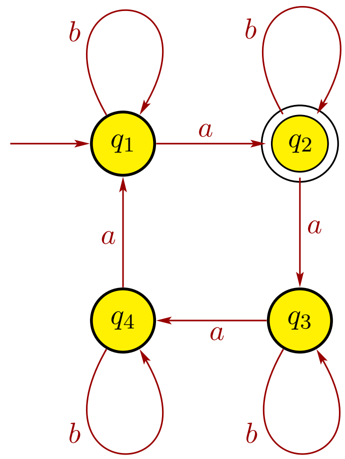
- Suppose strings v and y are uniform (e.g., $v = b^j$ for some $j \geq 0$, and $y = a^k$ for some $k \geq 0$). Then $|vy| \geq 1$ implies that $v \neq \varepsilon$ or $y \neq \varepsilon$ (or both), so uv^2xy^2z won't have the correct number of b's at the beginning, a's in the middle, and b's at the end. Hence, $uv^2xy^2z \notin B$.
- Now suppose strings v and y are not both uniform. Then uv^2xy^2z will not have the form $b \cdots ba \cdots ab \cdots b$. Hence, $uv^2xy^2z \notin B$.

Thus, there are no options for v and y such that $uv^ixy^iz \in B$ for all $i \geq 0$. This is a contradiction, so B is not a CFL.

(c) $C = \{ w \in \Sigma^* \mid n_a(w) \mod 4 = 1 \}$, where $\Sigma = \{a, b\}$ and $n_a(w)$ is the number of a's in string w. For example, $n_a(babaabb) = 3$. Also, recall $j \mod k$ returns the remainder after dividing j by k, e.g., $3 \mod 4 = 3$, and $9 \mod 4 = 1$.

Circle one type: REG CFL DEC

Answer: C is of type REG. A regular expression for C is $(b^*ab^*ab^*ab^*ab^*)^*b^*ab^*$, and a DFA for C is below:



The above DFA has 5-tuple $(Q, \Sigma, \delta, q_1, F)$, with $Q = \{q_1, q_2, q_3, q_4\}$, $\Sigma = \{a, b\}$, q_1 as the starting state, $F = \{q_2\}$, and transition function $\delta : Q \times \Sigma \to Q$ defined by

	a	b
q_1	q_2	q_1
q_2	q_3	q_2
q_3	q_4	q_3
q_4	q_1	q_4

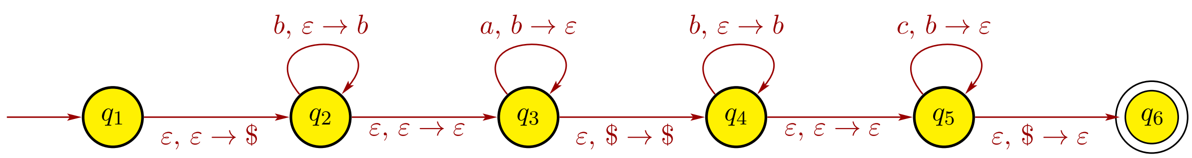
(d) $D = \{\, b^n a^n b^k c^k \mid n \geq 0,\ k \geq 0 \,\}$. [Hint: Recall that the class of context-free languages is closed under concatenation.]

Circle one type: REG CFL DEC

Answer: D is of type CFL. A CFG $G = (V, \Sigma, R, S)$ for D has $V = \{S, X, Y\}$, $\Sigma = \{a, b, c\}$, S as the starting variable, and rules R:

$$\begin{aligned} S &\to XY \\ X &\to bXa \mid \varepsilon \\ Y &\to bYc \mid \varepsilon \end{aligned}$$

A PDA for D is below:



We formally express the PDA as a 6-tuple $(Q, \Sigma, \Gamma, \delta, q_1, F)$, where $Q = \{q_1, q_2, \ldots, q_6\}$, $\Sigma = \{a, b, c\}$, $\Gamma = \{b, \$\}$ (use \$ to mark bottom of stack), q_1 is the start state, $F = \{q_6\}$, and the transition function $\delta : Q \times \Sigma_\varepsilon \times \Gamma_\varepsilon \to \mathcal{P}(Q \times \Gamma_\varepsilon)$ is defined by

Input:	a			b			c			ε		
Stack:	b	$\$$	ε	b	$\$$	ε	b	$\$$	ε	b	$\$$	ε
q_1												$\{(q_2,\$)\}$
q_2						$\{(q_2,b)\}$						$\{(q_3,\varepsilon)\}$
q_3	$\{(q_3,\varepsilon)\}$										$\{(q_4,\$)\}$	
q_4						$\{(q_4,b)\}$						$\{(q_5,\varepsilon)\}$
q_5							$\{(q_5,\varepsilon)\}$				$\{(q_6,\varepsilon)\}$	
q_6												

Blank entries are $\emptyset$.

An important point to note about the above PDA is that the transition from q_3 to q_4 pops and pushes \$. It is important to pop \$ to make sure that the number of a's matches the number of b's in the beginning. We need to push \$ to mark the bottom of the stack again for the second part of the string of b's and c's.

We now prove that D is not regular by contradiction. Suppose that D is regular. Let $p \geq 1$ be the pumping length of the pumping lemma (Theorem 1.I). Consider string $s = b^p\ a^p\ b^p\ c^p \in D$, and note that $|s| = 4p > p$, so the conclusions of the pumping lemma must hold. Thus, we can split $s = xyz$ satisfying (1) $xy^iz \in D$ for all $i \geq 0$, (2) $|y| > 0$, and (3) $|xy| \leq p$. Because all of the first p symbols of s are b's, (3) implies that x and y must only consist of b's. Also, z must consist of the rest of the b's at the beginning, followed by $a^p\ b^p\ c^p$. Hence, we can write $x = b^j$, $y = b^k$, $z = b^m\ a^p\ b^p\ c^p$, where $j + k + m = p$ because $s = b^p\ a^p\ b^p\ c^p = xyz = b^j\ b^k\ b^m\ a^p\ b^p\ c^p$. Moreover, (2) implies that $k > 0$. Finally, (1) states that $xyyz$ must belong to D. However,

$$xyyz = b^j\ b^k\ b^k\ b^m\ a^p\ b^p\ c^p = b^{p+k}\ a^p\ b^p\ c^p$$

because $j + k + m = p$. Also $k > 0$, so $xyyz \notin D$, which contradicts (1). Therefore, D is a nonregular language.

4. Each of the languages below in parts (a), (b), (c), (d) is of one of the following types:

Type DEC.	It is Turing-decidable.
Type TMR.	It is Turing-recognizable, but not decidable.
Type NTR.	It is not Turing-recognizable.

For each of the following languages, specify which type it is. Also, follow these instructions:

- If a language L is of Type DEC, give a description of a Turing machine that decides L.
- If a language L is of Type TMR, give a description of a Turing machine that recognizes L. **Also, prove that L is not decidable.**
- If a language L is of Type NTR, give a proof that it is not Turing-recognizable.

In each part below, if you need to prove that the given language L is decidable, undecidable, or not Turing-recognizable, you must give an explicit proof of this; i.e., do not just cite a theorem that establishes this without a proof. However, if in your proof you need to show another language L' has a particular property and there is a theorem that establishes this, then you may simply cite the theorem for L' without proof.

(a) $EQ_{\text{TM}} = \{ \langle M_1, M_2 \rangle \mid M_1, M_2 \text{ are TMs with } L(M_1) = L(M_2) \}$. [Hint: show $\overline{A_{\text{TM}}} \leq_{\text{m}} EQ_{\text{TM}}$.]

Circle one type: DEC TMR NTR

Answer: EQ_{TM} is of type NTR (see Theorem 5.K). We prove this by showing $\overline{A_{\text{TM}}} \leq_{\text{m}} EQ_{\text{TM}}$ and applying Corollary 5.I. Define the reducing function $f(\langle M, w \rangle) = \langle M_1, M_2 \rangle$, where

- $M_1 =$ "*reject* on all inputs."
- $M_2 =$ "On input x:
  1. Ignore input x, and run M on w.
  2. If M accepts w, *accept*;
     If M rejects w, *reject*."

Note that $L(M_1) = \emptyset$. For the language of TM M_2,

- if M accepts w (i.e., $\langle M, w\rangle \notin \overline{A_{\rm TM}}$), then $L(M_2) = \Sigma^*$;
- if M does not accept w (i.e., $\langle M, w\rangle \in \overline{A_{\rm TM}}$), then $L(M_2) = \emptyset$.

Thus, if $\langle M, w\rangle$ is a YES instance for $\overline{A_{\rm TM}}$ (i.e., $\langle M, w\rangle \in \overline{A_{\rm TM}}$, so M does not accept w), then $L(M_1) = \emptyset$ and $L(M_2) = \emptyset$, which are the same, implying that $f(\langle M, w\rangle) = \langle M_1, M_2\rangle \in EQ_{\rm TM}$ is a YES instance for $EQ_{\rm TM}$. Also, if $\langle M, w\rangle$ is a NO instance for $\overline{A_{\rm TM}}$ (i.e., $\langle M, w\rangle \notin \overline{A_{\rm TM}}$, so M accepts w), then $L(M_1) = \emptyset$ and $L(M_2) = \Sigma^*$, which are not the same, implying that $f(\langle M, w\rangle) = \langle M_1, M_2\rangle \notin EQ_{\rm TM}$ is a NO instance for $EQ_{\rm TM}$. Hence, we see that $\langle M, w\rangle \in \overline{A_{\rm TM}}$ $\iff$ $f(\langle M, w\rangle) = \langle M_1, M_2\rangle \in EQ_{\rm TM}$, so $\overline{A_{\rm TM}} \leq_{\rm m} EQ_{\rm TM}$. But $\overline{A_{\rm TM}}$ is not TM-recognizable (Corollary 4.M), so $EQ_{\rm TM}$ is not TM-recognizable by Corollary 5.I.

(b) $HALT_{\rm TM} = \{ \langle M, w\rangle \mid M \text{ is a TM that halts on input } w \}$. [Hint: modify the universal TM to show that $HALT_{\rm TM}$ is Turing-recognizable.]

Circle one type: DEC TMR NTR

Answer: $HALT_{\rm TM}$ is of type TMR (see Theorem 5.A). The following Turing machine recognizes $HALT_{\rm TM}$:

T = "On input $\langle M, w\rangle$, where M is a TM and w is a string:
1. Run M on w.
2. If M halts on w, *accept*."

We now prove that $HALT_{\rm TM}$ is undecidable, which is Theorem 5.A. Suppose there exists a TM R that decides $HALT_{\rm TM}$. Then we could use R to develop a TM S to decide $A_{\rm TM}$ by modifying the universal TM to first use R to see if it's safe to run M on w.

S = "On input $\langle M, w\rangle$, where M is a TM and w is a string:
1. Run R on input $\langle M, w\rangle$.
2. If R rejects, *reject*.
3. If R accepts, simulate M on input w until it halts.
4. If M accepts, *accept*; otherwise, *reject*."

Because TM R is a decider, TM S always halts and is a decider. Thus, deciding $A_{\rm TM}$ is reduced to deciding $HALT_{\rm TM}$. However, $A_{\rm TM}$ is undecidable (Theorem 4.I), so that must mean that $HALT_{\rm TM}$ is also undecidable.

(c) $EQ_{\rm DFA} = \{ \langle M_1, M_2\rangle \mid M_1, M_2 \text{ are DFAs with } L(M_1) = L(M_2) \}$.

Circle one type: DEC TMR NTR

Answer: EQ_{DFA} is of type DEC (see Theorem 4.E). The following TM decides EQ_{DFA}:

M = "On input $\langle A, B\rangle$, where A and B are DFAs:

0. Check if $\langle A, B\rangle$ is a proper encoding of 2 DFAs. If not, *reject.*
1. Construct DFA C such that
   $$L(C) = [L(A) \cap \overline{L(B)}] \cup [\overline{L(A)} \cap L(B)]$$
   using algorithms for DFA union, intersection and complementation.
2. Run TM decider for E_{DFA} (Theorem 4.D) on $\langle C\rangle$.
3. If $\langle C\rangle \in E_{\text{DFA}}$, *accept*; if $\langle C\rangle \notin E_{\text{DFA}}$, *reject.*"

(d) $\overline{A_{\text{TM}}}$, where $A_{\text{TM}} = \{\, \langle M, w\rangle \mid M \text{ is a TM that accepts string } w \,\}$.

Circle one type: DEC TMR NTR

Answer: $\overline{A_{\text{TM}}}$ is of type NTR, which is just Theorem 4.M. We prove this as follows. We know that A_{TM} is recognized by the universal Turing machine, so A_{TM} is Turing-recognizable. If $\overline{A_{\text{TM}}}$ were Turing-recognizable, then A_{TM} is co-Turing-recognizable. This makes A_{TM} both Turing-recognizable and co-Turing-recognizable. But then Theorem 4.L would imply that A_{TM} is decidable, which we know is not true by Theorem 4.I. Hence, $\overline{A_{\text{TM}}}$ is not Turing-recognizable.

5. Let $L_1, L_2, L_3, \ldots$ be an infinite sequence of regular languages, each of which is defined over a common input alphabet Σ. Let $L = \cup_{k=1}^{\infty} L_k$ be the infinite union of $L_1, L_2, L_3, \ldots$. Is it always the case that L is a regular language? If your answer is YES, give a proof. If your answer is NO, give a counterexample. Explain your answer. [Hint: Consider, for each $k \geq 0$, the language $L_k = \{a^k b^k\}$.]

   Answer: The answer is NO. For each $k \geq 1$, let $L_k = \{a^k b^k\}$, so L_k is a language consisting of just a single string $a^k b^k$. Because L_k is finite, it must be a regular language by Theorem 1.F. But $L = \cup_{k=1}^{\infty} L_k = \{\, a^k b^k \mid k \geq 1 \,\}$, which we know is not regular (see end of Chapter 1).

6. Let L_1, L_2, and L_3 be languages defined over the alphabet $\Sigma = \{a, b\}$, where

   - L_1 consists of all possible strings over Σ except the strings $w_1, w_2, \ldots, w_{100}$; i.e., start with all possible strings over the alphabet, take out 100 particular strings, and the remaining strings form the language L_1;
   - L_2 is recognized by an NFA; and
   - L_3 is recognized by a PDA.

   Prove that $(L_1 \cap L_2)L_3$ is a context-free language. [Hint: First show that L_1 and L_2 are regular. Also, consider $\overline{L_1}$, the complement of L_1.]

   Answer: Note that $\overline{L_1} = \{w_1, w_2, \ldots, w_{100}\}$, so $|\overline{L_1}| = 100$. Thus, $\overline{L_1}$ is a regular language because it is finite by Theorem 1.F. Then Theorem 1.H implies that the complement of $\overline{L_1}$ must be regular, but the complement of $\overline{L_1}$ is L_1. Thus, L_1 is regular. Language L_2 has an NFA, so it also has a DFA by Theorem 1.C. Therefore, L_2 is regular. Because L_1 and L_2 are regular, $L_1 \cap L_2$ must be regular by Theorem 1.G. Theorem 2.B then implies that $L_1 \cap L_2$ is context-free. Because L_3 has a PDA, L_3 is context-free by Theorem 2.C. Hence, because $L_1 \cap L_2$ and L_3 are both context-free, their concatenation is context-free by Theorem 2.F.

7. Write Y or N in the entries of the table below to indicate which classes of languages are closed under which operations.

Operation	Regular languages	CFLs	Decidable languages	Turing-recognizable languages
Union	Y (Thm 1.A)	Y (Thm 2.E)	Y (HW 7, prob 2a)	Y (HW 7, prob 2b)
Intersection	Y (Thm 1.G)	N (HW 6, prob 2a)	Y	Y
Complementation	Y (Thm 1.H)	N (HW 6, prob 2b)	Y	N (e.g., A_{TM})

We now prove the three "Y" entries that we haven't established before. We first prove the class of decidable languages is closed under intersection. Suppose a TM M_1 decides language L_1, and a TM M_2 decides language L_2. Then the following TM decides $L_1 \cap L_2$:

M' = "On input string w:

1. Run M_1 on input w, and run M_2 on input w.

2. If both M_1 and M_2 accept, *accept*. Otherwise, *reject*.

M' accepts w if both M_1 and M_2 accept it. If either rejects, M' rejects. The key here is that in stage 1 of M', both M_1 and M_2 are guaranteed to halt because both are deciders, so M' will also always halt, making it a decider. (Alternatively, we can change stage 1 to run M_1 and M_2 in parallel (alternating steps), both on input w, but this isn't necessary because M_1 and M_2 are deciders. In contrast, when we proved that the class of Turing-recognizable languages is closed under union, we did need to run M_1 and M_2 in parallel, both on input w, because if we didn't, then M_1 might loop forever on w, but M_2 might accept w.)

We now prove the class of decidable languages is closed under complementation. Suppose a TM M decides language L. Now create another TM M' that just swaps the accept and reject states of M. Because M is a decider, it always halts, so then M' also always halts. Thus, M' decides $\overline{L}$.

We now prove the class of Turing-recognizable languages is closed under intersection. Suppose a TM M_1 recognizes language L_1, and a TM M_2 recognizes language L_2. Then the following TM recognizes $L_1 \cap L_2$:

M' = "On input string w:

1. Run M_1 on input w, and run M_2 on input w.

2. If both M_1 and M_2 accept, *accept*. Otherwise, *reject*.

M' accepts w if both M_1 and M_2 accept it. If either rejects, M' rejects. But note that if M_1 or M_2 loops on w, then M' also loops on w. Hence, M' recognizes $L_1 \cap L_2$ but doesn't necessarily decide $L_1 \cap L_2$.

8. Consider the following context-free grammar G in Chomsky normal form:

$$
\begin{aligned}
S &\rightarrow a \mid YZ \\
Z &\rightarrow ZY \mid a \\
Y &\rightarrow b \mid ZZ \mid YY
\end{aligned}
$$

Use the CYK (dynamic programming) algorithm to fill in the following table to determine if G generates the string $babba$. Does G generate $babba$?

	1	2	3	4	5
1	Y	S	S	S	Y
2		S, Z	Z	Z	Y
3			Y	Y	S
4				Y	S
5					S, Z
	b	a	b	b	a

No, G does not generate *babba* because S is not in the upper right corner.

9. Recall that

$$\begin{aligned} \mathit{CLIQUE} &= \{\, \langle G, k\rangle \mid G \text{ is an undirected graph with a } k\text{-clique}\,\}, \\ \mathit{3SAT} &= \{\, \langle \phi \rangle \mid \phi \text{ is a satisfiable 3cnf-function}\,\}. \end{aligned}$$

Show that *CLIQUE* is NP-Complete by showing that *CLIQUE* $\in$ NP and *3SAT* $\leq_{\mathrm{P}}$ *CLIQUE*. Explain your reduction for the general case and not just for a specific example. Be sure to prove your reduction works and that it requires polynomial time. Also, be sure to provide proofs of these results, and don't just cite a theorem.

Answer:
Step 1: show that *CLIQUE* $\in$ NP. We accomplish by giving a polynomial-time verifier for *CLIQUE*. The following verifier V for *CLIQUE* uses the k-clique as the certificate c.

V = "On input $\langle\langle G, k\rangle, c\rangle$:

1. Test whether c is a set of k different nodes in G.
2. Test whether G contains all edges connecting nodes in c.
3. If both tests pass, *accept*; otherwise, *reject*."

We now show that the verifier V runs in deterministic polynomial time in the size of $\langle G, k\rangle$. First we need to measure the size of the encoding $\langle G, k\rangle$, which depends on the particular graph G and the encoding scheme. Suppose the graph G has m nodes, and assume that G is encoded as a list of nodes followed by a list of edges. To determine the size of the encoding $\langle G\rangle$ of the graph G, note that each edge in G corresponds to a pair of nodes, so G has $O(m^2)$ edges. Therefore, the size of $\langle G\rangle$ is $m+O(m^2) = O(m^2)$, but because we don't care about polynomial differences, we can simply measure the size of the $\langle G\rangle$ as m. Now we analyze the time complexity of V. In Stage 1, for each of the k nodes in c, we have to go through the m nodes in G, so Stage 1 of V takes $O(k)O(m) = O(km)$ time. For Stage 2, for each of the $\binom{k}{2} = k(k-1)/2 = O(k^2)$ pairs of nodes in c that we have to consider, we have to go through the list of $O(m^2)$ edges of G, so Stage 2 takes $O(k^2)O(m^2) = O(k^2m^2)$ time. Thus, the verifier V runs in (deterministic) polynomial time.

Step 2: show that *3SAT* $\leq_{\mathrm{m}}$ *CLIQUE*. Next we show how to reduce *3SAT* to *CLIQUE*. We need to convert an instance of the *3SAT* problem to an instance of the *CLIQUE* problem, with the property that a YES instance for *3SAT* maps to a YES instance of *CLIQUE*, and a NO instance for *3SAT* maps to a NO instance of *CLIQUE*. An instance of *3SAT* is a 3cnf-formula ϕ, and ϕ is a YES instance for *3SAT* if ϕ is satisfiable, and ϕ is a NO instance for *3SAT* if ϕ is not satisfiable. An instance of *CLIQUE* is a pair $\langle G, k\rangle$ of a graph G and an integer k, and $\langle G, k\rangle$ is a YES instance for *CLIQUE* if G has a clique of size k, and $\langle G, k\rangle$ is a NO instance for *CLIQUE* if G doesn't have a clique of size k. Thus, the reduction f needs to map each 3cnf-formula to a graph and number k, i.e., $f(\langle\phi\rangle) = \langle G, k\rangle$, where each YES instance $\langle\phi\rangle$ maps to a YES instance $\langle G, k\rangle$, and each NO instance $\langle\phi\rangle$ maps to a NO instance $\langle G, k\rangle$.

The reduction works as follows. Suppose that ϕ is a 3cnf-formula with k clauses. From ϕ, construct a graph G having a node for each literal in ϕ. Arrange the nodes in triples, where each triple corresponds to the literals from one clause. Add edges between every pair of nodes in G except when the nodes are from the same triple, or when the nodes are contradictory, e.g., x_i and $\overline{x_i}$.

To prove that this mapping is indeed a reduction, we need to show that $\langle\phi\rangle \in \mathit{3SAT}$ if and only if $\langle G, k\rangle \in \mathit{CLIQUE}$. Note that ϕ is satisfiable if and only if every clause has at least one true literal. Suppose ϕ is satisfiable, so it is a YES instance for $\mathit{3SAT}$. For each triple of nodes, choose a node corresponding to a true literal in the corresponding clause. This results in choosing k nodes, with exactly one node from each triple. This collection of k nodes is a k-clique because the graph G has edges between every pair of nodes except those in the same triple and not between contradictory literals. Thus, the resulting graph and number k is a YES instance for CLIQUE, so $\langle\phi\rangle \in \mathit{3SAT}$ implies $\langle G, k\rangle \in \mathit{CLIQUE}$.

Now we show the converse: each NO instance for $\mathit{3SAT}$ maps to a NO instance for CLIQUE, which is equivalent to $\langle G, k\rangle \in \mathit{CLIQUE}$ implying that $\langle\phi\rangle \in \mathit{3SAT}$. Suppose that G has a k-clique. The k nodes must be from k different triples because G has no edges between nodes in the same triple. Thus, the k literals corresponding to the k nodes in the k-clique come from k different clauses. Also, because G does not have edges between contradictory literals, setting the literals corresponding to the k nodes to true will lead to ϕ evaluating to true, so $\langle\phi\rangle \in \mathit{3SAT}$. Thus, $\langle G, k\rangle \in \mathit{CLIQUE}$ implies $\langle\phi\rangle \in \mathit{3SAT}$. Combining this with the proof from the last paragraph, we have shown $\langle\phi\rangle \in \mathit{3SAT}$ if and only if $\langle G, k\rangle \in \mathit{CLIQUE}$, so our approach for converting an instance of the $\mathit{3SAT}$ problem into an instance of the CLIQUE problem is indeed a reduction; i.e., $\mathit{3SAT} \leq_{\mathrm{m}} \mathit{CLIQUE}$.

Step 3: show that reduction $\mathit{3SAT} \leq_{\mathrm{m}} \mathit{CLIQUE}$ takes polynomial time. In other words, we have to show that the time to convert an instance $\langle\phi\rangle$ of the $\mathit{3SAT}$ problem to an instance $\langle G, k\rangle$ of the CLIQUE problem is polynomial in the size of the 3cnf-formula ϕ. We can measure the size of ϕ in terms of its number k of clauses and its number m of variables. The constructed graph G has a node for every literal in ϕ, and because ϕ has k clauses, each with exactly 3 literals, G has $3k$ nodes. We then add edges between each pair of nodes in G except for those between nodes in the same triple nor between contradictory literals. So the number of edges in G is strictly less than $\binom{3k}{2} = 3k(3k-1)/2 = O(k^2)$, so the time to construct G is polynomial in m and k. Thus, $\mathit{3SAT} \leq_{\mathrm{P}} \mathit{CLIQUE}$.

10. Recall that

$$\mathit{ILP} = \{\, \langle A, b\rangle \mid \text{matrix } A \text{ and vector } b \text{ satisfy } Ay \leq b \text{ with } y \text{ and integer vector} \,\}.$$

Show that ILP is NP-Complete by showing that $\mathit{ILP} \in$ NP and $\mathit{3SAT} \leq_{\mathrm{P}} \mathit{ILP}$. Explain your reduction for the general case and not just for a specific example. Be sure to prove your reduction works and that it requires polynomial time. Also, be sure to provide proofs of these results, and don't just cite a theorem.

Answer:
Step 1: show that $\mathit{ILP} \in$ NP. To do this, we now give a polynomial-time verifier V using as a certificate an integer vector c such that $Ac \leq b$. Here is a verifier for ILP:

V = "On input $\langle\langle A, b\rangle, c\rangle$:

1. Test whether c is a vector of all integers.
2. Test whether $Ac \leq b$.

3. If both tests pass, *accept*; otherwise, *reject*."

If $Ay \le b$ has m inequalities and n variables, we measure the size of the instance $\langle A, b\rangle$ as (m, n). Stage 1 of V takes $O(n)$ time, and Stage 2 takes $O(mn)$ time. Hence, verifier V has $O(mn)$ running time, which is polynomial in size of problem.

Step 2: show *3SAT* $\le_{\mathrm{m}}$ *ILP*. (We later show the reduction takes polynomial time.) To prove that *3SAT* $\le_{\mathrm{m}}$ *ILP*, we need an algorithm that takes any instance ϕ of the *3SAT* problem and converts it into an instance of the *ILP* problem such that $\langle\phi\rangle \in$ *3SAT* if and only if the constructed integer linear program has an integer solution. Suppose that ϕ has k clauses and m variables $x_1, x_2, \ldots, x_m$. For the integer linear program, define $2m$ variables $y_1, y_1', y_2, y_2', \ldots, y_m, y_m'$. Each y_i corresponds to x_i, and each y_i' corresponds to $\overline{x_i}$. For each $i = 1, 2, \ldots, m$, define the following inequality and equality relations to be satisfied in the integer linear program:

$$0 \le y_i \le 1, \quad 0 \le y_i' \le 1, \quad y_i + y_i' = 1. \tag{1}$$

If y_i must be integer-valued and $0 \le y_i \le 1$, then y_i can only take on the value 0 or 1. Similarly, y_i' can only take on the value 0 or 1. Hence, $y_i + y_i' = 1$ ensures exactly one of the pair (y_i, y_i') is 1 and the other is 0. This corresponds exactly to what x_i and $\overline{x_i}$ must satisfy.

Each clause in ϕ has the form $(x_i \vee \overline{x_j} \vee x_k)$. For each such clause, create a corresponding inequality

$$y_i + y_j' + y_k \ge 1 \tag{2}$$

to be included in the integer linear program. This ensures that each clause has at least one true literal. By construction, ϕ is satisfiable if and only if the constructed integer linear program with m sets of relations in display (1) and k inequations as in display (2) has an integer solution. Hence, we have shown *3SAT* $\le_{\mathrm{m}}$ *ILP*.

Step 3: show that the time to construct the integer linear program from a 3cnf-function ϕ is polynomial in the size of $\langle\phi\rangle$. We measure the size of $\langle\phi\rangle$ in terms of the number m of variables and the number k of clauses in ϕ. For each $i = 1, 2, \ldots, m$, display (1) comprises 6 inequalities:

- $y_i \ge 0$ (rewritten as $-y_i \le 0$),
- $y_i \le 1$,
- $y_i' \ge 0$ (rewritten as $-y_i' \le 0$),
- $y_i' \le 1$,
- $y_i + y_i' \le 1$, and
- $y_i + y_i' \ge 1$ (rewritten as $-y_i - y_i' \le -1$),

where the last two together are equivalent to $y_i + y_i' = 1$. Thus, we have $6m$ inequalities corresponding to display (1). The k clauses in ϕ leads to k more inequalities, each of the form in display (2). Thus, the constructed integer linear program has $2m$ variables and $6m + k$ linear inequalities, so the size of the resulting integer linear program is polynomial in m and k. Hence, the reduction takes polynomial time.

List of Theorems

Thm 1.A. The class of regular languages is closed under union.

Thm 1.B. The class of regular languages is closed under concatenation.

Thm 1.C. Every NFA has an equivalent DFA.

Thm 1.D. The class of regular languages is closed under Kleene-star.

Thm 1.E. (Kleene's Theorem) Language A is regular iff A has a regular expression.

Thm 1.F. If A is finite language, then A is regular.

Thm 1.G. The class of regular languages is closed under intersection.

Thm 1.H. The class of regular languages is closed under complementation.

Thm 1.I. (Pumping lemma for regular languages) If A is regular language, then $\exists$ number p where, if $s \in A$ with $|s| \geq p$, then $\exists$ strings x, y, z such that $s = xyz$ and (1) $xy^iz \in A$ for each $i \geq 0$, (2) $|y| > 0$, and (3) $|xy| \leq p$.

Thm 2.A. Every CFL can be described by a CFG $G = (V, \Sigma, R, S)$ in Chomsky normal form, i.e., each rule in G has one of two forms: $A \to BC$ or $A \to x$, where $A \in V$, $B, C \in V - \{S\}$, $x \in \Sigma$, and we also allow the rule $S \to \varepsilon$.

Thm 2.B. If A is a regular language, then A is also a CFL.

Thm 2.C. A language is context free iff some PDA recognizes it.

Thm 2.D. (Pumping lemma for CFLs) For every CFL L, $\exists$ pumping length p such that $\forall$ strings $s \in L$ with $|s| \geq p$, we can write $s = uvxyz$ with (1) $uv^ixy^iz \in L \ \forall\ i \geq 0$, (2) $|vy| \geq 1$, (3) $|vxy| \leq p$.

Thm 2.E. The class of CFLs is closed under union.

Thm 2.F. The class of CFLs is closed under concatenation.

Thm 2.G. The class of CFLs is closed under Kleene-star.

Thm 3.A. For every multi-tape TM M, there is a single-tape TM M' such that $L(M) = L(M')$.

Thm 3.B. Every NTM has an equivalent deterministic TM.

Cor 3.C. Language L is Turing-recognizable iff an NTM recognizes it.

Thm 3.D. A language is enumerable iff some enumerator enumerates it.

Church-Turing Thesis. The informal notion of algorithm is the same as Turing machine algorithm.

Thm 4.A. $A_{\text{DFA}} = \{ \langle B, w \rangle \mid B \text{ is a DFA that accepts string } w \}$ is Turing-decidable.

Thm 4.B. $A_{\text{NFA}} = \{ \langle B, w \rangle \mid B \text{ is an NFA that accepts string } w \}$ is Turing-decidable.

Thm 4.C. $A_{\text{REX}} = \{ \langle R, w \rangle \mid R \text{ is a regular expression that generates string } w \}$ is Turing-decidable.

Thm 4.D. $E_{\text{DFA}} = \{ \langle B \rangle \mid B \text{ is a DFA with } L(B) = \emptyset \}$ is Turing-decidable.

Thm 4.E. $EQ_{\text{DFA}} = \{ \langle A, B \rangle \mid A \text{ and } B \text{ are DFAs with } L(A) = L(B) \}$ is Turing-decidable.

Thm 4.F. $A_{\text{CFG}} = \{ \langle G, w \rangle \mid G \text{ is a CFG that generates string } w \}$ is Turing-decidable.

Thm 4.G. $E_{\text{CFG}} = \{ \langle G \rangle \mid G \text{ is a CFG with } L(G) = \emptyset \}$ is Turing-decidable.

Thm 4.H. Every CFL is Turing-decidable.

Thm 4.I. $A_{\text{TM}} = \{ \langle M, w \rangle \mid M \text{ is a TM that accepts string } w \}$ is undecidable.

Thm 4.J. The set $\mathcal{R}$ of all real numbers is uncountable.

Cor 4.K. Some languages are not Turing-recognizable.

Thm 4.L. A language is decidable iff it is both Turing-recognizable and co-Turing-recognizable.

Cor 4.M. $\overline{A_{\mathrm{TM}}}$ is not Turing-recognizable.

Thm 5.A. $\mathit{HALT}_{\mathrm{TM}} = \{ \langle M, w \rangle \mid M \text{ is a TM that halts on } w \}$ is undecidable.

Thm 5.B. $E_{\mathrm{TM}} = \{ \langle M \rangle \mid M \text{ is a TM with } L(M) = \emptyset \}$ is undecidable.

Thm 5.C. $\mathit{REG}_{\mathrm{TM}} = \{ \langle M \rangle \mid M \text{ is a TM and } L(M) \text{ is regular} \}$ is undecidable.

Thm 5.D. $\mathit{EQ}_{\mathrm{TM}} = \{ \langle M_1, M_2 \rangle \mid M_1, M_2 \text{ are TMs with } L(M_1) = L(M_2) \}$ is undecidable.

Thm 5.E. (Rice's Thm.) Let $\mathcal{P}$ be any subset of the class of Turing-recognizable languages such that $\mathcal{P} \neq \emptyset$ and $\overline{\mathcal{P}} \neq \emptyset$. Then $L_{\mathcal{P}} = \{ \langle M \rangle \mid L(M) \in \mathcal{P} \}$ is undecidable.

Thm 5.F. If $A \leq_{\mathrm{m}} B$ and B is Turing-decidable, then A is Turing-decidable.

Cor 5.G. If $A \leq_{\mathrm{m}} B$ and A is undecidable, then B is undecidable.

Thm 5.H. If $A \leq_{\mathrm{m}} B$ and B is Turing-recognizable, then A is Turing-recognizable.

Cor 5.I. If $A \leq_{\mathrm{m}} B$ and A is not Turing-recognizable, then B is not Turing-recognizable.

Thm 5.J. $E_{\mathrm{TM}} = \{ \langle M \rangle \mid M \text{ is a TM with } L(M) = \emptyset \}$ is not Turing-recognizable.

Thm 5.K. $\mathit{EQ}_{\mathrm{TM}} = \{ \langle M_1, M_2 \rangle \mid M_1, M_2 \text{ are TMs with } L(M_1) = L(M_2) \}$ is neither Turing-recognizable nor co-Turing-recognizable.

Thm 7.A. Let $t(n)$ be a function with $t(n) \geq n$. Then any $t(n)$-time multi-tape TM has an equivalent $O(t^2(n))$-time single-tape TM.

Thm 7.B. Let $t(n)$ be a function with $t(n) \geq n$. Then any $t(n)$-time NTM has an equivalent $2^{O(t(n))}$-time deterministic 1-tape TM.

Thm 7.C. $\mathit{PATH} \in \mathrm{P}$.

Thm 7.D. $\mathit{RELPRIME} \in \mathrm{P}$.

Thm 7.E. Every CFL is in P.

Thm 7.F. A language is in NP iff it is decided by some nondeterministic polynomial-time TM.

Cor 7.G. $\mathrm{NP} = \bigcup_{k \geq 0} \mathrm{NTIME}(n^k)$

Thm 7.H. $\mathit{CLIQUE} \in \mathrm{NP}$.

Thm 7.I. $\mathit{SUBSET\text{-}SUM} \in \mathrm{NP}$.

Thm 7.J. If $A \leq_{\mathrm{P}} B$ and $B \in \mathrm{P}$, then $A \in \mathrm{P}$.

Thm 7.K. $\mathit{3SAT}$ is polynomial-time reducible to CLIQUE.

Thm 7.L. If there is an NP-Complete problem B and $B \in \mathrm{P}$, then $\mathrm{P} = \mathrm{NP}$.

Thm 7.M. If B is NP-Complete and $B \leq_{\mathrm{P}} C$ for $C \in \mathrm{NP}$, then C is NP-Complete.

Thm 7.N. (Cook-Levin Thm.) SAT is NP-Complete.

Cor 7.O. $\mathit{3SAT}$ is NP-Complete.

Cor 7.P. CLIQUE is NP-Complete.

Thm 7.Q. ILP is NP-Complete.