Computability - Exercise 1

All answers should be proved formally (unless noted otherwise)

Due - March 12

1. What is the language of the automaton below? (remember to prove your answer formally.)

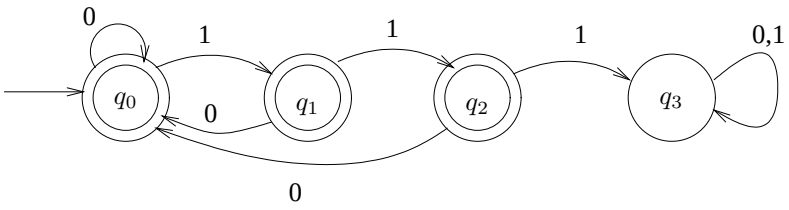


Figure 1: The automaton $\mathcal{A}$

2. Describe a deterministic finite automaton (a.k.a. DFA), for each of the following languages. A drawing can be considered a description, but only if it is *exact* and contains all the information needed to create from it the formal definition of the automaton.

   No need to prove formally the correctness of your construction.
   All languages are over the alphabet $\Sigma = \{0, 1\}$.

   (a) The language that contains all words that end with 0111.

   (b) The language that contains all words that begin with 0111.

   (c) The language that contains all words that contain 0111 as a subword.

   (d) The language that contains all words that do not contain 0111 as a subword.

3. The binary representation of a number is a word of the alphabet $\{0, 1\}$. The value of the word $s = s_1 \dots s_n$, denoted $v(s)$, is defined as follows: For

the empty word, $v(\epsilon) = 0$. For a longer word, the definition is inductive $v(s_1 \dots s_{n-1} s_n) = 2 \cdot v(s_1 \dots s_{n-1}) + s_n$. Note that this means that $v(s_1 \dots s_n) = \sum_{i=1}^{n} 2^{n-i} \cdot s_i$.

Describe a DFA that recognizes binary words that their value mod 3 is 0.

4. Let S be a set. A relation $E \subseteq S \times S$ is a *partition* relation, if there exists some partition of S into a set of subsets $\{S_i\}_{i \in I}$ (where I is a set of indices), such that:

   (a) the set S is covered by the subsets (i.e. $S = \bigcup_{i \in I} S_i$).

   (b) the subsets are disjoint, (i.e., for every $i, j \in I$ for which $i \neq j$ it holds that $S_i \cap S_j = \emptyset$).

   (c) for all $s_1, s_2 \in S$ it holds that $s_1 E s_2$ iff there exists an index $i \in I$ for which $s_1, s_2 \in S_i$. Equivalently, $E = \bigcup_{i \in I} S_i \times S_i$.

   A relation $E \subseteq S \times S$ is an *inverse function* relation, if there exists some set T and a function $f : S \to T$ such that: for all $s_1, s_2 \in S$ it holds that $s_1 E s_2$ iff $f(s_1) = f(s_2)$. Equivalently, $E = \bigcup_{t \in T} (f^{-1}(t) \times f^{-1}(t))$.

   Prove that a relation is an equivalence relation iff it is a partition relation, and that a relation is a partition relation iff it is an inverse function relation.

   Comment: As you shall prove, every equivalence relation is a partition relation. The sets S_i in the partition are called the *equivalence classes* of the equivalence relation.

5. optional ("reshut")
   Prove that there exists a non-regular language over the alphabet $\{0, 1\}$. Hints:

   (a) Prove that the set of all languages over the alphabet $\{0, 1\}$ is not countable.

   (b) Prove that the set of all deterministic finite automata over the alphabet $\{0, 1\}$ is countable.

שאלה 1

השפה המזוהה ע"י A היא כל המילים הבינאריות שאינן כוללת את תת-המילה 111.

נוכיח באינדוקציה על n כי לאחר קריאת מילה $w = w_1 \dots w_n$ באורך $n \geq 0$, האוטומט נמצא במצב:

א. q_3 אם המילה כוללת את 111 כתת-מילה.
ב. q_2 אם המילה אינה כוללת את 111 כתת-מילה וכן $w_{n-1}w_n=11$.
ג. q_1 אם המילה אינה כוללת את 111 כתת-מילה וכן $w_n=1$ אבל $w_{n-1}=0$ או $n=1$.
ד. q_0 אם המילה אינה כוללת את 111 כתת-מילה וכן $w_n=0$ או $n=0$.

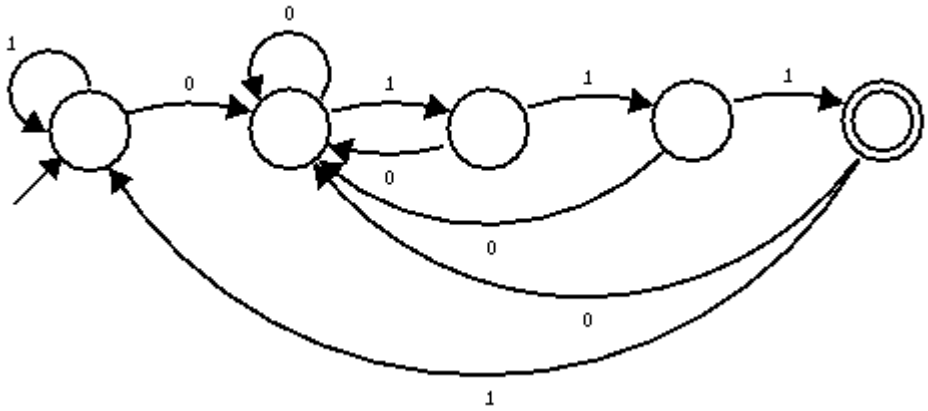
שימו לב שהתנאים בצד שמאל, מהוים חלוקה של כל האפשרויות למילים מהצורה $w_1 \dots w_n$.
בסיס: עבור $n=0$ כלומר המילה הריקה, ואכן לאחר "קריאתה" האוטומט נמצא במצב התחילי שהוא q_0.
צעד אינדוקציה: ראשית, אם המילה כוללת את 111 כתת-מילה אז או ש$w_1 \dots w_{n-1}$ כבר כוללת את 111 כתת-מילה, ואז ע"פ הנחת האינדוקציה, האוטומט נמצא בq_3 לפני קריאת w_n, ובגלל מבנה האוטומט הוא ישאר שם אחרי קריאת w_n, או שהמילה כוללת את 111 כתת-מילה אך $w_1 \dots w_{n-1}$ אינה כוללת את 111 כתת מילה, ואז בהכרח $w_{n-2}w_{n-1}=11$. לכן ע"פ הנחת האינדוקציה, לאחר קריאת $w_1 \dots w_{n-1}$ האוטומט יהיה במצב q_2, ולאחר קריאת w_n (שחייב להיות 1), האוטומט יעבור למצב q_3.
אחרת, w אינה כוללת את 111 כתת-מילה, ולכן גם $w_1 \dots w_{n-1}$ אינה כוללת את 111 כתת-מילה ולכן, ע"פ הנחת האינדוקציה לאחר קריאת $w_1 \dots w_{n-1}$ האוטומט אינו נמצא בq_3 . כעת:

- אם $w_n=0$ אז מכל מצב שהאוטומט עשוי להמצא בו הוא יעבור ל q_0, כנדרש (ד).
- אם $w_n=1$ ו $w_{n-1}=0$ או $n=1$, אז, ע"פ הנחת האינדוקציה, לאחר קריאת $w_1 \dots w_{n-1}$ האוטומט יהיה במצב q_0. לכן, לאחר קריאת w_n האוטומט יעבור למצב q_1.
- אם $w_{n-1}w_n=11$ אז בהכרח $w_{n-2}=0$ או $n=2$. לכן $w_1 \dots w_{n-1}$ מקיימת את תנאי (ג), ולכן לאחר קריאתה האוטומט יהיה במצב q_1. מכאן, שאחרי קריאת w_n האוטומט יעבור למצב q_2, כנדרש.

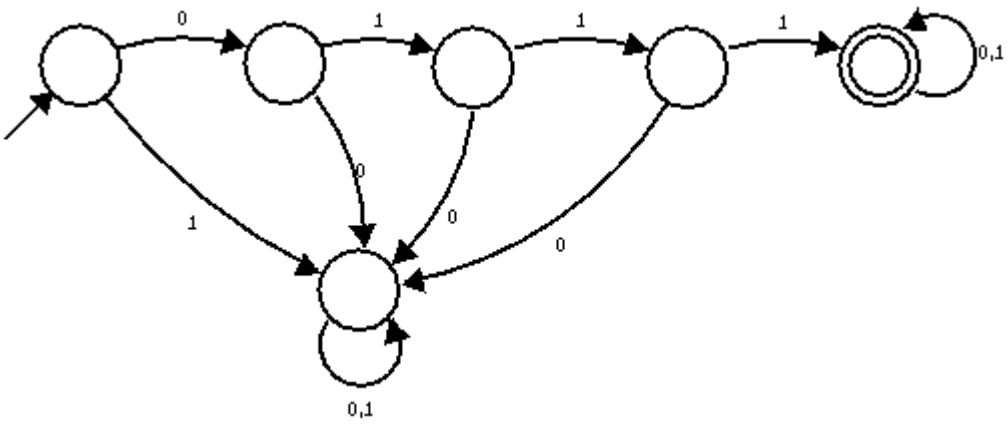
שאלה 2

האוטומטים המבוקשים מתואים באיורים למטה. ההוכחות בשאלה דומות להוכחה משאלה 1.

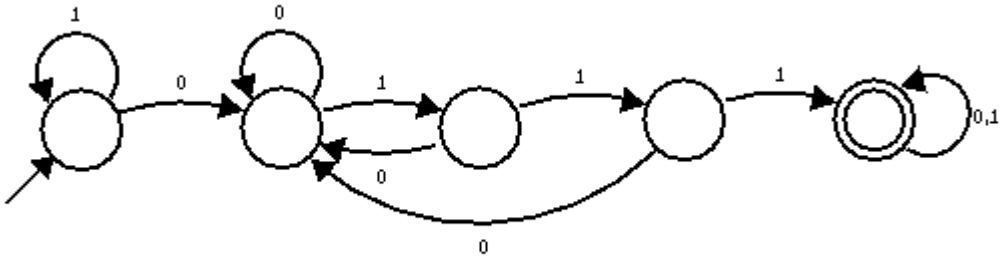
א.



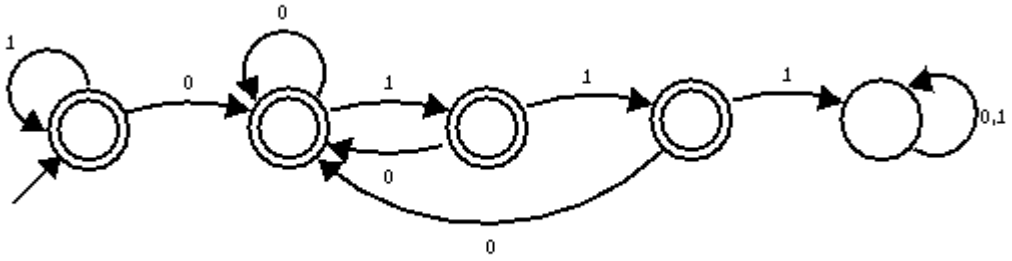
ב.



ג.

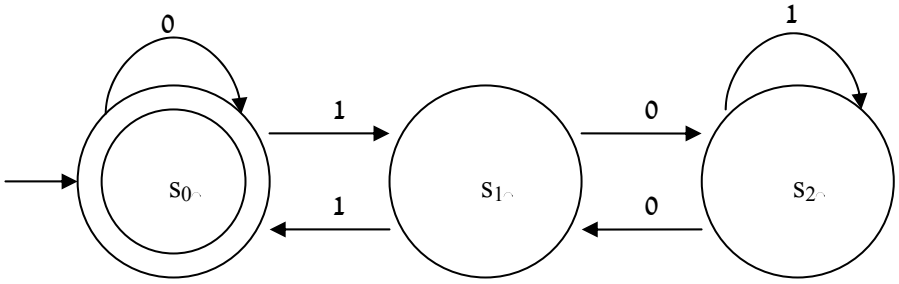


ד. זהו אוטומט השלילה של האוטומט שבנינו ב-ג'.



<u>שאלה 3</u>

DFA שמזהה מילים בינאריות שערכן מתחלק ב-3 :



טענה: הריצה של האוטומט הנ"ל על המילה $w = a_1 a_2 ... a_n$ מסתיימת במצב s_i המקיים $v(w) \equiv i (\mathrm{mod}\, 3)$.

הוכחה: באינדוקציה על n.

בסיס: n=0. אחרי קריאת ε האוטומט יהיה במצב S_0, ואכן $v(\varepsilon) = 0$.

נניח נכונות עבור n-1; לאחר שקרא את $a_1 ... a_{n-1}$, האוטומט נמצא במצב s_i, כך ש- $i \equiv v(a_1 a_2 ... a_{n-1}) (\mathrm{mod}\, 3)$. מההגדרה, $v(a_1 ... a_n) = 2v(a_1 ... a_{n-1}) + a_n$, ולכן

$$v(a_1 ... a_n) = 2v(a_1 ... a_{n-1}) + a_n = 2 \cdot i + v(a_n) (\mathrm{mod}\, 3).$$

בדיקת ששת המקרים האפשריים מראה שאכן הריצה מסתיימת במצב הרצוי.

<u>שאלה 4</u>

<u>E הוא יחס שקילות אמ"מ הוא יחס חלוקה.</u>

==> נניח $E \subseteq S \times S$ יחס שקילות, ונבנה את החלוקה המתאימה ל-E.

נגדיר $[x]_E = \{y \mid xEy\}$. לקבוצה $[x]_E$ נקרא (בשם המקורי) <u>מחלקת השקילות של x</u>. נתבונן בקבוצה $P = \{[x]_E \mid x \in S\}$, ונראה שהיא חלוקה של S העונה על הדרישות.

א. $S = \bigcup_{x \in S} [x]_E$: E רפלקסיבי, לכן xEx, לכן $x \in [x]_E \in P$.

ב. אם $[x]_E \neq [y]_E$, אז $[x]_E \cap [y]_E = \varnothing$: נניח בשלילה שקיים $z \in [x]_E, z \in [y]_E$, כלומר xEz, yEz. מסימטריה וטרנזטיביות, xEy.

נניח עכשיו $w \in [y]_E$. אז מההגדרה, yEw, ומטרנזיטיביות, xEz, ולכן מההגדרה, $w \in [x]_E$. לכן $[y]_E \subseteq [x]_E$. באופן דומה נוכיח $[y]_E \supseteq [x]_E$, וזאת סתירה.

ג. כאן יש שני כיוונים להראות. אם xEy, אז $x \in [x]_E$ (כמו שהראנו ב-א'), ו- $y \in [x]_E$ (מהגדרת $[x]_E$). וההיפך – אם $x, y \in [z]_E$, אז zEx, zEy, ואז עם רפלקסיביות וטרנזיטיביות נקבל xEy.

<== נניח E יחס חלוקה כפי שהוגדר בתרגיל. נראה ש-E יחס שקילות:

<u>רפלקסיביות</u>: עבור $x \in S$, מ-(a) קיימת S_i כך ש- $x \in S_i$, ואז מהכיוון השני של (c), xEx.

<u>סימטריה</u>: עבור $x, y \in S$ כך ש- xEy. אז לפי (c) (הכיוון הראשון) עבור $x, y \in S_i$ איזשהו i. לכן, לפי הכיוון השני של (c), yEx.

<u>טרנזיטיביות</u>: עבור $x, y, z \in S$, xEy, yEz. לפי (c) (הכיוון הראשון), קיימים i,j כך ש- $x, y \in S_i, y, z \in S_j$. לפי b, היות ש-y מופיע בשתי הקבוצות, $S_i = S_j$. לכן לפי (c) (הכיוון השני), xEz.

<u>E הוא יחס חלוקה אמ"מ E יחס פונקציה הפוכה.</u>

==> נניח E יחס חלוקה ונראה שהוא יחס פונקציה הפוכה.

נגדיר $f : S \to \{S_i\}_{i \in I}$, כאשר $\{S_i\}_{i \in I}$ החלוקה המושרית על-ידי E כפי שהוגדרה בתרגיל.

נשים לב ש- f מוגדרת היטב: מתכונה (a) של יחס החלוקה היא מלאה, ומתכונה (b) היא חד-ערכית.

לכל $x, y \in S$ xEy אמ"מ קיים $S' \in \{S_i\}_{i \in I}$ כך ש- $x, y \in S_i$ (מהכיוון הראשון של (c)) וזה אמ"מ $f(x) = f(y) = S_i$.

<== נניח E יחס פונקציה הפוכה ונראה שהוא יחס חלוקה.
E יחס פונקציה הפוכה, אז קיימת קבוצה T ופונקציה $f:S\to T$, כך ש- xEy אמ"מ $f(x)=f(y)$, לכל $x,y\in S$. נגדיר עבור $t\in T$, $f^{-1}(t)=\{x\mid x\in S, f(x)=t\}$, כלומר קבוצת המקורות של t לפי f.
נסתכל על $P=\left\{f^{-1}(t)\mid t\in T\right\}$, ונראה שזאת חלוקה העונה על התנאים.

א. $S=\bigcup_{t\in T} f^{-1}(t)$. נובע ממלאות הפונקציה. הגדרת הפונקציה דורשת שלכל $x\in S$, יהיה $t\in T$ כך ש- $f(s)=t$.

ב. אם $t_1\neq t_2$, אז $f^{-1}(t_1)\cap f^{-1}(t_2)=\varnothing$: נניח בשלילה ש- $x\in f^{-1}(t_1), x\in f^{-1}(t_2)$, אז $f(x)=t_1$ אבל גם $f(x)=t_2$, אז מחד-ערכיות הפונקציה, $t_1=t_2$ וזאת סתירה.

ג. אם xEy, אז $f(x)=f(y)$, נסמן $f(x)=t$, אז $x,y\in f^{-1}(t)\in P$. מצד שני, אם $x,y\in f^{-1}(t)$, אז מחד-ערכיות הפונקציה $f(x)=f(y)$ ולכן xEy.

<u>שאלה 5</u>

ראשית, נשים לב כי $\{0,1\}^*$ היא קבוצה בת-מנייה: ניתן לסדר את איברי הקבוצה (כל המילים הבינאריות) על פי האורך כאשר מילים באותו אורך מסודרות לקסיקוגרפית. קבוצת השפות מעל הא"ב $\{0,1\}$ הינה למעשה קבוצת החזקה של $\{0,1\}^*$. ישנה התאמה חח"ע ועל בין קבוצה זו לקבוצת הסדרות האינסופיות של 0 ו-1: לכל ת"ק של $\{0,1\}^*$, נתאים את הסדרה שבה 0 במקומות המקבילים (לקסיקוגרפית) למילים שלא שייכות לשפה, ו-1 במקומות המקבילים למילים השייכות לשפה. יתר על כן, קיימת העתקה חח"ע מהמספרים הממשיים בקטע $[0,1]$ לסדרות אינסופיות כנ"ל: לכל מספר נתאים את הייצוג הבינארי שלו (אחרי הנקודה). ראינו בכיתה כי קבוצת הממשיים בקטע אינה בת-מנייה, ומכאן הטענה נובעת.

בנוסף, קבוצת כל האוטומטים הסופיים מעל הא"ב $\{0,1\}$ היא בת-מניה: לכל מספר טבעי n יש מספר סופי של אוטומטים על n מצבים (כי יש מספר סופי של אפשרויות לבחור את (F,q_0,δ). אפשר למנות את כל האוטומטים על ידי כך שנמנה ראשית את האוטומטים עם מצב אחד, שני מצבים, וכן הלאה.

לכן, ניתן לסדר את כל ה DFA בסדרה $A_1,A_2,\ldots$. בסדרה זו מופיעים אוטומטים שיש להם אותה שפה, ניתן להביט בתת-סדרה של סדרה זו, בה מופיע אוטומט רק אם השפה שהוא מקבל שונה מהשפה שקיבל כל אחד מהאוטומטים שהופיעו לפניו בסדרה $A_1,A_2,\ldots$.
נביט עתה בסדרת השפות של שאוטומטים המופיעים בתת-הסדרה שבנינו.
מצד אחד, ברור שאף שפה לא מופיעה פעמיים (על פי בניית תת-הסדרה). מצד שני, ברור שכל שפה רגולרית מופיעה (שכן לכל שפה רגולרית יש DFA שמקבל אותה). לכן, קבוצת השפות הרגולריות בת מניה.
מכאן, חייבת להיות שפה לא-רגולרית מעל הא"ב $\{0,1\}$.

Computability - Exercise 2

All answers should be proved formally

Due March 19

1. For a language $L \subseteq \Sigma^*$, let

$$Pref(L) = \{x : \text{ there exists } y \text{ such that } xy \in L\},$$

and

$$Suff(L) = \{x : \text{ there exists } y \text{ such that } yx \in L\}.$$

Show that if L is regular then so are $Pref(L)$ and $Suff(L)$.

2. For a word $w = w_1 w_2 \cdots w_n$, the reverse of w, denoted w^R is the word w written in reverse order, i.e., $w_n \cdots w_2 w_1$. For a language $L \subseteq \Sigma^*$, let $L^R = \{w^R | w \in L\}$.
Show that if L is regular then so is L^R.

3. Consider the following finite language (over $\Sigma = \{0, 1\}$):

$$L_n = \{ww \mid w \in \{0,1\}^n\}.$$

   (a) Prove that every *nondeterministic* finite automaton for L_n must contain at least 2^n states.

   (b) Show a nondeterministic finite automaton for $\overline{L_n}$ with $O(n)$ states. Where $\overline{L_n}$ is the complement of L_n.

4. Draw an equivalent deterministic finite automaton for the following automaton (see Figure 1).

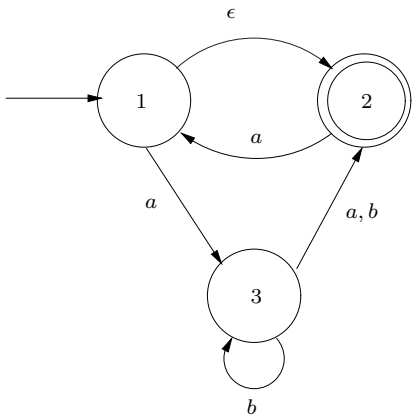


Figure 1: Determinize me!

If you use the determinization construction taught in class (with or without omitting unreachable states) there is no need to provide a proof.

5. (optional)

For a language L (over Σ), define the language $L_{\frac{1}{2}}$ (over Σ) as follows: $L_{\frac{1}{2}} = \{w : \exists y \text{ such that } |w| = |y| \text{ and } wy \in L\}$

Prove that if L is regular then so is $L_{\frac{1}{2}}$.

Computability - Solution of Exercise 2

1. Preliminary: The extended transition function $\delta^* : Q \times \Sigma^* \to Q$ is defined by induction on the length of its input word w:

   $$\delta^*(q, \epsilon) = q$$

   For $w = xa$, where $|x| = |w| - 1$ and $a \in \Sigma$,

   $$\delta^*(q, w) = \delta(\delta^*(q, x), a)$$

   (a) For a regular language L, the language $Pref(L)$ is regular. Let $\mathcal{A} = \langle Q, \Sigma, \delta, q_0, F \rangle$ be a DFA that accepts L. We define $\mathcal{A}_{pref} = \langle Q, \Sigma, \delta_{pref}, q_0, F' \rangle$ that accepts $Pref(L)$, where $F' = \{q \in Q | \exists w \in \Sigma^* \ s.t. \ \delta^*(q, w) \in F\}$.

   For $x \in Pref(L)$, there is a y such that $xy \in L$. The state $\delta^*(q_0, x)$ must be in F', since $\delta^*(q_0, xy) \in F$. Therefore, $x \in L(A_{pref})$.
   For $x \in L(A_{pref})$, $\delta^*(q_0, x) \in F'$, hence there exists a w such that $xw \in L$. Therefore, $x \in Pref(L)$.

   (b) Constructing an automaton is trivial, what follows is an alternative proof. For a regular language L, the language $Suff(L)$ is regular. It is easy to see that $[Pref(L^R)]^R = Suff(L)$.
   The left-hand-side is regular, since regular languages are closed under both the reverse operation (question 2), and under the prefix operation (the above item).

2. Let $A = \langle Q, \Sigma, Q_0, \delta, F \rangle$ be a DFA accepting L. We build an NFA $A' = \langle Q, \Sigma, F, \delta^{-1}, Q_0 \rangle$ accepting L^R as follows: the initial states of A' are the *accepting states* of A. The transition function of A' is δ^{-1} the inverse relation of δ defined as: $\delta^{-1}(s, \sigma) = \{r \in Q : \delta(r, \sigma) = s\}$. The accepting states of A' are the *initial states* of A.

   Let $w = w_1 \cdot w_2 \cdots w_n$ be a word in L. Look at the run of A on w. This is a sequence of states $r_0, r_1, \ldots, r_n$ where $r_0 \in Q_0$, $r_{i+1} = \delta(r_i, \sigma_{i+1})$, and

r_n is in F (why?). Therefore, $r_n, r_{n-1}, \ldots, r_0$ is computation of A' on input w^R.

On the other hand, if $r'_0, \ldots, r'_n$ is an accepting run of A' on input w, then $r'_n, r'_{n-1}, \ldots, r'_0$ is an accepting run of A on input w^R. Thus $L(A') = L^R$ and L^R is regular.

3. (a) Assume towards contradiction that there exists an NFA $A = \langle Q, \Sigma, Q_0, \delta, F \rangle$ such that its language $L(A)$ is $\{ww \mid w \in \{0,1\}^n\}$, and $|Q| < 2^n$.

   For each word $w \in \{0,1\}^n$, the word ww is in L, and is therefore accepted by some run $r^w_0, r^w_1, \ldots, r^w_n, r^w_{n+1}, \ldots, r^w_{2n}$ of A. Note that there are 2^n words in $\{0,1\}^n$. Therefore, since $|Q| < 2^n$, there must be two different words w and u in $\{0,1\}^n$ for which $r^w_n = r^u_n$. This means that $r^w_0, r^w_1, \ldots, r^w_n = r^u_n, r^u_{n+1}, \ldots, r^u_{2n}$ is an accepting run of A on wu. Note, however, that wu is not in the language of A, and we reach a contradiction.

   (b) The language $\overline{L_n}$ is the union of two cases: first, it might be that the word is not of length $2n$. Second, it might be that the word is of length $2n$ but is not of the type ww. In the second case, it is always the case that there exists two letters at distance n that are different (which is impossible if the word is of the form ww). We therefore choose nondeterministically between two $O(n)$ size automata. The first A_1 checks that a word is of length different then $2n$ and the second A_2 checks that there are two letters in distance n that are different. Note that A_2 will accept also words of length different then $2n$ but we do not care. The construction of A_1 and A_2 is easy and we leave it as an exercise.

4. See the deterministic automaton in Figure 4.

5. We construct an NFA for the language $L_{\frac{1}{2}}$. Since L is regular, there is a DFA $A = \langle Q, \Sigma, \delta, s_0, F \rangle$ that accepts it. Define the NFA $A' = \langle Q', \Sigma, \delta', s'_0, F' \rangle$, as follows:

   $Q' = Q \times Q \times Q$
   $Q_0 = \{(s_0, s, s) | s \in Q\}$
   $F' = \{(s, s, q) | q \in F, s \in Q\}$
   $\delta'((s, q, r), a) = \{(\delta(s, a), q, \delta(r, b)) | b \in \Sigma\}$

   Intuitively, the NFA A' runs in parallel on two tracks. The first track (which is represented by the first element of each state) simulates A from the starting state. The second track (which is represented by the third element of each state) simulates A from some state (which we nondeterministically guess).

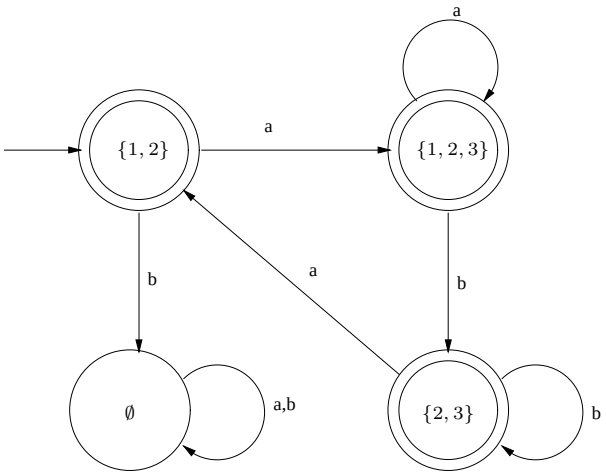


Figure 1: I'm deterministic!

The second element of each state is fixed throughout the run path and is used to remember the state from which we started the second track. We accept if and only if we have a computation path on the first track that starts in s_0 and on reading the input w it terminates in the state s, that was guessed at the start of the run. At the same time we have a computation path (that we choose nondeterministically) on the second track, that starts in s and terminates in an accepting state. We can then conclude that there exists y such that $wy \in L$.

Formally, we want to show that $L(A') = L_{\frac{1}{2}}$.
($\Rightarrow$) Assume that $w \in L(A')$. Then, there is an accepting run $(s_0, q, r_0), ..., (s_n, q, r_n)$ of A' on w. By the definition of A', we have that $r_0 = q$, $s_n = q$, and $r_n \in F$. Consider the second track of the run, by definition, $r_{i+1} = \delta(r_i, b_i)$ for some $b_i \in \Sigma$ ($0 \leq i < n$). Define $y = b_0 b_1 ... b_{n-1}$. Then, $wy \in L$. Indeed, by running A on wy we terminate in the state $r_n \in F$. Also, $|w| = |y|$ because for each step on the first track we make a step on the second. Therefore, $w \in L_{\frac{1}{2}}$.

($\Leftarrow$) Assume that $w \in L_{\frac{1}{2}}$. That is, there exists y such that $wy \in L$ and $|w| = |y|$. Denote the accepting run of A on wy by $r_0 r_1 \dots r_{|w|} r_{|w|+1} \dots r_{2|w|}$. Then the following sequence is an accepting run of A' on w:
$(r_0, r_{|w|}, r_{|w|}), (r_1, r_{|w|}, r_{|w|+1}) \dots (r_{|w|}, r_{|w|}, r_{2|w|})$.

Computability - Exercise 3

All answers should be proved formally

Due Monday, March 26

1. For each of the following languages over the alphabet $\{0,1\}$, write a regular expression for the language. No need to prove your answer (but make sure you are correct).
   (a) All words of odd length. (hint: is it easier to think of even?)
   (b) All words in which the number of 1's is strictly smaller then 5.
   (c) All words that do not contain neither 00 not 11 as a subwords.
2. Are the following languages regular? If ,in your proof, you choose to describe an automaton or a regular expression, there is no need to prove the correctness of the construction.

   All the languages are over $\Sigma = \{0,1\}$
   (a) $L = \{1^{2^n} \mid n \geq 0\}$
   (b) $L = \{0^m 1^n \mid 0 \leq m \leq n \leq 1000\}$
   (c) $L = \{w \mid$ The number of 01 substrings in w equals the number of 10 substrings in $w\}$
   (d) $L = \{ww \mid w \in \Sigma^*\}$
   (e) For a fixed natural $n \geq 0$, the language $L_n = \{ww \mid w \in \Sigma^n\}$
3. Define deterministic infinite automaton in the same way that DFA's are defined, with the only difference that the set of states can be infinite, and so is the set of accepting states. What languages are accepted by deterministic infinite automata?
4. In this question (taken from the exam in 2006) we define a new type of automata: universal automata. The definition of universal automaton is very similar to that of a nondeterministic automaton. The only

difference is that a word w is accepted by a universal automaton $\mathcal{A}$ iff *all* the runs of $\mathcal{A}$ on w are accepting (rather then if there exists a run that is accepting).

For example: look at the universal automaton $\mathcal{A}$ in Figure 1:

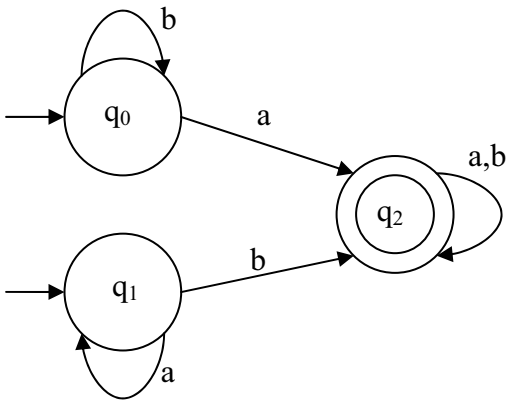


Figure 1: Example: $\mathcal{A}$ a universal automaton

Note that $bbb \notin L(\mathcal{A})$ because $q_0q_0q_0q_0$ is a non-accepting run of $\mathcal{A}$ on bbb.

(a) What is $L(A)$?

(b) Prove or refute: for every language $L \subseteq \Sigma^*$ it holds that L is regular iff L is accepted by some universal automaton

5. (Optional question)
Prove that a DFA with n states accepts an infinite language iff it accepts a word w such that $n \leq |w| \leq 2n$ (where $|w|$ is the length of w).

6. (Optional question)
Show that the language $L = \{a^ib^jc^k \mid i, j, k \geq 0 \text{ and if } i \geq 1 \text{ then } j = k\}$ satisfies all the conditions of the pumping lemma. Prove that L is not regular. Does this fact contradict the pumping lemma?

<u>חישוביות – פתרון תרגיל 3</u>

<u>שאלה 1</u>

א. $(0+1)(00+01+10+11)^*$

ב. $0^*(1+\varepsilon)0^*(1+\varepsilon)0^*(1+\varepsilon)0^*(1+\varepsilon)0^*$

ג. $(0+\varepsilon)(10)^* + (1+\varepsilon)(01)^*$

<u>שאלה 2</u>

א. $L=\left\{1^{2^n} \mid n\geq 0\right\}$ לא רגולרית. הוכחה באמצעות למת הניפוח: לכל p לוקחים $w=1^{2^p}\in L$, ומראים שאם פירוק מקיים את שני תנאי הלמה הראשונים, אז $2^p<\left|xyyz\right|<2^{p+1}$. הוכחה באמצעות Myhill-Nerode: לכל שתי מילים שונות $1^{2^n+1}, 1^{2^m+1}$ הסיפא 1^{2^n-1} מפרידה ביניהן, ולכן יש אינסוף מחלקות שקילות.

ב. $L=\left\{0^m1^n \mid 0\leq m\leq n\leq 1000\right\}$ סופית ולכן רגולרית.

ג. $L=\left\{w \mid \text{The number of 01 substrings in } w \text{ equals the number of 10 substrings in } w\right\}$

L רגולרית, לדוגמה עם הביטוי הרגולרי: $0(0+1)^*0+1(0+1)^*1+0+1+\varepsilon$

ד. $L=\left\{ww \mid w\in\Sigma^*\right\}$ לא רגולרית. הוכחה באמצעות למת הניפוח: לכל p לוקחים $w=0^p10^p1$, ומראים שאם פירוק מקיים את שני תנאי הלמה הראשונים, אז $xyyz=0^{p+m}10^p1\notin L$ עבור $m>0$. הוכחה באמצעות Myhill-Nerode: לכל $n\neq m$, $0^n1, 0^m1$ הן במחלקות שקילות שונות, כאשר 0^n1 סיפא מפרידה, ולכן יש אינסוף מחלקות שקילות.

ה. עבור מספר טבעי קבוע $n\geq 0$, השפה $L_n=\left\{ww \mid w\in\Sigma^n\right\}$ היא סופית, ולכן רגולרית.

<u>שאלה 3</u>

ההגדרה זהה להגדרה הסטנדרטית מלבד הדרישה שקבוצת המצבים סופית.

אוטומט דטרמיניסטי עם מספר אינסופי של מצבים יכול לקבל כל שפה $L\subseteq\Sigma^*$. בהנתן שפה L, האוטומט שיקבל אותה יהיה: $A=\left\langle\Sigma^*,\Sigma,\delta,\varepsilon,L\right\rangle$, כאשר Σ^* היא קבוצת המצבים, ε המצב ההתחלתי, L קבוצת המצבים המקבלים, ו- $\delta(w,a)=w.a$.

<u>שאלה 4</u>

א. כל המילים מעל{a,b} המכילות גם a וגם b.
הביטוי הרגולרי: $(a+b)^*a(a+b)^*b(a+b)^* + (a+b)^*b(a+b)^*a(a+b)^*$.

ב. נוכיח את שני הכיוונים:

i) אם L רגולרית אז קיים DFA שמזהה אותה, וכיון שכל DFA הוא גם אוטומט כולל, יש אוטומט כולל שמזהה אותה.

ii) בהינתן אוטומט כולל, ניתן לבנות לו DFA שקול בדרך דומה למה שראינו עבור NFA. נבנה את אוטומט החזקה, שבו כל מצב הוא קבוצת מצבים של האוטומט המקורי, המצב התחילי והמעברים מוגדרים כמו בבניה עבור NFA, והמצבים המקבלים הם כל הקבוצות שבהן **כל המצבים** הם מקבלים (בניגוד לבניה עבור NFA שבה קבוצה היא מקבלת אם **קיים** בה מצב מקבל).
לכן, אם שפה ניתנת לזיהוי על ידי אוטומט כולל אז היא רגולרית.

שאלה 5

DFA מקבל שפה אינסופית אמ״מ הוא מקבל מילה $n \le |w| \le 2n$.

שני הכיוונים קלים אם מבינים את הוכחת למת הניפוח.

אם מתקבלת מילה באורך גדול מ-n אז בריצה המקבלת על מילה זו יש מעגל. ניתן לנפח מעגל זה כרצוננו ולכן יש אינסוף מילים בשפה.

מצד שני, אם בשפה יש אינסוף מילים אז יש בה מילים ארוכות כרצוננו, ובפרט יש מילה באורך גדול מ-n. בריצה על כל מילה כזו יש מעגל, ובפרט מעגל פשוט, כך שאם המילה באורך גדול מ-2n ניתן להשמיט את המעגל מהריצה ולקבל ריצה מקבלת על מילה קצרה יותר. במעגל פשוט יש לכל היותר n מצבים, לכן אם המילה היתה באורך גדול מ-2n, לאחר ההשמטה היא עדיין באורך גדול מ-n. אם המילה הקצרה יותר עדיין באורך גדול מ-2n, ניתן לחזור על התהליך עד שתתקבל מילה באורך הרצוי.

שאלה 6

ראשית נראה כי השפה מקיימת את תנאי למת הניפוח, עם p=1.

תהי מילה בשפה מהצורה $a^ib^jc^k$, ובאורך לכל הפחות 1. יש שני מקרים:

(i) $i \ge 1$: נבחר $x=\varepsilon$, $y=a$, ו-z יהיה שאר המילה. מתקיים $|y|>0$, $|xy| \le 1=p$, וכן אם ״ננפח״ את מספר ה-a, המילה המתקבלת עדיין תהיה בשפה.

(ii) $i=0$: נבחר $x=\varepsilon$, y יהיה האות הראשונה, ו-z יהיה שאר המילה. ברור שכל התנאים מתקיימים.

מצד שני, L אינה רגולרית. עובדה זו נובעת ישירות ממשפט Myhill-Nerode: לכל $i \ne j$ טבעיים, המילים ab^i ו-ab^j נמצאות במחלקות שקילות שונות, ומכאן מספר מחלקות השקילות אינסופי. לא מדובר בסתירה ללמת הניפוח: למה זו נותנת תנאי הכרחי לכך ששפה כלשהי הינה רגולרית, אך זהו איננו תנאי מספיק.

Computability - Exercise 4

All answers should be proved formally

Due Monday, April 16

1. (a) Specify the Myhill-Nerode equivalence classes of the language $L_1 = \{0^i 1^j \mid i > j\}$. (You do not have to prove that these are the equivalence classes, but do be careful not to miss anything). Is L_1 regular?
   (b) Is $L_2 = \{w \in \{0,1\}^* \mid$ the number of 0's in w is greater then the number of 1's in $w\}$ regular? (remember to prove your answer).
   Hint: You may use the language 0^*1^*.

2. Describe the (Myhill-Nerode) equivalence classes of the language $L = (0+1)^*010(0+1)^*$ and draw a deterministic finite automaton (a.k.a. DFA) for L, based on these classes. (No proof required)

3. (based on a question from last year's midterm exam)
   Let $C = \{L_1, L_2, \ldots, L_n\}$ be a finite set of regular languages over an alphabet Σ. Suppose that for each one of the languages in the set, the number of Myhill-Nerode equivalence classes is k. For $m \geq 0$, let L^m denote the language

   $$L^m = \{w \in \Sigma^* : w \text{ belongs to exactly } m \text{ languages from } C\}$$

   Give a tight bound for the number of Myhill-Nerode equivalence classes of L^m.
   A tight bound is a function $f : \mathbb{N} \times \mathbb{N} \times \mathbb{N} \to \mathbb{N}$, such that for all n,k, and m, there is no set $C = \{L_1, \ldots, L_n\}$ for which the DFA for L^m needs more than $f(n, m, k)$ states (that is, f is an upper bound), and there exists a set $C = \{L_1, \ldots, L_n\}$ for which an automaton for L^m needs at least $f(n, k, m)$ states (that is, f is also a lower bound).
   Justify your answer (without a formal proof).

4. For each of the following languages over $\Sigma = \{0, 1\}$, write a context-free grammar with the **minimal** number of variables that generates the language (without further proof).
   (a) $\{w \mid w = w^R\}$ (w^R denotes the reverse of w).
   (b) $\{w \mid w \neq w^R\}$.
   (c) $\{w \mid$ the number of 0's in w equals to the number of 1's$\}$.

5. (optional)
   Let G be a context-free grammar in Chomsky normal form that contains k variables. Show that, if G generates some string using a derivation with at least 2^k steps, then $L(G)$ is infinite (that is, contains infinitely many words).

Computability - Exercise 4 - Solution

1. (a) The equivalence classes of L_1 are as follows.
   - For every $i \geq 0$, the set $\{0^i\}$ is an equivalence class.
   - For every $k > 0$, the set $\{0^i 1^j : j > 0, i - j = k\}$ is an equivalence class.
   - All other words, i.e., all the words not in L_1 except for ϵ, form an equivalence class.

   Since L_1 has infinitely many equivalence classes, it is not regular.

   (b) The language 0^*1^* is regular (in fact, it is given by regular expression). Recall that the class of regular languages is closed under intersection. Observe that the language L_1 is the intersection of 0^*1^* and L_2. Thus, if L_2 is regular then so is L_1, and we have reached a contradiction. Therefore, L_2 is not regular.

2. The language L has 4 equivalence classes:

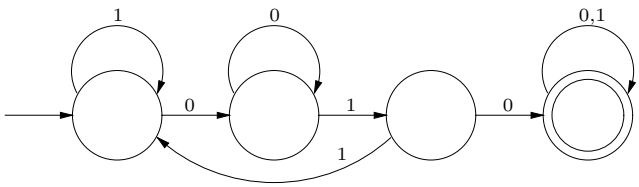


Figure 1: A DFA for L

   - Words that contain 010 as a subword.
   - Words that do not contain 010 as a subword, and their longest suffix that is also a prefix of 010 is of length 0 (e.g., words that end with 11).
   - Words that do not contain 010 as a subword, and their longest suffix that is also a prefix of 010 is of length 1 (e.g., words that end with 00).
   - Words that do not contain 010 as a subword, and their longest suffix that is also a prefix of 010 is of length 2 (e.g., words that end with 001).

3. The bound for the number of equivalence classes of L^m is k^n.
   The equivalence classes of L^m are the product of the equivalence classes of the languages in C. Note that the number does not depend on m. The parameter m just determines which classes are accepting.

   To prove the upper bound formally, construct the product automaton from the minimal DFA's for each language. Each state in the product is an n-tuple and the accepting states of the product automaton are the states that have exactly m accepting elements.

   The lower bound is a bit trickier for technical reasons. The problem is in ensuring that the various languages are "independent" in the following sense: we would like to ensure that knowing that a word $s \in \Sigma^*$ is in class c_i^1 of $\sim_{L_1}$ does not give any information on the class of s in $\sim_{L_2}$. To ensure such "independence", we work with an alphabet that is a Cartesian product of n alphabets, and let L_i care only about the i-th coordinate. In addition, to make sure the length of the word does not convey information, we pick L_i's in which for any (large enough) length of word j, there are words of length j both in L_i and outside L_i. For example, let $L'_1, \ldots, L'_n \subseteq \{0,1\}^*$ be languages over $\{0,1\}$ each with k equivalence classes such that for all sufficiently large j we have $L'_i \cap \{0,1\}^k \neq \emptyset$ and $L'_i \cap \{0,1\}^k \neq \{0,1\}^k$. We define L_i to be the language of words over $\Sigma = \{0,1\}^n$ in which for every word $x \in \Sigma^*$ we have $x \in L_i$ iff the projection of x on the i-th coordinate is in L'_i. It is not hard to see that the new languages $L_1, \ldots L_n$ are "independent" even if $L'_1, \ldots, L'_n$ were not. The formal proof is to define the equivalence classes of L^m as the Cartesian product of the equivalence classes of each language, and show separating words between each two of them. (Once the alphabet construction is understood, this is just easy technical writing.)

4. (a) $S \to 0S0|1S1|0|1|\epsilon$
   (b) $S \to 0S0|1S1|0A1|1A0$
   $A \to 0A|1A|\epsilon$
   (c) $S \to 0S1|1S0|SS|\epsilon$

5. Let G be in Chomsky normal form, such that it has k variables. Suppose that G generates the word w using 2^{k+1} derivation steps. The parse tree of w is a binary tree since the derivations are of the form $A \to BC$. As a binary tree with 2^{k+1} nodes, it must have a height of a least $k+1$. On the longest path there must be a variable that appears twice. The rest of the proof continues along the lines of the proof of the pumping lemma for CFLs.

Computability - Exercise 5

All answers should be proved formally

Due Wednesday, April 25

In this exercise, whenever you describe an automaton (either NFA or PDA) for a language, there is no need to prove the correctness of the construction. The same holds for the language of regular expressions or context-free grammars.

1. Describe a CFG for the language of regular expressions over $\Sigma = \{0, 1\}$. Assume that the expressions that are built from binary operators such as $+$ or $\cdot$ are parenthesized.

   For example, this language contains words such as:
   $(0 + (1 + \varepsilon)^*)$, $((0 \cdot 0) \cdot 1^*)$, and so on.

   Note that the language of the CFG is over a larger alphabet $\Sigma' = \{0, 1, \cdot, +, ^*, \emptyset, (,), \text{“}\varepsilon\text{”}\}$, and be careful not to mix “ε” (the symbol from Σ') with ε (the empty word).

2. Draw a pushdown automaton, and describe a context-free grammar for the language

   $$L = \{a^i b^j c^k : i < j \text{ or } j < k\}.$$

3. (a) Describe the language of the following grammar
   $S \to A1B$
   $A \to 00A|\epsilon$
   $B \to 000B|\epsilon$

   (b) Is the language regular?

4. Prove that if C is a context-free language and R is a regular language, then $C \cap R$ is context-free.

5. (optional)

   Prove that the following languages are context-free.

   (a) $\{x\#y \mid x, y \in \{0, 1\}^* \text{ and } x \neq y\}$

   (b) $\{xy \mid x, y \in \{0, 1\}^*, |x| = |y| \text{ and } x \neq y\}$

Computability - Solution 5

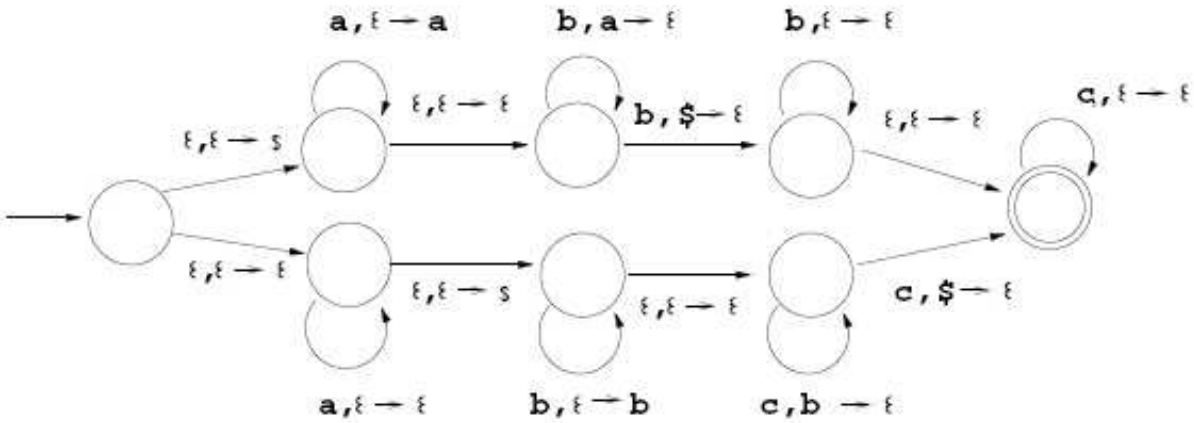
1. The CFG contains the following derivation rules (S is the initial variable).

$$S \to \emptyset \mid 0 \mid 1 \mid \text{“}\varepsilon'' \mid (S+S) \mid (SS) \mid (S^*)$$

2. Grammar:

   - $S \to XbC \mid AYc$
   - $X \to aXb \mid Xb \mid \varepsilon$
   - $C \to Cc \mid \varepsilon$
   - $A \to Aa \mid \varepsilon$
   - $Y \to bYC \mid Yc \mid \varepsilon$

   PDA:



3. (a) $L = \{0^{2i}10^{3j} \mid i, j \geq 0\}$
   (b) The language regular, $(00)^*1(000)^*$

4. Let $\mathcal{A}_1 = \langle Q_1, \Sigma, \Gamma_1, Q_0^1\delta_1, F_1\rangle$ be a PDA for C and let $\mathcal{A}_2 = \langle Q_2, \Sigma, Q_0^2, \delta, F_2\rangle$ be a DFA for R.

   As in the case of the intersection between two NFA's, we define the product automaton.

   (a) The state space is $Q_1 \times Q_2$ (i.e., states are ordered pairs $\langle q^1, q^2\rangle$).
   (b) The input alphabet is Σ.

(c) The tape alphabet is Γ_1.

(d) The initial states set is $Q_0^1 \times Q_0^2$.

(e) The transition relation changes the first coordinate of the state according to δ_1 and the second coordinate according to δ_2. The stack operations are set by δ_1 alone.

The ϵ transitions of δ_1 that do not read symbols, are performed changing the first coordinate, but leaving the second coordinate unchanged.

(f) The accepting states set is $F_1 \times F_2$.

5. (a) $L_1 = \{x\#y \mid x, y \in \{0,1\}^* \text{ and } x \neq y\}$

We describe informally a PDA recognizing L_1. There are two ways in which x might be different from y. First, it might be the case that their lengths differ (i.e., $|x| \neq |y|$). If $|x| = |y|$ then there must be an index i for which $x_i \neq y_i$. The PDA chooses nondeterministically to check either the first case or the second case.

To check the first case, $\mathcal{A}$ push to the stack until $\mathcal{A}$ sees a #. After the #, $\mathcal{A}$ starts popping. It is not hard to see that $|x| = |y|$ iff the word ends exactly when the stack is empty.

To check the second case, $\mathcal{A}$ start reading symbols, pushing them to the stack. At some stage $\mathcal{A}$ chooses nondeterministically to check some letter σ (this means guessing that $x_i = \sigma$). At that stage $\mathcal{A}$ moves to a state q_σ that "remembers" σ. Note that at this stage the stack is at depth $i-1$. Now, $\mathcal{A}$ continues to read symbols, this time without pushing them, until it sees a #. After seeing a #, $\mathcal{A}$ starts popping symbols, and when the stack is empty $\mathcal{A}$ is about to read y_i. Since $\mathcal{A}$ "remembers" σ it can easily compare y_i to σ and accept if $y_i \neq \sigma$.

(b) $L_2 = \{xy \mid x, y \in \{0,1\}^*, |x| = |y| \text{ and } x \neq y\}$ The grammar is:

i. $S \to AB \mid BA$

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

iii. $B \to 0B0 \mid 0B1 \mid 1B0 \mid 1B1 \mid 1$

Explanation: Choose $xy \in L_2$. Then there exists an i for which $x_i \neq y_i$. Assume w.l.o.g. that $x_i = 0$ (and $y_i = 1$). Denote the length of $|x|$ and $|y|$ by n. We have $xy = x_1 \dots x_{i-1} 0 x_{i+1} \dots x_n y_1 \dots y_{i-1} 1 y_{i+1} \dots y_n$. We can look at the same word in another way: First we have a word w_A, of length $(i-1)+1+(i-1) = 2i-1$, in which middle letter is 0. After that comes a word w_b of length $n-2i+1$ in which the middle letter is 1.

Computability - Exercise 6

Due June 11

1. We define a Turing Machine (TM for short) with 2-dimensional tape as a standard TM with the only difference that the tape has cells with names of the form (i, j) for every $i, j \geq 1$ ($i, j \in \mathbf{N}$). In each step the machine can move left, right, up or down, unless it is on the boundary of the tape. Thus, $\delta : Q \times \Gamma \to Q \times \Gamma \times \{L, R, U, D\}$. For example, if there is a transition rule: $\delta(q, a) = (q', b, U)$, and the reading head is currently in cell (i, j) reading a and the machine is in state q, then we change the a to b, we move to state q' and the reading head moves to cell $(i, j + 1)$. If we are in the bottom row (respectively left column) then moving down (respectively left) means that we stay in the same cell. The input is given in the bottom row.

   Define formally this model of TM and show that it is equivalent to standard TM's.

2. Define formally the model of pushdown automata with two stacks, and prove that it is equivalent to standard TM's.

3. Describe (in details, but don't define formally) the TM that accepts the language

   $$L = \{w \in \{\text{a,b,c}\}^* \mid w \text{ contains equal number of a's, b's, and c's}\}.$$

4. (optional) Prove that deterministic TM's (with one tape) that are not allowed to write on the area on which the input is written (but allowed to write on the area beyond), are equivalent to deterministic finite automata.

   Hint: use Myhill-Nerod theorem.

Computability - Solution of Exercise 6

1. A *two dimensional Turing machine* is 7 a tuple $M = \langle \Sigma, \Gamma, Q, q_0, \delta, q_{acc}, q_{rej} \rangle$, where Σ is an input alphabet, Γ is the tape alphabet, Q is a set of states, q_0 is an initial state, $\delta : Q \times \Gamma \to Q \times \Gamma \times \{L, R, U, D\}$ is a transition function, q_{acc} is an accepting state, and q_{rej} is a rejecting state. A configuration of the machine is a 4 tuple $\langle A, i, j, q \rangle$ where A is a finite square matrix of size $n \times n$ for some $n > 0$, the elements of A are taken from Γ. The indices $i, j \leq n$ stand for the place of the reading head, and q stands for the state of the machine. Intuitively, the machine works (potentially) on the first quadrant and all cells that do not appear in A are presumed to contain blank.

   The initial configuration of M with input $w = w_0 \dots w_n$ is $\langle A_0^w, 0, 0, q_0 \rangle$ where A_0^w is a matrix which is all blank but the cells $A_{0,0}, A_{0,1}, \dots, A_{0,n}$, which contain w. Let $C = \langle A, i, j, q \rangle$ be a configuration for which $\delta(q, A_{i,j}) = (q', \sigma', x)$. The successor configuration of C is $C' = \langle A', i', j', q' \rangle$, where A' is a matrix in which all the cells except perhaps $A_{i,j}$ have the same content as the corresponding cell in A. If the matrices are not of the same size then cells that appear in one of the matrices but not the other contain blank. The cell $A_{i,j}$ contains σ. Finally, i' and j' depend on x. If $x = L$, then $i' = i$, and $j' = j - 1$ unless $j = 0$, in which case $j' = 0$. If $x = R$ then $i' = i$, and $j' = j + 1$. If $x = D$, then $i' = i - 1$ unless $i = 0$ in which case $i' = 0$, and $j' = j$. If $x = U$, then $i' = i$ and $j' = j + 1$. An accepting computation of M on $w \in \Sigma^*$ is a sequence of successive configurations where the first configuration is an initial configuration of M on w, and the state in the last configuration is q_{acc}. A rejecting computation is defined analogously with the last state being q_{rej}.

   It is clear that a regular Turing machine is also a two dimensional Truing machine. Therefore, to prove equivalence it is enough to show that a regular Turing machine can simulate a two dimensional Turing machine. We show that a k-tape Turing machine can simulate a two dimensional Turing machine. We saw in class that a 1-tape Turing machine can simulate a k-tape Turing machine.

We represent a configuration of a two dimensional Turing machine in the following way: a special tape holds the matrix A where the rows are written one after the other separated by a special symbol $\#$. Thus, the row 0 is written first, then a $\#$ symbol is written, then the row 1 etc. To represent blank in the matrix A a special symbol is used, *not* the blank symbol of the k-tape machine.

A second tape holds the numbers i and j (in binary representation) separated by $\#$. In addition, the state of the simulated machine is written on a third tape. We shall use some auxiliary tapes as needed.

It is easy to see that given an input $w_0 \dots w_n$ (given according to the input conventions of a regular Turing machine) one can write the representation of an initial configuration of a two dimensional Turing machine. This involves writing 0,0 on head location tape, writing q_0 on the state tape, and writing the representation of the initial matrix on the matrix tape. Writing the initial matrix involves counting the length of the input, copying the input and then filling as many rows as needed by blanks (separating rows by $\#$). All easily done by a k-tape Turing machine.

To move from the representation of one configuration to the representation of the successor configuration, the k-tape machine has to locate the representation of the i^{th} row and the j^{th} cell in it, and then change it's content and the contents of the head location and state tapes according to the two dimensional machine transition. The only subtlety is to note that if the head location is changed, the representation of the matrix may no longer be large enough to contain it. If this happens, a blank should be added to each row and a new row should be added as well.

2. For an alphabet Δ we denote by Δ_ϵ the set $\Delta \cup \{\epsilon\}$. A 2 Stacks Push Down Automata (2SPDA in short) is a 6-tuple $A = \langle Q, \Sigma, \Gamma, \delta, Q_0, F \rangle$, where Q is a set of states, Σ is an input alphabet, Γ is a stack alphabet, $\delta : Q \times \Sigma_\epsilon \times \Gamma_\epsilon \times \Gamma_\epsilon \to 2^{Q \times \Gamma_\epsilon \times \Gamma_\epsilon}$ is a transition relation, Q_0 is a set of initial states, and F is a set of accepting states.

   A configuration of a 2SPDA is an ordered triple $(q, s_1, s_2) \in Q \times \Gamma^* \times \Gamma^*$. The set of initial configurations is $Q_0 \times \{\epsilon\} \times \{\epsilon\}$. For a configuration $C = (q, w_1, w_2)$ and $\sigma \in \Sigma_\epsilon$ we say that a configuration $C' = (q', w_1', w_2')$ is a σ successor of C if there exists $x_1, x_2 \in \Gamma_\epsilon$ and $(q', x_1', x_2') \in \delta(q, \sigma, x_1, x_2)$ such that: there exists $v_1, v_2 \in \Gamma^*$ for which $w_1 = x_1 v_1$ and $w_2 = x_2 v_2$ while $w_1' = x_1' v_1$ and $w_2' = x_2' y_2$.

   A run of a 2SPDA on a word $w_1 \dots w_n \in \Sigma^n$ is a sequence of configurations $r_0 \dots r_n$ where r_0 is an initial configuration, and for all $i \in \{1, \dots n\}$ it

holds that r_i is a w_i successor of r_{i-1}. A run of a 2SPDA is accepting if the state of the last configuration is in F. A 2SPDA accepts a word $w \in \Sigma^*$ if there exists an accepting run of it on w.

As for the equivalence, it is easy to see that a 2SPDA can be simulated by a Turing with 2 tapes, (one for each stack). We saw in class that such a Turing machine can be simulated by a 1-tape Turing machine.

We are left to show that a Turing machine can be simulated by a 2SPDA. A configuration of a Turing machine can be characterized by a the state, and two strings: the left string is what is to the left of the head, while the right string is the symbol the head is on, and the string to it's right (we ignore the infinite suffix of blanks). Such a configuration will be represented in the 2SPDA in the following way: The left string will be kept in one stack (the left stack), while the right string will be kept in the other stack (the right stack). The state of the Turing machine will be "kept" in the state of the 2SPDA (this can be done since there are only finitely many states). We now have to show that the 2SPDA can get to the initial configuration and that it can get from one configuration to its successor.

For the initial configuration, the 2SPDA reads the input while pushing every symbol read to the left stack. Afterwards, using ϵ transitions it pops the contents of the left stack and push it to the right stack. (This maneuver is needed to have the string lie in the right stack in the correct order.) Simulating one transition of the Turing machine is easy as the movement of the head can be simulated by popping from one stack and pushing to the other.

3. Intuitively, the machine "erases" (by replacing with a special symbol) one a, one b, and one c iteratively, until nothing is left.

   Denote our machine by M. We assume that the left end of the tape is marked by a special sign \$ (we saw in class how to do that). The alphabet $\Gamma \setminus \Sigma$ has another special symbol $\#$ that we will use later (and a blank of course). The initial state of the machine q_a^l goes left until the left end of the tape. The it moves to q_a^r that searches for an a. Intuitively q_a^r will go right until it sees an a and rewrite it with an $\#$. However, we want to record whether or not we saw a b or a c while searching for an a. Therefore if q_a^r sees a b or a c it turns into q_a^{r2}. The state q_a^{r2} goes right until it reaches an a and if so, replaces it with a $\#$. If q_a^r reaches blank (i.e. the right end) then M accepts. If q_a^{r2} reaches a blank, than M rejects.

   If either q_a^r or q_a^{r2} finds an a and replaces it with an $\#$, then it turns into q_b^l. The state q_b^l goes left until it reaches the left end, and turns into q_b^r. The state q_b^r goes right in search of a b. If q_b^r reaches a b it replaces it with a $\#$ and

turns into q_c^l. Otherwise, if q_b^r reaches a blank then M rejects. The state q_c^l behaves as q_b^l except that when it reaches the left end it turns into q_c^r. The state q_c^r behaves as q_b^r except that it searches for a c (rather than a b), and when it finds one it turns into q_a^l (rather then q_c^l).

4. A Turing machine can simulate a finite automaton without writing at all. Therefore it is left to show that such a Turing machine accepts a regular language, and can be simulated by a finite automaton.

   To prove the language is regular we will use the Myhill-Nerode theorem, and show there are only finitely many equivalence classes to $\equiv_L$.

   Let $s \in \Sigma^*$ be a string. When the machine starts to work on an input that begins with s, there are three possibilities: First it might be that the machine will never read a symbol cell of s since it will enter an accepting state. Second, it might be that the machine will never read a symbol outside of s because of some infinite loop. Third it might be that the string will read a symbol ouside of s. We define a function $f : \Sigma^* \to Q \cup \{\perp\}$ in the following way: in the first case $f(s) = q_{accept}$. In the second case $f(s) = \perp$. In the third case set the value of f to be the state in which the machine is in when it first reads a symbol outside of s.

   In a similar fashion, suppose the machine enters s from the left, in state q. There are three possibilities: the machine might accept whithin s, it might enter a loop whithin s, or it might leave s in state q'. For each $q \in Q$ we define a function $g_q : \Sigma^* \to Q \cup \{\perp\}$, in the first case set $g_q(s) = q_{accept}$. In the second case set $g_q(s) = \perp$. In the third case set $g_q(s) = q'$. If for two string $s_1, s_2 \in \Sigma^*$ it holds that both $f(s_1) = f(s_2)$ and for all $q \in Q$ it holds that $g_q(s_1) = g_q(s_2)$, then it must be that these strings are right invariant in the Myhill-Nerode sense.

   Since there are only finitely many posibilities to choose values for each of these $|Q| + 1$ functions, it follows that there are only finitely many Myhill-Nerode equivalence classes, so the language it regular.

Computability - Exercise 7

All answers should be proved formally

Due Monday, June 18

1. (a) Show that the class of decidable languages (denoted R) is closed under the operations: union, concatenation and complementation.
   (b) Show that the class of recognizable languages (denoted RE) is closed under the operations: union, intersection and Kleene star (*).
2. Show that for every enumerator E (including those which may print a few times the same word), there exists an enumerator E' that prints each word only once, such that $L(E) = L(E')$.
3. Show that a language is decidable by a TM if and only if there is an enumerator that enumerates it in the lexicographic order (where short words come before longer words, and within the same length, words are sorted according to the lexicographic order).
4. Let L_1 be a language. Prove that $L_1 \in RE$ iff there exists $L_2 \in R$ such that $L_1 = \{x : \exists y \text{ such that } < x, y > \in L_2\}$.
5. Show that for every infinite (that is, contains infinitely many words) language $L \in RE$ there exists an infinite language $L' \in R$ such that $L' \subseteq L$.
6. We define a new class of languages $EX7$ to be the following class: a language $L \subseteq \Sigma^*$ is in $EX7$, if there exists a deterministic TM M such that for every $x \in \Sigma^*$: if $x \in L$ then M never stops running on input x. If, on the other hand, $x \notin L$ then M running on input x rejects in $e^{|x|^7}$ steps or less. Which of the following claims is true (prove your answer even if the claim is false).
   (a) $EX7 \subseteq coRE \setminus RE$
   (b) $EX7 \subseteq RE \setminus R$
   (c) $EX7 \subseteq R$.

Computability - Exercise 7 - Solution

1. (a) **Union:** For two languages $L_1, L_2 \in R$, let M_1 and M_2 be the TM's that decide L_1 and L_2, respectively. We describe a TM M that decides the language $L = L_1 \cup L_2$. M is a machine with two tapes. On an input x, M first copies x to the second tape. It then simulates the run of M_1 on x on the first tape and the run of M_2 on x on the second tape, one step at a time. If M_1 or M_2 (or both) accept, M accepts. If both reject, M rejects. Since both M_1 and M_2 are deciders, eventually they both halt, thus M is a decider too, and its language is clearly $L_1 \cup L_2$.

   Remark: It is not necessary here to simulate one step at a time, but it is necessary to do so in the case where $L_1, L_2 \in RE$, as follows in part (b).

   Concatenation: Given languages $L_1, L_2 \in R$ and their corresponding (deciding) TM's M_1 and M_2, we describe a machine M that decides the concatenation of L_1 and L_2. On an input x of size n, for each one of the $n+1$ possible ways to split x to y, z such that $x = y \cdot z$ (i.e., $\varepsilon \cdot x, x_1 \cdot x_2 \dots x_n, \dots, x_1 \dots x_{n-1} \cdot x_n, x \cdot \varepsilon$) in its turn, M simulates M_1 on y and if it accepts, M simulates M_2 on z. If both M_1 and M_2 accepts, M accepts. Otherwise, M tries the next possible way to split x. If for all the possible ways M_1 rejected y or M_2 rejected z, M rejects. Clearly, $L(M) = L_1 \cdot L_2$. Moreover, since M_1 and M_2 are deciders, each one of the simulations eventually halts, and since M carries out at most $2(n+1)$ simulations, it eventually halts, thus M is a decider.

   Complement: Given a language L and a TM M that decides it, Define $\overline{M}$ the same as M, with the only difference, that its accepting state is the rejecting state of M, and its rejecting state is the accepting state of M. Since M is a decider, $\overline{M}$ is clearly a decider too. In addition, it is easy to see that $L(\overline{M}) = \overline{L}$.

   (b) **Union:** The construction is similar to the construction in the case of union in the previous section. M accepts when either M_1 or M_2 accept. If both reject, M rejects, and if none of the above happens (which is possible since M_1 and M_2 are not deciders) then M runs to infinity. For $x \in L$ we know that at least one of the machines M_1 and M_2 eventually accepts, since $L_1, L_2 \in RE$, and x is in one of these languages. Thus, M recognizes $L_1 \cup L_2$.

   Intersection: The construction is the same as for the union, except that M accepts only if both M_1 and M_2 accept, and if at least one of them rejects, M rejects. In any other case M loops forever. For $x \in L$, both M_1 and M_2 eventually accept, thus M recognizes $L_1 \cap L_2$.

   Star: Given a language L and a TM M that recognizes L, we describe a TM M' that recognizes L^*. The construction is similar to the construction for concatenation in the previous section. Here we try all the possible ways to split the input to substrings and we simulate the machine M on each substring (for i steps, $i = 1, 2, 3, \dots$). More formally, given an input x, for $i = 1, 2, 3, \dots$:

- For all possible ways to split x to substrings $y_1, \ldots, y_k$, run M on each of the substrings for i steps.
- If for some split, M accepted all the substrings $y_1, \ldots, y_k$, accept.
- If for all the ways to split x, M rejected at least one substring, reject.
- Move on to the next iteration.

If $x \in L^*$ then there exists a way to split x to substrings $y_1^*, \ldots, y_k^*$ such that $y_j^* \in L$ for all $1 \leq j \leq k$. Thus, for every y_j^* there is a number n_j such that M accepts y_j^* after n_j steps of computation. Let $m = \max_{1 \leq j \leq k} n_j$. Since the number of ways to split x is bounded, and in every iteration M' runs M for i steps on every substring, every iteration eventually ends. When M' gets to the m'th iteration in its run on x, for the split $y_1^*, \ldots, y_k^*$, M would accept all the substrings within m steps, therefore M' recognizes L^*.

2. Let E be an enumerator. We modify E as follows. Before writing a word w to the output tape, compare w to all the words previously written to the output tape. if w was already written, do not write it again. In this way we obtain an enumerator E' that prints every word once, and $L(E) = L(E')$

3. **Claim:** $L \in R$ iff there exists an enumerator that enumerates L in a lexicographic order (where short strings come before long ones).
($\Rightarrow$) Let M be a TM that decides L. Define the enumerator E as follows. E goes through all the strings in Σ^* in the order described above, and for each such string it runs M on it. If M accepts, E prints the string and goes to the next one. Otherwise, it goes to the next string without printing. Since M is a decider, M eventually halts on every input, thus E eventually simulates M on every input. Clearly, E enumerates L in the required order.
($\Leftarrow$) We distinguish between two cases. First, it might be that the language is finite. If this is the case, then the language is trivially decidable and there is nothing to prove. We prove the result for the case the language is infinite. Let E be the enumerator that enumerates L in lexicographic order, where short strings are printed before longer ones. Since the language is infinite, for every word $w \in \Sigma^*$ if we run E for long enough, it will print a word which is bigger then w in the lexicographic order.

   The TM M works as follows: On input x it runs E either until it prints x or until it prints a string bigger then x in the lexicographic order. If E prints x then M accepts. If, on the other hand, E prints a word bigger then x without printing x, then M rejects.

4. ($\Rightarrow$) Let M_1 be a TM that recognizes L_1.

   Define $L_2 = \{< x, y >: M_1 \text{ accepts } x \text{ within } y \text{ steps}\}$.

   We show that $L_2 \in R$ by describing a TM M that decides L_2. M_2 acts as follows :

   - Checks whether the input is of the form $< x, y >$. Otherwise, rejects.
   - Simulates M_1 on x for y steps. if M_1 accepted, accepts. Otherwise, rejects.

   It is easy to see that M_2 decides L_2.

   The above implies that $L_1 = \{x : \exists y \text{ such that } < x, y > \in L_2\}$. This is because if $x \in L_1$, there exists y such that M_1 accepts x after y steps, therefore $< x, y > \in L_2$. On the other

hand if $x \notin L_1$, there is no y such that M_1 accepts x after y steps, so there is no y such that $< x, y > \in L_2$.

($\Leftarrow$) Let M_2 be a TM that decides L_2. We describe a TM M_1 that recognizes L_1. Given an input x, M_1 acts as follows.

For all $w \in \Sigma^*$, in lexicographic order :

- simulate M_2 on $< x, w >$.
- If M_2 accepts, accept. Otherwise, move on to the next w.

If there exists y such that $< x, y > \in L_2$, then, since M_2 is a decider, every simulation eventually ends, so eventually M_1 will simulate M_2 on $< x, y >$ and will accpet. On the other hand, if there is no y such that $< x, y > \in L_2$ then M_1 will never halt on x.

5. Let $L \in RE$, and let E be an enumerator that enumerates L.

   Define $L' = \{w \in L : E \text{ does not print any word larger than } w \text{ before it prints } w\}$.

   $L' \subseteq L$, because L' contains only words in L.

   L' is infinite. Assume by a way of contradiction that L' is finite. Let w be the largest word in L'. It follows that E does not print any word largest than w after it printed w. This implies that $L(E)$ is finite, since there is only a finite number of words that are not larger than w, and this contradicts our assumption, that L is infinite.

   $L' \in R$. We describe an algorithm that decides L'. Given an input x:

   Simulate E. For each word w printed by E:

   - If $w = x$, accept.
   - If $|w| > |x|$, reject.
   - Otherwise, continue.

   If $x \in L'$, then E does not print any word larger then x before x. Thus, x is one of the first $2^{|x|+1}$ words printed by E. When the algorithm gets to x, it will accept. If $x \notin L'$, then since L is infinite, E eventually prints a word z larger then x. When the algorithm gets to z, it will reject.

6. We prove that if L is in $EX7$ then it is decidable (i.e., $L \in R$). This clearly implies that (a) and (b) are false but (c) is true.

   For $L \in EX7$ there exists a TM M as in the definition of $EX7$. We construct another TM M' which operates as follows: on input x, it runs M on x for $e^{|x|^7}$ steps. If M rejects (i.e. $x \notin L$) then M' rejects as well. Otherwise, M' accepts. Since we are assured that either M will reject within $e^{|x|^7}$ steps or it will run forever, we know that M' accepts exactly the words in L. Note that M' always stops and therefore decides L.

Computability - Exercise 8

All answers should be proved formally

Due Monday, June 25

1. For each of the following languages decide whether it is in R, in RE but not in R, or not in RE.
   (a) $L_1 = \{\langle M \rangle \mid M$ is a TM that passes the 100^{th} position in the tape during its run on the empty input$\}$.
   (b) $L_2 = \{\langle M \rangle \mid M$ is a TM and there exists an input such that M halts on it in not more than 1000 steps $\}$.
   (c) $L_3 = \{\langle M \rangle \mid M$ is a TM and $L(M)$ is context free$\}$.
   (d) $L_4 = \{\langle M \rangle \mid M$ is a TM and M stops on any input$\}$.

2. For each of the following languages state (and prove) whether it is in R, $RE \setminus R$, $coRE \setminus RE$, or none of them. Recall that $L \in coRE$ if $\overline{L} \in RE$.
   (a) $ALL_{TM} = \{\langle M \rangle \mid L(M) = \Sigma^*\}$
   (b) $L = \{\langle M_1, M_2 \rangle : L(M_1) \cap \overline{L(M_2)} = \emptyset\}$.
   (c) $L = \{\langle M \rangle : M$ accepts some word after more than 100 steps of computation$\}$.
   (d) $EQ_{CFG} = \{\langle G, H \rangle : G$ and H are CFGs and $L(G) = L(H)\}$.

3. Let A, B and C be languages over Σ. Prove that
   (a) If A reduces to B then the complement of A reduces to the complement of B.
   (b) A_{TM} is not mapping reducible to E_{TM}.
   Recall that $E_{TM} = \{\langle M \rangle \mid M$ is a TM and $L(M) = \emptyset\}$.

4. (a) Prove that for every two languages $L_1, L_2 \in R$ which are not Σ^* or $\emptyset$, it holds that $L_1 \leq_m L_2$.
   (b) Let L_1, L_2 be two languages such that $L_1 \leq_m L_2$ and L_2 is regular. Does this imply that L_1 is a regular language? Remember to prove your answer.

5. **(optional)** Let $A_0, A_1, A_2, \ldots$ be an infinite sequence of languages over alphabet Σ, such that for every $k \geq 1$, there is a mapping reduction, f_k, from A_k to A_{k-1}. That is, $A_k \leq A_{k-1}$.
   Let us denote: $B_r = \bigcup_{k=0}^{r} A_k$ and $B_\infty = \bigcup_{k=0}^{\infty} A_k$.
   It is given that $A_0 \in R$.
   For each one of the following languages, state the smallest class of languages that contains it from: R, RE, or not in any of these classes.
   That is, if you claim that the language is in some class, prove it. If you claim it is not in a class, give an example for a sequence $A_0, A_1, A_2, \ldots$ that shows it.
   (a) A_k.
   (b) B_r.
   (c) B_∞.

Computability - Exercise 8 - Solution

1. (a) $L_1 \in R$.
   We describe a TM T with two tapes that decides L_1: Given $\langle M \rangle$, T simulates M's run on the empty input. The second tape is used for recording the configurations of T during its run. At each step, if M passes the 100'th position, T accepts. Otherwise, T checks whether this configuration was previously recorded. If so, T rejects; else, it records the current configuration of M on the second tape, and goes on to the next step.

   Notice that T rejects if and only if M's run on the empty input has entered an infinite loop using only the first 100 positions on the tape, that is, returned to a configuration that was recorded before. Moreover, since the number of configurations which use only the first 100 positions on M's tape is bounded by $100 \cdot |Q| \cdot |\Gamma|^{100}$, T runs only a finite number of steps on any input.

   (b) $L_2 \in R$.
   Observe that in order to determine whether there exists a word on which a TM halts after at most 1000 steps, it is sufficient to look at words of length at most 1000, since M cannot pass the 1000'th position on the tape within 1000 steps.

   Using this understanding, we describe a TM T which decides L_2: Given $\langle M \rangle$, T simulates 1000 steps of M's run on every possible word of length at most 1000. T accepts if and only if M halts on one of these words.

   (c) $L_3 \notin RE$.
   We prove this by showing a reduction from $\overline{A_{TM}}$ (this is sufficient since $\overline{A_{TM}} \notin RE$). Given $\langle M, w \rangle$, the reduction outputs $\langle T \rangle$, where T is the following turing machine: Given an input x, T simulates M on w, and accepts if and only if M accepts w and x is of the form $a^n b^n c^n$.

   If $\langle M, w \rangle \in \overline{A_{TM}}$, then M does not accept w, and $L(T) = \emptyset$ which is a CFL, so $\langle T \rangle \in L_3$. If $\langle M, w \rangle \notin \overline{A_{TM}}$, then M accepts w, so T accepts exactly the words of the form $a^n b^n c^n$, i.e., $L(T) = \{a^n b^n c^n : \ n \in \mathbb{N}\}$, which is not a CFL; thus $\langle T \rangle \notin L_3$.

   (d) $L_4 \notin RE$.
   We prove this by showing a reduction from $\overline{A_{TM}}$ (this is sufficient since $\overline{A_{TM}} \notin RE$). Given $\langle M, w \rangle$, the reduction outputs $\langle T \rangle$, where T is the following turing machine: Given an input x, T simulates $|x|$ steps of M's run on w. If M accepted, T enters an infinite loop. Otherwise, T accepts.

   If $\langle M, w \rangle \in \overline{A_{TM}}$, then regardless of x, M never accepts w in $|x|$ steps, so T always halts and accepts; thus $\langle T \rangle \in L_4$. If $\langle M, w \rangle \notin \overline{A_{TM}}$, then M accepts w after k steps, so T's run doesn't halt on any x with $|x| \geq k$; thus $\langle T \rangle \notin L_4$.

2. (a) $ALL_{TM} = \{< M > \ | \ L(M) = \Sigma^*\}$. $ALL_{TM} \notin RE \cup coRE$.

We show a reduction both from A_{TM} and from $\overline{A}_{TM}$.

The reduction from A_{TM}: On an input $(\langle M\rangle, w)$ to A_{TM}, the output of the reduction will be the machine M_w, that on (any) input x, runs M on the string w, and accepts if M accepts w. If M accepts w then M_w accepts every input. Otherwise it doesn't accept any input.

The reduction from $\overline{A}_{TM}$: On an input $(\langle M\rangle, w)$ to $\overline{A}_{TM}$, the output of the reduction will be the machine M_w that on input x runs M on w for $|x|$ steps. If within this time M doesn't accept w then M_w accepts x, otherwise it rejects. If M doesn't accept w, then for every x, M_w will not see M accepting w within $|x|$ steps (because it just never happens) and it will accept x. Therefore $L(M_w) = \Sigma^*$. Otherwise, M accepts x after some k steps. Then by definition M_w will see M accepting x for every x such that $|x| \geq k$, and M_w will reject such x's. It follows that $L(M_w) \neq \Sigma^*$.

(b) $L = \{< M_1 >, < M_2 > : \ L(M_1) \cap \overline{L(M_2)} = \emptyset\}$. L is not in $RE \cup coRE$. We proved that ALL_{TM} is not in $RE \cup co-RE$. Here we prove that L is not in $RE \cup co-RE$ by reduction from ALL_{TM} to L. Take M_2 to be the input for ALL_{TM}, and M_1 as a machine that accepts all inputs. $L(M_1) \cap \overline{L(M_2)} = \Sigma^* \cap \overline{L(M_2)} = \overline{L(M_2)} = \emptyset$ iff $M_2 \in ALL_{TM}$.

(c) $L = \{< M > : \ M$ accepts some word with more than 100 steps of computation$\}$. $L \in RE \setminus R$

Let $w_1, w_2, \ldots$ be an enumeration of all the words in Σ^*. To see that L is in RE we describe a recognizing algorithm: For all $i \geq 0$ run M for i steps on the words $w_1, \ldots, w_i$. If one of these words is accepted by within i steps, and is accepted in more then 100 steps of computation, then $M \in L$. Otherwise move to the next i. Clearly if some word w_n is accepted by M, then we will eventually accept it. On the other hand, if no word is accepted by more then 100 steps then M will not be accepted (our procedure will run forever).

To see that L is not in R we show a reduction from A_{TM}. On an input $(\langle M\rangle, w)$ to A_{TM}, the output of the reduction will be the machine M_w, that on input x, enters a loop of 101 iterations, and then runs M on w and answers as M does. If M accepts w then M_w will accept every string in more than 100 steps. Otherwise it will not accept any string.

(d) $EQ_{CFG} = \{\langle G, H\rangle \mid G$ and H are CFGs and $L(G) = L(H)\} \in coRE \setminus RE$.
First, $ALL_{CFG} = \{< G > \mid L(M) = \Sigma^*\}$ reduces to EQ_{CFG} since we can feed EQ_{CFG} with the pair $\langle G, H\rangle$, where G is the input to ALL_{CFG}, and H such that it generates Σ^*. It is known that $ALL_{CFG} \notin RE$, therefore, we conclude that $EQ_{CFG} \notin RE$.
To see that $EQ_{CFG} \in coRE$, consider a machine that checks for each word $w \in \Sigma^*$, whether both G and H generate w.

3. (a) Let f be the reduction from A to B. We claim that f is also the reduction from $\overline{A}$ to $\overline{B}$:

$$x \in \overline{A} \Rightarrow x \notin A \Rightarrow f(x) \notin B \Rightarrow f(x) \in \overline{B}.$$

Similarly,

$$x \notin \overline{A} \Rightarrow x \in A \Rightarrow f(x) \in B \Rightarrow f(x) \notin \overline{B}.$$

(b) We showed that $A_{TM} \in RE \setminus R$. It follows that $A_{TM} \notin coRE$, as a language that is both in RE and $coRE$ is in R. We also showed that $E_{TM} \in coRE$. Therefore, it can't hold that $A_{TM} \leq_m E_{TM}$ because that would imply that $A_{TM} \in coRE$.

4. (a) Let L_1, L_2 be two arbitrary languages in R which are not Σ^* or $\emptyset$. Let x_y and x_n be two fixed inputs to L_2, such that $x_y \in L_2$ and $x_n \notin L_2$. Then the reduction f from L_1 to L_2 will do as follows: on an input w to L_1 it will run the machine M_1 that decides L_1. Since $L_1 \in R$ such M_1 exists. Then if M_1 accepts w the reduction will output x_y otherwise it will output x_n. It follows that $w \in L_1 \Leftrightarrow f(w) \in L_2$.

   (b) The answer is no. A trivial counterexample is $L_1 = \{a^n b^n | n \geq 0\}$ and $L_2 = a$. $L_1 \leq_m L_2$ by the previous section. From a more meaningful point of view, if the statement were true, it would entail $R = REG$ as both $\emptyset$ and $\Sigma^* \in REG$.

5. Let $A_0, A_1, A_2, \ldots$ be an infinite sequence of languages over alphabet Σ, such that $A_0 \geq A_1 \geq A_2 \geq \ldots$, and $A_0 \in R$.

   For each one of the following languages we state the smallest class of languages that contains it from: R, RE, or not in any of these classes.

   (a) $A_k \in R$. We show this by induction. The base case: $A_1 \leq A_0$ and $A_0 \in R$ and so $A_1 \in R$. Now, $A_i \leq A_{i-1}$ and by induction hypothesis $A_{i-1} \in R$ and so $A_i \in R$.

   (b) $B_r = \bigcup_{k=0}^{r} A_k \in R$. The language B_r is a finite union of r languages in R, namely $A_1, A_2, \ldots, A_r$. As a finite union of languages in R also B_r is in R. (We can use induction to show that such finite union is decidable: clearly $B_2 = A_1 \cup A_2 \in R$ and $B_k = B_{k-1} \cup A_k \in R$, using the induction hypothesis that $B_{k-1} \in R$.)

   (c) We give an example to show that $B_\infty = \bigcup_{k=0}^{\infty} A_k \notin RE \cup coRE$.

   Let $x \in \Sigma^*$ be some fixed word in ALL_{TM}. Let $w_1, w_2, \ldots$ be an enumeration of all the words in Σ^*. For every $i > 0$ set A_i as follows: if $w_i \in ALL_{TM}$ then $A_i = \{w_i\}$, otherwise $A_i = \{x\}$.

   For each $i \geq 0$ language A_i contains one word and is therefore in R. Therefore, by (4a), for every i there is a reduction from A_i to A_{i+1}.

   However, $B_\infty = \bigcup_{k=0}^{\infty} A_k = ALL_{TM}$ and by (2a) this language is not in $RE \cup coRE$.

Computability - Exercise 9

Due Monday, July 2

1. (a) Prove that polynomial-time mapping reductions are transitive. That is, prove that if $L_1 \leq_p L_2$ and $L_2 \leq_p L_3$, then $L_1 \leq_p L_3$.
   (b) (optional) Prove that polynomial-time mapping reductions are NOT symmetric. That is, there exist languages L_1, L_2 such that $L_1 \leq_p L_2$, but $L_2 \not\leq_p L_1$.
   (c) Prove that for every $L_1, L_2 \in P$, that are neither $\emptyset$ nor Σ^*, it holds that $L_1 \leq_p L_2$.

2. For a natural number k, we say that a formula φ is k-cnf if it is of the form $\bigwedge_{i=1}^{m}\left(\bigvee_{j=1}^{k}\ell_j^i\right)$, where ℓ_j^i are literals (i.e. variables or their negation).
   We define the language, k-SAT $= \{\varphi : \varphi$ is a satisfiable k-cnf formula$\}$.
   (a) Prove that for every $k > 3$, k-SAT is NP-complete. Recall that 3-SAT is NP-complete.
   (b) Prove that 2-SAT is in P.
   Hint: Note that $a \vee b$ is equivalent to $\neg a \to b$ and $\neg b \to a$.

3. We saw in class that SUBSET-SUM where the numbers are written in decimal is NP-complete.
   (a) We define the binary version of subset sum (where all the numbers are given in binary) by:
   BSUBSET-SUM $= \{\langle n_1, \ldots n_k, t\rangle \mid n_1, \ldots n_k, t$ are given in binary and $\exists b_1 \ldots, b_k \in \{0,1\}$ such that $\sum_{i=1}^{k} b_i n_i = t\}$.
   Is BSUBSET NP-complete? Is it in P?
   (b) (optional) We define the unary version of subset sum (where all the numbers are given in unary) by:
   USUBSET-SUM $= \{\langle 1^{n_1}, \ldots 1^{n_k}, 1^t\rangle \mid \exists b_1 \ldots, b_k \in \{0,1\}$ such that $\sum_{i=1}^{k} b_i n_i = t\}$.
   Is USUBSET NP-complete? Is it in P? (prove your answer).
   (c) Define CLIQUE$_{2007}$ $= \{G : G$ is an undirected graph that contains a clique of size 2007.$\}$
   Is CLIQUE$_{2007}$ NP-complete? Is it in P?

   Comment: We didn't give the option that a language is neither NP-complete nor in P. If NP$\neq$P then such languages exist, but proofs might be hard for an ex'.

4. A relation $R = \{(x, y) \in \Sigma \times \Sigma\}$ is called an *NP-relation* if both of the following hold:

(a) There exists some polynomial $p(\cdot)$ such that for all $(x, y) \in R$ it holds that $|y| \leq p(|x|)$. (That is the length of the second coordinate is polynomial in the length of the first coordinate).

(b) There is a deterministic polynomial time machine that decides R.

For an NP-relation R, define $L_R = \{x \in \Sigma^* \mid \text{ there exists } y \in \Sigma^* \text{ such that } (x, y) \in R\}$.

A language L is an NP-relation language iff there exists an NP-relation R such that $L = L_R$.

(a) Prove that a language is in NP iff it is an NP-relation language.

(b) We saw in class a model of a Turing machine with a special "guess" tape. A deterministic Turing machine with a guess tape accepts an input $x \in \Sigma^*$ iff there exists some $y \in \Sigma^*$ such that the machine accepts if it starts running when the input tape contains x and the guess tape contains y. Note that the running time of such machine is measured in terms of the length of the input (not the guess). Prove that NP is the class of languages that are accepted in polynomial time by Turing machines with a guess tape.

Computability - Exercise 9 - Solution

1. (a) Let f be a polynomial time mapping reduction from L_1 to L_2, and let g be such a reduction from L_2 to L_3. We claim that $g \circ f$ is a polynomial time mapping reduction from L_1 to L_3. We have

$$x \in L_1 \Leftrightarrow f(x) \in L_2 \Leftrightarrow g \circ f(x) \in L_3.$$

Moreover, we assume that the running time of f bounded by some polynomial p, and the running time of g is bounded by q. Therefore, $|f(x)| \leq p(|x|)$, and thus the running time of $g \circ f$ is at most $q(p(|x|))$, which is still a polynomial.

(b) Let $L_1 = \{0\}$, and $L_2 = ALL_{TM}$ ($L_1 \in P$. Recall that $L_2 \notin R$). Let T be a Turing machine that immediately accepts any input, and T' a machine that immediately rejects any input. Therefore, $\langle T \rangle \in L_2$, while $\langle T' \rangle \notin L_2$. Consider the following mapping from L_1 to L_2: 0 is mapped to $\langle T \rangle$, and all other words are mapped to $\langle T' \rangle$. This is obviously a polynomial time reduction. On the other hand, by Lemma 1 as follows in 2-(a) and since L_2 is not decidable, $L_2 \not\leq_p L_1$.

(c) Let $y \in L_2$, and $y' \notin L_2$. For $x \in \Sigma^*$, define the function

$$f(x) = \begin{cases} y & x \in L_1 \\ y' & x \notin L_1 \end{cases}$$

f is a reduction, since $x \in L_1 \Leftrightarrow f(x) \in L_2$. It also runs in polynomial time, since it can be determined in polynomial time whether $x \in L_1$.

2. (a) k-SAT$\in$NP, with a satisfying assignment as witness. For NP-hardness, we show a poly-time reduction from 3SAT, that is known to be NP-hard. Given a 3CNF formula $\varphi = \wedge_{i=1}^{m} C_j$, where $C_j = l_j^1 \vee l_j^2 \vee l_j^3$, the reduction outputs the k-CNF formula $\varphi' = \wedge_{i=1}^{m} C'_j$, where $C'_j = l_j^1 \vee l_j^2 \vee \underbrace{l_j^3 \vee ... \vee l_j^3}_{k-2 \text{ times}}$. Clearly, an assignment satisfies φ iff it satisfies φ'.

(b) 2-SAT$\in$P. Given a 2CNF formula φ over the variables $\{x_1, ..., x_n\}$, we costruct a directed graph $G = \langle V, E \rangle$, where:

$$V = \{x_1, ..., x_n, \neg x_1, ..., \neg x_n\}$$

$$E = \{\langle \neg l_1, l_2 \rangle , \langle \neg l_2, l_1 \rangle : l_1 \vee l_2 \text{ is a clause in } \varphi\}$$

For example, if we have the clause $(x_3 \vee \neg x_7)$ in φ, then we will have a both the directed edges $\langle \neg x_3, \neg x_7 \rangle$ and $\langle x_7, x_3 \rangle$. (Note that $a \vee b$ is equivalent to $\neg a \Rightarrow b$ and to $\neg b \Rightarrow a$). We denote by $a \rightarrow\rightarrow b$ the situation in which there exists a directed path from a to b.

Claim: $\varphi \notin$2-SAT $\Leftrightarrow$ there exists $1 \leq i \leq n$ such that G contains both $x_i \rightarrow\!\!\!\!\rightarrow \neg x_i$ and $\neg x_i \rightarrow\!\!\!\!\rightarrow x_i$.

Proof: Assume first that G contains both paths for some x_i. Assume by way of contradiction, that there's also a satisfying assignment π for φ. Assume w.l.o.g. that $\pi(x_i) = T, \pi(\neg x_i) = F$, and look on the directed path from x_i to $\neg x_i$. The path is logically equivalent to a sequence of logical implications, $x_i \Rightarrow y$, ..., $z \Rightarrow \neg x_i$. Thus, since $\pi(x_i) = T$ and all the implications are satisfied, it must be that $\pi(\neg x_i) = T$ reaching contradiction.

For the other direction, Assume now that for all $1 \leq i \leq n$, either G does not contain a path $x_i \rightarrow\!\!\!\!\rightarrow \neg x_i$ or G does not contain a path $\neg x_i \rightarrow\!\!\!\!\rightarrow x_i$ (of course it might be that G contains neither). We construct a satisfying assignment π in the following way:

For $i = 1, \ldots, n$:

- If $\pi(x_i)$ is already defined, move on to the next i.
- If G contains $x_i \rightarrow\!\!\!\!\rightarrow \neg x_i$, set $\pi(x_i) = F$. Otherwise, set $\pi(x_i) = T$. Set $\pi(\neg x_i) = \neg\pi(x_i)$.
- If $\pi(x_i) = T$, then for every literal l reachable from x_i on the graph, where $\pi(l)$ is still unset, set $\pi(l) = T$.
- If $\pi(\neg x_i) = T$, then for every literal l reachable from $\neg x_i$ on the graph, where $\pi(l)$ is still unset, set $\pi(l) = T$.

We show now that π satisfies φ, by showing that it satisfied all clauses. Let $(l_1 \vee l_2)$ be a clause in φ. Assume without loss of generality, that l_1 is assigned value before l_2. If $\pi(l_1) = T$, then we're done. Otherwise, there's an edge $\langle \neg l_1, l_2 \rangle$ in the graph. So once l_1 is assigned F, l_2 would also be assigned a value, and $\pi(l_2) = \pi(\neg l_1) = T$. Thus, the clause is satisfied.

The claim above suggests the following algorithm to decide 2-SAT. Given a 2CNF formula, construct G as above. For each $1 \leq i \leq n$, run BFS to check if there is a path from x_i to $\neg x_i$ and from $\neg x_i$ to x_i. If there is an i for which these two paths exist, *reject*. Otherwise, *accept*.

It takes $O(n+m)$ to construct the graph (where n is the number of variables and m is the number of clauses), and $O(n(n+m))$ to search for the paths. Altogether, $O(n(n+m))$.

3. (a) $BSUBSET{-}SUM$ is NP-complete. $BSUBSET{-}SUM \in NP$: a witness is $b_1, b_2, \ldots, b_k$ such that $\sum_{i=1}^{k} b_i n_i = t$. We demonstrate that BSUBSET-SUM is NP-hard by showing that $SUBSET - SUM \leq_p BUSBSET - SUM$; this is sufficient since we have seen in class that SUBSET-SUM is NP-hard.

   Indeed, given an input for the SUBSET-SUM problem (where the numbers $n_1, \ldots, n_k, t$ are given in decimal), the reduction simply maps each n_i and t to their binary representation. This can be done using the usual algorithm: at each stage, the next digit of the binary representation is the current decimal number mod 2; the current decimal number is divided by 2 (integer division). The running time of this algorithm is linear in the size of the input, and the overhead in the size of the input is in a constant factor $(\ln 10 / \ln 2)$. Furthermore, it is clear that this is a reduction.

   (b) $USUBSET{-}SUM \in P$. A suitable (dynamic programming) polynomial time algorithm was presented in the algorithms course. The algorithm builds a binary matrix M of size

$k \times t$. The (i,j)'th cell contains 1 iff there is a subset of $\{n_1, \ldots, n_i\}$, such that the sum of the elements in the subset is j, and the subset includes n_i. The matrix is filled column-by-column, according to the formula

$$M(i,j) = \begin{cases} 1 & j = n_i \vee \exists l \leq i-1 \ s.t. \ M(l, j-n_i) = 1 \\ 0 & else \end{cases}.$$

There is a subset-sum iff there is exists l such that $M(l,t) = 1$. The running time is polynomial in t and k.

(c) $CLIQUE_{2007} \in P$. In order to prove this, we present an algorithm which determines whether a given graph contains a clique of size 2007 in polynomial time. The algorithm simply scans all possible choices of 2007 vertices from V, and accepts if and only if one of these sets of vertices is a clique. Since 2007 is a constant, each of the clique-tests can be executed in constant time. The number of tests to be performed is bounded by $\binom{n}{2007} \leq n^{2007}$.

4. The proof of (b) is a repetition of the proof given in Recitation 6 for the equivalence of the computing powers of deterministic and nondeterministic Turing machines. (Of course one must note that everything remains polynomial).

We prove (a). Let L be a language in NP, then there exists a nondeterministic polynomial time machine M that decides L. For an input x there is a witness y that corresponds to the guesses made during the accepting computation of M on x. Let
$R = \{(x,y) \in \Sigma^* \mid x \in L$ and y encodes the guesses made during the accepting computation of M on $x\}$. Clearly R is an NP relation and $L = L_R$.

On the other hand, let $L = L_R$ for some NP-relation R. Then, there exists some polynomial p such that $|y| \leq p(|x|)$ for all $(x,y) \in R$, and a deterministic machine M that decides R. We construct a nondeterministic machine M' that on input x first guesses a y of length $p(|x|)$ or less, and then runs M on (x,y). Clearly M' is (nondeterministic) polynomial time machine. The machine M' accepts x iff there exists some y such that $(x,y) \in L$. That is $L(M') = L_R = L$

Computability - Exercise 11

Due Monday, July 9

1. We proved in class that the language $HAMPATH = \{\langle G, s, t\rangle \mid G \text{ is a directed graph that contains a Hamiltonian path from } s \text{ to } t\}$ is NP-complete. A **Hamiltonian cycle** is a cycle (a path that begins and ends in the same vertex), that contains each vertex exactly once.

   Prove that the following languages are NP-complete.

   (a) $HAM = \{\langle G\rangle \mid G \text{ is a directed graph that contains a Hamiltonian cycle}\}$.

   (b) $\exists HAMPATH = \{\langle G\rangle \mid G \text{ is a directed graph that contains a Hamiltonian path}\}$.

2. Assume there exists a language L such that $L \leq_p (01)^*$ and $3SAT \leq_p L$.

   For each of the following propositions, determine whether the assumption above implies the proposition, and justify your answer.

   (a) $L^c \in$P. (Where $L^c = \Sigma^* \setminus L$ is the complement of L.)

   (b) P=NP.

   (c) $(01)^* \leq_p L$.

   (d) L is NP-complete.

3. A class $\mathcal{C}$ of languages is **closed under poly-time Karp reductions** if for every two languages $L, L' \subseteq \Sigma^*$, it holds that if $L \in \mathcal{C}$ and $L' \leq_p L$ then $L' \in \mathcal{C}$.

   For each of the following classes determine whether or not it is closed under poly-time Karp reductions, and justify your answer.

   (a) P

   (b) $DTIME(log(n))$

   (c) $DTIME(n^2)$

   (d) $EXP = \bigcup_{k>0} DTIME(2^{n^k})$

   (e) (optional) $\bigcap_{\delta>0} DTIME(2^{n^\delta})$

4. The Class *co-NP* is the class of languages whose complement is NP.

   Thus, co-NP= $\{L \subseteq \Sigma^* \mid (\Sigma^* \setminus L) \in NP\}$

(a) Give formal definitions for co-NP hard languages and for co-NP complete languages. Note, the definitions should not refer directly to the class NP.

(b) Prove that a language $L \subseteq \Sigma^*$ is co-NP hard iff $\Sigma^* \setminus L$ is NP-hard.

(c) Prove that a language $L \subseteq \Sigma^*$ is co-NP complete iff $\Sigma^* \setminus L$ is NP-complete.

(d) For $n \geq 0$, a finite sequence of natural numbers $\overline{a} = \langle a_1, \ldots, a_n \rangle$ is *half-sum breakable*, if there exists $j \leq n$ and a subsequence $\langle a_{i_1}, \ldots, a_{i_j} \rangle$ such that $\sum_{k=1}^{j} a_{i_k} = \frac{1}{2} \sum_{i=1}^{n} a_i$.
Let $L = \{\overline{a} \mid \overline{a}$ is a finite sequence of natural numbers that is not half-sum breakable$\}$. Show that L is co-NP complete.

Note that in this case, as in every case in which nothing is said about the representation of numbers, we assume that all the numbers are given in binary.

5. Show that $PSPACE$ is closed under the operations union, complementation and Kleene star.

Computability 2007: Exercise 10 solution

Reminders:

- $L \in NP$ if there exists a Turing machine $V(x, y)$, with running time polynomial in $|x|$, such that $x \in L$ iff $\exists y$ such that $V(x, y)$ accepts. y is referred to as a 'proof' or as a 'witness' to x's membership in L.
- L is *NP-hard* if for any $L' \in NP$, $L' \leq_p L$.
- L is *NP-complete* if it both a member of NP and is *NP-hard*.
- Proving NP hardness is done by either one of two ways: direct proof as in the Cook-Levin theorem which proves that $3SAT$ is *NP-complete*. *NP-hard*. This is done for one language ($3SAT$), and other languages are shown to be *NP-hard* via reduction. That is, if L is NP-hard, and $L \leq_p L'$ then L' is *NP-hard*.

1. $HAMPATH = \{\langle G, s, t\rangle \mid G$ is a directed graph that contains a Hamiltonian path from s to $t\}$.
   (a) $HAMCYCLE = \{\langle G\rangle \mid G$ is a directed graph that contains a Hamiltonian cycle $\}$.
   $HAMCYCLE \in NP$: A proof of $\langle G\rangle \in HAMCYCLE$ is the sequence of vertices that comprise a Hamiltonian cycle in G. This can be verified in polynomial time.
   $HAMCYCLE$ is *NP-hard*: $HAMPATH \leq_p HAMCYCLE$. Given $\langle G, s, t\rangle$, the reductions creates a new graph G' by adding a new vertex v to G, along with the edges (t, v) and (v, s). A Hamiltonian cycle in G' must traverse v and must do so through above edges (t, v) and (v, s). The existence of such a Hamiltonian cycle in G' therefore implies the existence of a Hamiltonian path from s to t. Conversely, a Hamiltonian path from s to t in G can be extended to a Hamiltonian cycle in G'.
   (b) $\exists HAMPATH = \{\langle G\rangle \mid G$ is a directed graph that contains a Hamiltonian path $\}$.
   $\exists HAMPATH \in NP$: A proof of $\langle G, k\rangle \in IS$ is a sequence of vertices that comprise a Hamiltonian path in G. This can be verified in polynomial time.
   $HAMCYCLE$ is *NP-hard*: $HAMPATH \leq_p \exists HAMPATH$. Given $\langle G, s, t\rangle$, the reductions creates a new graph G' by adding a new vertices u, v to G, along with the edges (u, s) and (t, v). A Hamiltonian path in G' must begin with u and end with v. Its existence implies the existence of a Hamiltonian path from s to t. Conversely, a Hamiltonian path from s to t in G can easily be extended to a Hamiltonian path in G'.

2. Given L, such that $L \leq_p (01)^*$ and $3SAT \leq_p L$ we can conclude:
   (a) $L^c \in P$. First, $L \in P$, as $REG \subset P$. Secondly, P is closed under complement.

(b) $P = NP$. $L \in P$ and $3SAT \leq_p L$ and so $3SAT \in P$. Also, $3SAT$ is *NP-hard* and so every language in NP (polynomially) reduces to it, and is therefore in P (reductions are transitive).

(c) $(01)^* \leq_p L$. From $3SAT \leq_p L$ we learn that L is non-trivial (i.e., neither $\emptyset$ nor Σ^*). The same is true for $(01)^*$. Both languages are in P and hence reduce to one another.

(d) L is *NP-complete*. $3SAT \leq_p L$ so L is *NP-hard*. $L \in P \subseteq NP$ so $L \in NP$.

3. The question is: does $L' \leq_p L$ and $L \in \mathcal{C}$ implies that $L' \in \mathcal{C}$?

(a) P is obviously closed under polynomial reductions. Assume that the reduction runs in time $O(n^k)$ and that a machine that decides L runs in time $O(n^m)$ (where n is the length of the input). The suggested polynomial time machine for L' computes the reduction and then runs the algorithm for L. Running the reduction takes $O(n^k)$ time steps, and therefore the output is at most $O(n^k)$ letters long. Let c be the constant in the $O(\cdot)$ definition, then the output is of length bounded by cn^k. Therefore running the machine that decides L on the output takes at most $O((cn^k)^m) = O(n^{km})$ time. The entire procedure takes $O(n^k + n^{km}) = O(n^{km})$ time steps.

(b) $DTIME(log(n))$ is not closed under polynomial reductions. Using the time hierarchy theorem, there exists a language L' decidable in time $O(n^3)$, but not in time $o(n^3/log(n^3)$ and particularly, not decidable in time $O(log(n))$. A polynomial time reduction can decide if $x \in L$ and prepend one bit that signifies the result to x as its output. And now, a logarithmic time algorithm can decide the language $L = \{x \mid$ the first symbol in x is 1 $\}$. Note: Machines operating in time logarithmic in n are very limited. Note that they cannot even read the input (since that would take $O(n)$ time). In fact, they can't even know how long they can run, since they can't read the input. A proof showing that $DTIME(log(n)) = DTIME(1)$ can be constructed from this observation.

(c) $DTIME(n^2)$ is not closed under polynomial reductions. Use the same argument as in the previous section.

(d) EXP is closed under polynomial reductions. As in the case of P, the machine that first performs the reduction, and then decides whether the input of the reduction is in the language we reduced to, takes exponential time. In this case, the reduction runs $O(n^k)$ time, and produces an output of length at most $O(n^k)$. Now the exponential time machine runs (in time $O(2^{(m^l)})$ on input of length $m = n^k$ so it's running time is $O(2^{((m^l)^k)}) = O(2^{(m^{lk})})$ so the running time of the entire machine is still exponential.

(e) $\bigcap_{\delta>0} DTIME(2^{n^\delta})$ is closed under polynomial reductions.
Intuitively, since $\forall \delta > 0, P \subset DTIME(2^{n^\delta})$, we have $P \subset \bigcap_{\delta>0} DTIME(2^{n^\delta})$, and so, in this class we can carry out the reduction without exceeding the time constraint (as in the case of EXP).
Formally, we assume the existence of a reduction with a running time $O(n^k)$. Given $\delta_0 > 0$, set $\mu = \frac{\delta_0}{k}$. L can be decided in time $O(2^{n^\mu})$ (since it is the intersection and thus in $O(2^{n^\delta})$ for all δ). Therefore, the machine that first runs the reduction and then the machine deciding L, runs in time $O(n^k + 2^{(n^k)^\mu}) = O(2^{(n^k)^{\frac{\delta}{k}}}) = O(2^{n^\delta})$, thus proving the claim.

4. (a) A language L is co-NP hard if for every language $L' \in \text{co} - \text{NP}$ we have $L' \leq_p L$. A language L is co-NP complete if it both in co-NP and is co-NP hard.

   (b) Let L be a language whose complement L^c is NP-hard. For every $L_1 \in \text{co} - \text{NP}$, we know that L_1^c is in NP. Therefore $L_1^c \leq_p L^c$. The reduction f is polynomial time computable function such that $x \in L_1^c \iff f(x) \in L^c$. Therefore, $x \in L_1 \iff f(x) \in L$. Thus f is also a reduction from L_1 to L implying that $L_1 \leq_p L$

   (c) Let L be a language whose complement L^c is NP-complete. Then, L^c is NP-hard and therefore L is co-NP hard. Also L^c is in NP, and therefore L is in co-NP. Thus, L is co-NP complete.

   (d) By the previous clauses it is enough to prove that $\overline{L} = \{\overline{a} \mid \overline{a} \text{ is a finite sequence of natural numbers that } \textit{is} \text{ half-sum breakable}\}$ is NP-complete. It is easy to see that $\overline{L}$ is in NP, the witness is the sub-sequence, and it easy to check in polynomial that the subsequence sum is half of the total sum. As for hardness, we proved in the tirgul that $SUBSET\text{-}SUM$ is NP-hard, we will show a reduction $SUBSET\text{-}SUM \leq_p \overline{L}$. For $\overline{a} = \langle a_1, \ldots, a_n \rangle$ and t input for $SUBSET\text{-}SUM$, we denote by s the sum of the sequence $s = \sum_{i=1}^{n} a_i$. Let $b = 2006s - l$ and $c = 2006s - (s - l)$. The reduction computes b and c and outputs $\overline{a'} = \langle a_1, \ldots, a_n, b, c \rangle$ (intuitively as input to $\overline{L}$). Clearly the reduction can be done in polynomial time. We shall now see that $\overline{a'}$ is half-sum breakable iff there is a subsequence of $\overline{a}$ whose sum is exactly l.

   Assume first that there is a subsequence $\overline{a_s}$ of $\overline{a}$ whose sum is exactly l. Then, the sum of $\overline{a_s}$ and b is exactly $2006s$. The sum of all the elements of $\overline{a}$ that are not in $\overline{a_s}$ and c is exactly $2006s$ as well. Therefore, $\overline{a'}$ is half-sum breakable.

   Assume now that $\overline{a'}$ is half-sum breakable. Note that the sum of the entire sequence $\overline{a'} = ((\sum_{i=1}^{n} a_i) + b + c)$ is exactly $2 * 2006s$. Denote by $\overline{a'_s}$ a subsequence whose sum is exactly half the total sum (namely $2006s$). It is impossible that $\overline{a'_s}$ contains both b and c since the sum of those two elements is larger then $2006s$. For the same reason it is impossible that $\overline{a'_s}$ contains neither b nor c.

   Assume w.l.o.g. that $\overline{a'_s}$ contains b and does not contain c. Then the sum of the a_i's in $\overline{a'_s}$ must be exactly l. Therefore $\overline{a}$ contains a subsequence whose sum is exactly l.

5. **Union**: Let $L_1, L_2 \in$ PSPACE, and let M_1, M_2 be Turing Machines that decide them in polynomial space. We can decide $L = L_1 \cup L_2$ in polynomial space, by first simulating M_1, then M_2, and accepting if at least one of them accepts.

   Complement: Let $L \in$ PSPACE, and let M be a TM that decides it in polynomial space. Since M is deterministic, we can decide L^c in polynomial space by simulating M, and accepting iff it rejects.

   Kleene star: Let $L \in$ PSPACE, and let M be a TM that decide it in polynomial space. The following non-deterministic TM decides L^* in polynomial space: the machine guesses the partition of the input word into substrings in L (the guess can be saved in polynomial space), and then checks that each of them is indeed in L by simulating M. Since PSPACE=NPSPACE, we get that $L^* \in$ PSPACE.

Computability - Exercise 11 - not for submission

Note: Some of the questions refer to the classes L and NL that will be taught next week.

1. Prove that the following language is PSPACE-complete.

$$\text{CONT} = \{\langle A_1, A_2\rangle : A_1 \text{ and } A_2 \text{ are NFAs and } L(A_1) \subseteq L(A_2)\}$$

2. Show that NL is closed under the operations union, intersection and star.

3. Prove that the following language is in NL.

$$\text{FAR} = \{\langle G, s, t, k\rangle : \text{ all paths from } s \text{ to } t \text{ are of length at least } k\}$$

   Note that if there is no path from s to t, then $\langle G, s, t, k\rangle \in \text{FAR}$ for all k.

4. (a) Show that $A_{DFA} = \{\langle D, w\rangle : D \text{ is a DFA and } w \in L(D)\}$ is in L.
   (b) Show that $A_{NFA} = \{\langle N, w\rangle : N \text{ is a NFA and } w \in L(N)\}$ is NL-complete.

5. (optional) Let $G = (V, E)$ be a directed graph on vertices from the set $\{0,1\}^n$. As this set is of cardinality 2^n the graph is too large to be given as an input. So instead we represent the graph by a TM M_G that computes the function:

$$M_G(u, v) = \begin{cases} 1 & \langle u, v\rangle \in E \\ 0 & otherwise \end{cases}$$

   That is, by querying M_G, we can determine whether a certain edge occurs in G.

   Now consider the language:

   $Implicit - Connectivity = \{\langle M_G, s, t\rangle : M_G \text{ is a deterministic TM as above and uses at most polynomial-space, } s, t \text{ are vertices in } G, \text{ and there is a path from } s \text{ to } t \text{ in } G\}$

   Show that this language is $PSPACE$-complete.

Computability - Exercise 11 - Solution

1. On page 3.

2. Let M_1 and M_2 be two non deterministic log space machines that recognize the languages L_1 and L_2, respectively. First, we prove closure to union. To construct a machine for $L_1 \cup L_2$, simply use the first non deterministic bit to decide whether to run M_1 or M_2. To construct a machine for $L_1 \cap L_2$, one can run M_1 first, and only if it accepts run M_2. The space can be reused so no extra space is needed.

   To construct a machine for L_1^* we need a slightly more complicated construction. A word $w \in \Sigma^n$ is in L_1^* iff it can be broken into subwords $w = w_1 \cdot w_2 \cdots w_n$ such that for all i the subword w_i is in L_1. Our machine, which accepts L_1^*, will use the non determinism to guess the length of the first subword w_1, and simulate M on it. (Note there is a slight complication here, our machine should remember the length of w_1 and even if M tries to move it's input-reading head from it, simulate M as if the input tape contains only w_1). If M rejects w_1 our machine will reject. Otherwise, our machine will check to see if w_1 already contains all the input, or otherwise, continue to guess w_2 in a similar manner. The process ends either in a rejection, or when all of the input w was covered by accepted subwords in which case we accept.

3. On page 4.

4. (a) Following the computation of a DFA is easy, all one has to do is to keep in the memory the current state (of size $O(\log(n))$), the index of the symbol read (of size $O(\log(n))$), and the symbol read (of size $O(1)$). What one has to do is to go over the transition table, and find the relevant transition (easily done using $O(\log(n))$ auxiliary memory). Repeat the process for all letters in the word, and finally check if the state at the end of the run is accepting (again easily done using $O(\log(n))$ auxiliary memory).

   (b) First, to see that the language is in NL, note that the only difference from the deterministic case, is that when going over the transition table, the machine, does not necessarily take the first suitable transition, but rather uses a non deterministic bit, to decide if to take the transition, or search for another one. In case all the transition table was read, and no transition was taken, the machine rejects.

   Next, to see that the language A_{NFA} is NL-hard, we reduce $PATH$ to it. Given $\langle G, s, t\rangle$, where $G = (V, E)$ is a graph and $s, t \in V$ are vertices, we construct an NFA $A = \langle \Sigma, Q, Q_0, \delta, F\rangle$ and a word w such that $w \in L(A)$ iff there is a path in G from s to t. We set Σ to be a one letter alphabet $\{a\}$. We set $Q = V$ where $Q_0 = \{s\}$, and $F = \{t\}$. As for δ, for a state (i.e. vertex) u, other then t, we set $\delta(u, a) = \{v \mid (u, v) \in E\}$. For the state (vertex) t we set $\delta(t, a) = \{t\}$. Finally, set $w = a^n$. Clearly, all these simple

constructions can be made using logarithmic space. It is also easy to see that there is a path from s to t iff there is an accepting run of A on a^n.

5. We first show that *Implicit-Connectivity* $\in$ PSPACE. We use the fact that PSPACE=NPSPACE. The following polynomial space algorithm decides the language. The algorithm first guesses a length l of an st-path, and then guesses its $l-2$ inner vertices.

   - Guess a number $l \in \{2, 3, \ldots, 2^n\}$. Guess $v_1 \in \{0,1\}^n$, and simulate M_G on the input (s, v_1). If M_G rejects then "reject".
   - $i \leftarrow 2$.
     Repeat until i is equal to $l-1$:
     - Guess $v_i \in \{0,1\}^n$. If $M_G(v_{i-1}, v_i)$ rejects then "reject".
     - $i \leftarrow i+1$.
   - If $M_G(v_{l-1}, t)$ rejects then "reject", otherwise "accept".

   The simulation of the polynomial space machine M_G takes polynomial space. Note that this is almost the same algorithm used to show that $PATH \in$ NL, only applied to a huge graph.

   We next show that *Implicit-Connectivity* is PSPACE-hard. Let L be an arbitrary language in PSPACE, and M_L a machine deciding L using no more then $p(n)$ space. We show that $L \leq_p$ *Implicit-Connectivity*.

   The idea is very simple, the graph G (given implicitly) will represent the configuration graph of M_L. Throughout the proof we use n to represent the size of the input given to M_L and m to represent the length of a binary description vertex in the *Implicit-Connectivity* graph.

   Given a word w, where $|w| = n$, the possible configurations of M_L when running on w are of length at most $p(n)$. Each configuration can be represented as a sequence of $p(n)$ cells where each cell contains either a symbol from Γ (i.e. a symbol M_L work alphabet) or a symbol from $\Gamma \times Q$ (i.e. a symbol and a state of M_L). Therefore, each cell can be represented by a constant number of bits (that does not depend on the input length), and the entire configuration can be represented by a sequence of $m = c_1 n$ bits. In addition we want to have one vector of m bits, that does not represent a configuration, represent a special vector we will call "accepted".

   The machine M_G is a machine that given two vectors (u, v) answers 1 either if v is a configuration successor to u in M_L, or if u is an accepting configuration and v is "accepted". Otherwise, the machine M_G answers 0. Note that it is easy to build, in polynomial time, such a machine M_G that works in polynomial space. The vertex s is the initial configuration (of M_L on w), and the vertex t is "accepted".

   Note that the entire reduction (which consists mostly of constructing M_G), is easy and can be done in polynomial time. It is also easy to see that M_L accepts w iff there is a path in G from s to t.

פתרון תרגיל מספר 11 בחישוביות - המשך

1. נוכיח תחילה כי CONT היא PSPACE- קשה.
נשתמש בסעיף א', ונראה כי $ALL_{NFA} \leq_p CONT$.
בהינתן קלט $<A>$ של ALL_{NFA}, ניצור קלט $<A_1,A_2>$ של CONT בצורה הבאה:
A_1 זהו NFA המקבל את כל המילים ($L(A_1)=\Sigma^*$), ו- $A_2=A$.
עתה, נבחין כי $<A>\in ALL_{NFA}$ אמ"מ $<A_1,A_2>\in CONT$:
אם $<A>\in ALL_{NFA}$ אזי $L(A_2)=L(A)=\Sigma^*$ ובוודאי כי $L(A_1)$ מוכלת ב $L(A_2)$.
אם $<A_1,A_2>\in CONT$ אזי $L(A_1)$ מוכלת ב $L(A_2)$. ידוע מהבנייה כי $L(A_1)=\Sigma^*$, לכן $L(A)=L(A_2)=\Sigma^*$.

ניתוח סיבוכיות: נותר להראות כי הרדוקציה פולינומית (שימו לב שלפי ההגדרה של PSPACE- קושי צריך להוכיח כי הרדוקציה נעשית **בזמן** פולינומי). הרדוקציה כאן מחשבת את האוטומט שמקבל את כל המילים, וחישוב זה ניתן לבצע במספר צעדים קבוע. כל מעבר על הקלט נעשה כמובן בזמן פולינומי. לכן הרדוקציה פולינומית.

נראה עתה כי $CONT\in PSPACE$:
כמסקנה ממשפט Savitch, NPSPACE=PSPACE. לכן, נובע גם coNPSPACE=PSPACE
(אם מכריעים שפה בזיכרון פולינומי, הרי שגם מכריעים את משלימתה בזיכרון פולינומי).
לכן, מספיק להראות כי $CONT\in coNPSPACE$.
כלומר, נרצה להכריע בצורה לא דטרמיניסטית, ובזיכרון פולינומי, האם עבור שני אוטומטים נתונים A_1,A_2 השפה המתקבלת על ידי A_1 **אינה** מוכלת בשפה המתקבלת על ידי A_2.
המכונה הלא דטרמיניסטית תפעל כך:
היא תנחש מילה w (בגודל של, נניח, עד סכום מספרי המצבים של האוטומטים), ותסמלץ את A_1 (בצורה לא דטרמיניסטית) על w. לגבי A_2, המכונה "תתייחס" אל ה-DFA השקול לו, ותבדוק האם יש מסלול מהמצב התחילי למצב מקבל ב-DFA.
המכונה תקבל אמ"מ A_1 קיבל את w ו- A_2 דחה את w.

הוכחת נכונות: ברור שאם אכן השפה המתקבלת על ידי A_1 אינה מוכלת בשפה המתקבלת על ידי A_2, אזי יהיה למכונה חישוב מקבל (בניחוש מילה שנמצאת בהפרש). שימו לב שבשביל להראות קבלה ב- NFA מספיק חישוב מקבל אחד, אך בשביל להראות דחייה צריך להוכיח דחייה של כל החישובים (או, מאותה סיבה, להראות דחייה על ה- DFA השקול).
גודל המילה שבחרנו יספיק, שכן לכל מילה המתקבלת על ידי אחד האוטומטים הגדולה מסכום מספרי המצבים, תהיה מילה קטנה מגודל זה שגם תתקבל באותה צורה- ע"י הורדת המעגלים שנוצרו.

ניתוח סיבוכיות זיכרון: גודל המילה w שהמכונה ניחשה מוגדר להיות פולינומי בגודל הקלט.
הסימולציה של A_1 נעשית בזיכרון פולינומי – פשוט שומרים באיזה מצב אנחנו, ואיזה אות קלט מ- w קראנו. כמו בסעיף הקודם של השאלה, ריצה על הDFA ש"בנינו" עדיין תהיה פולינומית בזיכרון (למרות שמספר המצבים עלול לגדול בצורה אקספוננציאלית), שכן מציאת מסלול מקודקוד לקודקוד (או לקבוצה של קודקודים) הינה ב-NL. כלומר, אנו משתמשים בזיכרון לוגריתמי לגבי אובייקט בעל גודל אקספוננציאלי, לכן סה"כ הזיכרון שמשתמשים בו הוא פולינומי.
נקודה חשובה היא שהDFA השקול אינו נבנה מפורשות (אז היה דורש זיכרון אקספוננציאלי) אלא המכונה מסמלצת ריצה של אוטומט החזקה תוך שימוש בתיאור ה- NFA המקורי.

3. נוכיח כי FAR∈coNL. מכיוון ש- NL=coNL, נובע כי FAR∈NL.
נגדיר את המחלקה המשלימה:
NOT-FAR = {(*G,s,t,k*) : there is a path from *s* to *t* of length < *k* }
ונראה כי NOT-FAR∈NL (לכן כמובן FAR∈coNL). נתבונן באלגוריתם (הלא דטרמיניסטי) הבא:

- נחש קודקוד v_1 ב- G.
- נחש שכן v_2 של v_1, נחש שכן v_3 של v_2, והמשך באופן דומה.
- אם הגעת לקודקוד t בפחות מ- k צעדים, קבל.
  אם ביצעת k-1 צעדים ולא הגעת ל-t, דחה.

ניתוח זיכרון: נסמן את מספר הקודקודים בגרף ב- n. במהלך האלגוריתם נצטרך לשמור את המידע הבא בזיכרון: הקודקוד בו נמצאים ברגע נתון (זיכרון logn – הייצוג הבינארי של מספר הקודקוד), מונה של מספר הצעדים שצעדנו (logn- הייצוג הבינארי של המונה שלכל היותר ימנה עד n>k).

נכונות האלגוריתם: אם יש מסלול קצר מ- s ל- t בגרף הנתון אזי כמובן ישנו ניחוש (כלומר ריצה לא דטרמיניסטית) עבורו נקבל את הגרף.