# INDIAN INSTITUTE OF TECHNOLOGY KHARAGPUR
## Mid-Autumn Semester 2018-19

**Date of Examination :** 20-09-2018 **Session(FN/AN)** AN **Duration** 2 hrs **Total Marks** 60
**Subject No :** CS41001 **Subject Name :** THEORY OF COMPUTATION
**Department/Centre/School :** Computer Science and Engineering
**Specific charts, graph paper, log book etc. required** No
**Special Instructions (if any)** Answer all questions. If required, you may make assumptions and state them upfront. No credit will be given to sketchy proofs and claims without proper reasoning.

---

1. A one-counter automaton is an automaton with a finite set of states $Q$ a two-way read-only input head and a separate counter that can hold any non-negative integer. The input $x$ is enclosed in endmarkers $\vdash, \dashv \notin \Sigma$ and the input head may not go outside the endmarkers. The machine starts in its start state $s$ with its counter empty and with its input head pointing to $\vdash$. In each step, it can test its counter for 0. Based on this information, its current state and the symbol its input head is currently reading, it can either add $1, -1$ to its counter and move its input head either left or right and enter a new state. It accepts by entering a distinguished final state $t$.

   (a) Give a rigorous formal definition of these machines, including a definition of acceptance. Your definition should begin as follows: "A one-counter automaton is a 7-tuple $\mathcal{M} = (Q, \Sigma, \vdash, \dashv, s, t, \delta)$, where ...". **[4]**

   **Solution:** A one-counter automaton is a 7-tuple $\mathcal{M} = (Q, \Sigma, \vdash, \dashv, s, t, \delta)$, where

   - $Q$ is a finite set of states
   - $\Sigma$ is the input alphabet
   - $\vdash$, $\dashv$ are left and right endmarkers
   - $s$ is the start state
   - $t$ is the accept state
   - $\delta : Q \times \Sigma \times \{Z, NZ\} \to Q \times \{L, R\} \times \{1, -1\}$ is the transition function with $Z, NZ$ denoting the counter being zero, non-zero respecively.

   The following restrictions apply:

   - $\forall p \in Q \, \forall A \in \{Z, NZ\}, \exists q \in Q \, c \in \{1, -1\}$ such that $\delta(p, \vdash, A) = (q, R, c)$
   - $\forall p \in Q \, A \in \{Z, NZ\}, \exists q \in Q \, c \in \{1, -1\}$ such that $\delta(p, \dashv, A) = (q, L, c)$
   - $\forall p \in Q \setminus t \, a \in \Sigma, \exists q \in Q \, Y \in \{L, R\}$ such that $\delta(p, a, Z) = (q, Y, 1)$
   - $\forall a \in \Sigma \, A \in \{Z, NZ\}, \exists Y \in \{L, R\} \, c \in \{1, -1\}$ such that $\delta(t, a, A) = (t, Y, c)$

   Define a configuration of $\mathcal{M}$ to be a string in $Q \times \Sigma^* \times \mathbb{N} \times \mathbb{N}$ containing the current state, contents of the tape, position of the tape head and value of the counter. The start configuration on input $x$ is $(s, \vdash x \dashv, 0, 0)$. Further define for $m \neq 0$

$$(p, z, n, m) \xrightarrow[\mathcal{M}]{1} \begin{cases} (q, z, n+1, m-1) \text{ if } \delta(p, z_n, NZ) = (q, R, -1) \\ (q, z, n+1, m+1) \text{ if } \delta(p, z_n, NZ) = (q, R, 1) \\ (q, z, n-1, m-1) \text{ if } \delta(p, z_n, NZ) = (q, L, -1) \\ (q, z, n-1, m+1) \text{ if } \delta(p, z_n, NZ) = (q, L, 1) \end{cases}$$

   and

$$(p, z, n, 0) \xrightarrow[\mathcal{M}]{1} \begin{cases} (q, z, n+1, 1) \text{ if } \delta(p, z_n, Z) = (q, R, 1) \\ (q, z, n-1, 1) \text{ if } \delta(p, z_n, Z) = (q, L, 1) \end{cases}$$

The machine $\mathcal{M}$ is said to accept a string $x$ if $(s, \vdash x \dashv, 0, 0) \xrightarrow[\mathcal{M}]{*} (t, \vdash x \dashv, m, n)$ for some $m, n \in \mathbb{N}$.

(b) Prove that the membership problem for deterministic one-counter automata is decidable: given $\mathcal{M}, x$, does $\mathcal{M}$ accept $x$? **[6]**

**Solution:** At any point of time, $\mathcal{M}$ can be in one of $|Q|$ possible states and its tape head can be in one of $|x| + 2$ possible positions. The machine loops on a given input whenever the first three components of a configuration $(p, z, n, m)$ repeats with a counter value $\geq m$. So, for a halting sequence of configurations, the counter value can never exceed $t = |Q| \times (|x| + 2)$. For otherwise, atleast one combination of state and tape-head position must repeat as the counter is incremented each step of the computation thus making the machine loop. This immediately gives us a decision procedure for testing if $\mathcal{M}$ accepts $x$ – simulate $\mathcal{M}$ on $x$ for $t^2$ steps; if the first three components of any configuration repeats with an increased value of counter, then the machine loops; otherwise, if $\mathcal{M}$ halts, check whether it enters the accept or reject state and accordingly determine if $\mathcal{M}$ accepts $x$.

2. For $A, B \subseteq \Sigma^*$, define

$$A/B = \{x \in \Sigma^* \mid \exists y \in B \quad xy \in A\}.$$

Show that if $A$ and $B$ are recursively enumerable, then so is $A/B$. **[10]**

**Solution:** Let $\mathcal{M}_A, \mathcal{M}_B$ be Turing machines accepting $A, B$ respectively. Define a TM $\mathcal{N}$ that on input $x$ does the following.

- For each $y \in \Sigma^*$, simulate $\mathcal{M}_B$ on $y$ on a time-shared basis. That is, simulate $\mathcal{M}_B$ on $y_1$ for one step and then simulate it on $y_2$ for one step and continue simulations for some fixed ordering $y_1, y_2, \ldots$ of strings in $\Sigma^*$.
- If $\mathcal{M}_B$ accepts, then simulate $\mathcal{M}_A$ on $xy$.
- Halt and accept if $\mathcal{M}_A$ accepts.

If $x \in A/B$, then for some $y \in \Sigma^*$, $\mathcal{M}_B$ accepts $y$ eventually and $\mathcal{M}_A$ accepts $xy$. Hence $A/B$ is recursively enumerable.

3. (a) Show that the following problems of pairs of Turing machines are undecidable by reduction.

(i) $\{(\mathcal{M}, \mathcal{N}) \mid L(\mathcal{M}) \cap L(\mathcal{N}) = \emptyset\}$ **[4]**

**Solution:** Let EI-TM denote the language. We show that $\neg$HP $\leq_m$ EI-TM. Let $(\mathcal{K}, x)$ be an instance of $\neg$HP. Construct $\mathcal{M}$ so that it accepts any input. Clearly $L(\mathcal{M}) = \Sigma^*$. Let $\mathcal{N}$ be defined so that, on input $y$ it does the following: store $y$ on a separate track; simulate $\mathcal{K}$ on $x$. If it halts, then accept $y$. If $(\mathcal{K}, x) \in \neg$HP, then $L(\mathcal{N}) = \emptyset$ and $L(\mathcal{M}) \cap L(\mathcal{N}) = \emptyset$. If $(\mathcal{K}, x) \notin \neg$HP, then $L(\mathcal{N}) = \Sigma^*$ and as a result $\mathcal{M} \cap \mathcal{N} \neq \emptyset$. Since $\neg$HP is undecidable, so is EI-TM.

(ii) $\{(\mathcal{M}, \mathcal{N}) \mid L(\mathcal{M}) \cap L(\mathcal{N}) \text{ is finite}\}$ **[4]**

**Solution:** The same reduction as the previous problem works here since $\emptyset$ is finite.

(b) Formalise and prove the following extension of Rice's theorem (of which the above results are special cases): every non-trivial property of pairs of r.e. sets is undecidable. **[8]**

**Solution:** Let $P$ be a non-trivial property of pairs of r.e. sets. Let $L_P = \{(\mathcal{M}, \mathcal{N}) \mid P(L(\mathcal{M}), L(\mathcal{N})) = \mathsf{T}\}$. Assume w.l.g. that $P(\emptyset, \emptyset) = \mathsf{F}$. Since $P$ is non-trivial, there exist r.e. sets $L_1, L_2$ such that $P(L_1, L_2) = \mathsf{T}$. Let $\mathcal{M}_1$, $\mathcal{M}_2$ be TMs recognising $L_1$, $L_2$ repectively. We show that HP $\leq_m L_P$. Let $\mathcal{N}, x$ be an instance of HP. Construct $\mathcal{M}_1'$ that on input $w$ does the following:

- Run $\mathcal{N}$ on $x$ ($\mathcal{N}, x$ are hardwired in the finite control of $\mathcal{M}_1'$).
- If $\mathcal{N}$ halts, simulate $\mathcal{M}_1$ on $w$.
- Accept if $\mathcal{M}_1$ accepts.

Similarly, construct $\mathcal{M}_2'$ that on input $w$ does the following:

- Simulate $\mathcal{N}$ on $x$
- If $\mathcal{N}$ halts, simulate $\mathcal{M}_2$ on $w$.
- Accept if $\mathcal{M}_2$ accepts.

If $\mathcal{N}$ halts on $x$, then $L(\mathcal{M}_1') = L(\mathcal{M}_1)$ and $L(\mathcal{M}_2') = L(\mathcal{M}_2)$. As a result, $P(L(\mathcal{M}_1'), L(\mathcal{M}_2')) = \mathsf{T}$. If $\mathcal{N}$ does not halt on $x$, then $L(\mathcal{M}_1') = L(\mathcal{M}_2') = \emptyset$ whence $P(L(\mathcal{M}_1'), L(\mathcal{M}_2')) = \mathsf{F}$. In other words, $(\mathcal{N}, x) \in \mathsf{HP} \Leftrightarrow (\mathcal{M}_1', \mathcal{M}_2') \in L_P$. Therefore, $L_P$ is undecidable.

4. Prove that a language $L$ is recursive~~ly enumerable~~ if and only if there is an enumeration machine enumerating the strings of $L$ in lexicographic order. **[10]**

**Solution:** Suppose that $L \subset \Sigma^*$ is a recursive language. Then there exists a total TM $\mathcal{K}$ deciding $L$. Construct an enumeration machine $\mathcal{M}$ that does the following for each string $y$ in $\Sigma^*$ according to the lexicographic order

- Simulate $\mathcal{K}$ on $y$.
- Enumerate $y$ if $\mathcal{K}$ accepts $y$
- If $\mathcal{K}$ rejects $y$, then do not enumerate.

Since $\mathcal{K}$ is total, all the each enumeration occurs after finitely many steps $\mathcal{M}$.

Conversely, suppose that there is an enumeration machine $\mathcal{M}$ enumerating strings of a language $L$ in lexicographic order. Define a TM $\mathcal{K}$ that does the following for each $y \in \Sigma^*$,

- Run $\mathcal{M}$ until it enumerates $y$ or a string that comes after $y$ in a lexicographic order.
- If $y$ is enumerated, then accept and halt.
- If the last string enumerated is different from $y$, then reject and halt.

Since $\mathcal{M}$ enumerates all strings of $L$ in lexicographic order, it should eventually enumerate $y$ or a string that follows $y$ if $y \in L$ in that order. The latter happens when $y \notin L$. As a result, in finite time $\mathcal{K}$ determines if $y \in L$ or not. Therefore, $\mathcal{K}$ is total and $L$ is recursive.

5. Show that Post's correspondence problem is decidable over the unary alphabet $\{1\}$. **[6]**

**Solution:** Let $A = \{w_1, w_2, \ldots, w_n\}$, $B = \{x_1, x_2, \ldots, x_n\}$ be an instance of Post's correspondence problem over unary alphabet $\{1\}$. For strings $w, x$ over $\{1\}$, $|w| = |x| \Leftrightarrow w = x$. If there is an $i$ such that $w_i = x_i$, then $i$ is a solution. If for each $i$, $|w_i| > |x_i|$, then there is no solution. Similarly there exists no solution if for each $i$, $|x_i| > |w_i|$. Assume for each $i \in [1, n]$, $w_i \neq x_i$. Choose $i$ such that $|w_i| > |x_i|$ and $j$ such that $|w_j| < |x_j|$. Let $a = |w_i| - |x_i|$ and $b = |x_j| - |w_j|$. Then the sequence $\underbrace{i, i, \ldots, i}_{b}, \underbrace{j, j, \ldots, j}_{a}$ is a solution to the instance $A, B$.

This is because repeating $w_i, x_i$, $b$ times will make the first string longer than the second by length $ab$. Now repeating $j$'th pair $w_j, x_j$ $i$ times places an excess of $ab$ many 1's in the second string thus making both strings of equal length.

6. Show that $\mathsf{TOTAL} = \{\mathcal{M} \mid \mathcal{M} \text{ is a total TM}\}$ is $\leq_m$-complete for $\Pi_2^0$. **[8]**

**Solution:** We can write

$$\mathsf{TOTAL} = \{\mathcal{M} \mid \forall x \ \exists t \ R(\mathcal{M}, x, t)\},$$

where

$$R(\mathcal{M}, x, t) = \begin{cases} 1 & \text{if } \mathcal{M} \text{ halts on } x \text{ in } t \text{ steps} \\ 0 & \text{otherwise} \end{cases}$$

$R$ is a decidable predicate since we can build a total (universal) machine that simulates $\mathcal{M}$ for $t$ steps on input $x$ and checks whether $\mathcal{M}$ halts or not. Hence, by the arithmetic hierarchy theorem $\in \Pi_2^0$.

We now show that $\mathsf{TOTAL}$ is $\Pi_2^0$-hard. Let $L \in \Pi_2^0$. Then $L$ can be written as $\{x \mid \forall y \exists z R(x, y, z)\}$ for some decidable ternary predicate $R$. Let $\mathcal{K}$ be a TM that decides $R$. We show that $L \leq_m \mathsf{TOTAL}$ by defining a map $f$ that takes a string $x$ to TM $\mathcal{M}$ such that $x \in L$ iff $\mathcal{M}$ is total. Given $x$, define a TM $\mathcal{M}$ that does the following on input $w$:

- Write down all strings $y$ of length at most $|w|$.
- For each $y$, find a $z$ such that $R(x, y, z)$ is true i.e., such that $\mathcal{K}$ accepts $x, y, z$. If each trial is successful, accept (or reject) $w$ and halt.

If $x \in L$, then for each $y$, $\exists z\, R(x, y, z)$. So, each trial in the second step will succeed and since there are finitely many $y$'s, $\mathcal{K}$ eventually halts on $w$. If $x \notin L$, $\exists y_0 \forall z\, \neg R(x, y_0, z)$. Suppose $|y_0| \leq |w|$, then $\mathcal{M}$ does not halt in the trial for $y_0$. If $|y_0| > |w|$, then $\mathcal{M}$ may halt on $w$. Summing up, we can conclude that $\mathcal{M}$ does not halt on all inputs and hence is not total. We have, $x \in L \Leftrightarrow \mathcal{M} \in \mathsf{TOTAL}$.

# Class Test 1

CS41001: Theory of Computation — 4th September, 2018
Time = 1 hour — Full marks = 30

Answer at least 3 questions. If you make any assumptions, state them upfront.

1. Show that the set $\{\mathcal{M} \mid \mathcal{M}$ is a DFA not accepting any string with odd number of 1's$\}$ is decideable. **[10]**

   **Solution:** We will use the fact that the emptiness problem for DFAs is decidable (just compute the set of states that are reachable from the start state and accept if any final state belongs to this set). In other words, there exists a TM $\mathcal{N}$ that decides $\{\mathcal{M} \mid \mathcal{M}$ is a DFA and $\mathcal{L}(\mathcal{M}) = \emptyset\}$. Let $L = \{\mathcal{M} \mid \mathcal{M}$ is a DFA not accepting any string with odd number of 1's$\}$. Define a set $A = \{x \in \{0,1\}^* \mid x$ contains an odd number of 1's$\}$ and let $\mathcal{M}_A$ denote a DFA recognising $A$.

   Define a TM $\mathcal{K}$ that, on input a DFA $\mathcal{M}$, does the following:

   - Construct a DFA $\mathcal{M}'$ that accepts $\mathcal{L}(\mathcal{M}) \cap \mathcal{L}(\mathcal{M}_A)$ (here, the description of $\mathcal{M}_A$ is hardwired in $\mathcal{N}$'s finite control).
   - Run $\mathcal{N}$ on input $\mathcal{M}'$.
   - Accept if $\mathcal{N}$ accepts; reject otherwise

   $\mathcal{K}$ accepts $L$ since $\mathcal{L}(\mathcal{M}) \cap A$ is empty iff $\mathcal{M}$ does not accept any string with odd number of 1's. Since both $A$ and $\mathcal{L}(\mathcal{M})$ are regular (the latter follows from the fact that $\mathcal{M}$ is a DFA) and regular sets are closed under intersection, the construction of $\mathcal{M}'$ accepting $\mathcal{L}(\mathcal{M}) \cap \mathcal{L}(\mathcal{M}_A)$ can be done in finite time. Hence $\mathcal{K}$ is total and decides $L$.

2. Show that $\{\mathcal{M} \mid \mathcal{L}(\mathcal{M})$ halts on all inputs of length less than 300$\}$ is recursively enumerable but its complement is not. **[10]**

   **Solution:** Denote the language by $\mathsf{T}_{300}$. Define a TM $\mathcal{N}$, that on input a TM $\mathcal{M}$, does the following:

   - For each string $w$ with $|w| < 300$, run $\mathcal{M}$ on $w$
   - If $\mathcal{M}$ halts on all such strings $w$, then accept and halt

   If $\mathcal{M} \in \mathsf{T}_{300}$, then it halts on all string of length less than 300 and so $\mathcal{N}$ accepts $\mathcal{M}$. Otherwise, $\mathcal{M}$ loops on some string $w$ thus making $\mathcal{N}$ loop on $\mathcal{M}$. Hence $\mathcal{L}(\mathcal{N}) = \mathsf{T}_{300}$ establishing that $\mathsf{T}_{300}$ is recursively enumerable.

   To show that $\neg\mathsf{T}_{300}$ is not r.e., we provide a reduction from $\neg\mathsf{HP}$. Let $(\mathcal{M}, x)$ be an instance of $\neg\mathsf{HP}$. Define a TM $\mathcal{N}$, that does the following on input $y$:

   - If $|w| >= 300$, then reject and halt (or accept or enter a trivial loop).

- Otherwise, run $cM$ on $x$.
- Accept and halt if $\mathcal{M}$ halts on $x$.

If $\mathcal{M}$ does not halt on $x$, then $\mathcal{N}$ loops on all inputs of length $< 300$. If $\mathcal{M}$ halts on $x$, then $\mathcal{N}$ halts on all inputs of length $< 300$. That is, $(\mathcal{M}, x) \in \neg\mathsf{HP} \Leftrightarrow \mathcal{N} \in \neg\mathsf{T}_{300}$.

3. Let $G, H$ denote context-free grammars over $\{0, 1\}$. Prove that it is undecidable whether

   (a) $\mathcal{L}(G) = \mathcal{L}(H)$. **[5]**

   **Solution:** Let $\mathsf{E} = \{(G, H) | \mathcal{L}(G) = \mathcal{L}(H)\}$. We show that $\neg\mathsf{HP} \leq_m \mathsf{E}$. Given an instance $(\mathcal{M}, x)$ of $\neg\mathsf{HP}$, define $G$ such that $\mathcal{L}(G) = \neg\mathsf{VALCOMPS}_{\mathcal{M},x}$ and $H = (S, \Delta, P, S)$ where $\Delta$ is the alphabet of $\mathsf{VALCOMPS}_{\mathcal{M},x}$ and $P$ consists of productions $S \to a_i S \mid \epsilon$ for each $a_i \in \Delta$ ($\epsilon$ is the empty string). Clearly $H$ generates all strings over $\Delta$ i.e.$\mathcal{L}(H) = \Delta^*$. If $(\mathcal{M}, x) \in \neg\mathsf{HP}$, then there are no valid computation histories i.e., $\neg\mathsf{VALCOMPS}_{\mathcal{M},x} = \Delta^*$ and hence $\mathcal{L}(G) = \mathcal{L}(H)$. If $(\mathcal{M}, x) \notin \neg\mathsf{HP}$, then $\neg\mathsf{VALCOMPS}_{\mathcal{M},x} \neq \Delta^*$ whence $\mathcal{L}(G) \neq \mathcal{L}(H)$. Thus $\mathsf{E}$ is undecidable.

   (b) $\mathcal{L}(G) = \mathcal{L}(G)\mathcal{L}(G)$. **[5]**

   **Solution:** Let $\mathsf{CC} = \{G \mid \mathcal{L}(G) = \mathcal{L}(G)\mathcal{L}(G)\}$ (where the double 'C' stands for closed under concatenation). We show that $\neg\mathsf{HP} \leq_m \mathsf{CC}$. Given an instance $(\mathcal{M}, x)$ of $\neg\mathsf{HP}$, define $G$ such that $\mathcal{L}(G) = \neg\mathsf{VALCOMPS}_{\mathcal{M},x}$. Let $\Delta$ be the alphabet of $\mathsf{VALCOMPS}_{\mathcal{M},x}$. If $(\mathcal{M}, x) \in \neg\mathsf{HP}$, then there are no valid computation histories i.e., $\neg\mathsf{VALCOMPS}_{\mathcal{M},x} = \Delta^*$ and hence $\mathcal{L}(G) = \mathcal{L}(G)\mathcal{L}(G) = \Delta^*$. If $(\mathcal{M}, x) \notin \neg\mathsf{HP}$, then $\neg\mathsf{VALCOMPS}_{\mathcal{M},x} \neq \Delta^*$ but $\mathcal{L}(G)\mathcal{L}(G) = \Delta^*$ (this follows from the fact that you can always split a valid computation history into two parts that are themselves not valid histories) whence $\mathcal{L}(G) \neq \mathcal{L}(G)\mathcal{L}(G)$. Therefore $\mathsf{CC}$ is undecidable.

4. For a set $A$, define $A^{\mathbf{R}} = \{w^{\mathbf{R}} \mid w \in A\}$ where $w^{\mathbf{R}}$ denotes the reverse of the string $w$. Is it decidable for a given TM $\mathcal{M}$ whether $\mathcal{L}(\mathcal{M}) = \mathcal{L}(\mathcal{M})^{\mathbf{R}}$? Justify. **[10]**

   **Solution:** Let $\mathsf{REV} = \{\mathcal{M} \mid \mathcal{L}(\mathcal{M}) = \mathcal{L}(\mathcal{M})^{\mathbf{R}}\}$. We know that $\mathsf{REV}$ is not recursive (undecidable) iff its complement is not, for otherwise by the fact that recursive sets are closed under complementation, both would be recursive. We show that $\neg\mathsf{REV}$ is undecidable via a reduction from $\mathsf{HP}$. Let $(\mathcal{M}, x)$ be an instance of $\mathsf{HP}$. Construct a TM $\mathcal{N}$ over $\Sigma$ with $|\Sigma| > 1$ that on input $y$, does the following.

   - Run $\mathcal{M}$ on $x$.
   - If $\mathcal{M}$ halts and $y = a_1 a_2$, then accept and halt. (Here $a_1, a_2 \in \Sigma$ and $a_1 \neq a_2$).
   - Reject otherwise.

   We choose $\Sigma$ of size $> 1$ for otherwise the problem is trivially decidable. Every language over the unary alphabet is closed under reversal. Now, if $\mathcal{M}$ does not halt on $x$, then $\mathcal{L}(\mathcal{N}) = \emptyset$ and trivially $\mathcal{L}(\mathcal{N}) = \mathcal{L}(\mathcal{N})^{\mathbf{R}}$. Otherwise, $\mathcal{L}(\mathcal{M}) = \{a_1 a_2\}$. Since $(a_1 a_2)^{\mathbf{R}} = a_2 a_1 \notin \mathcal{L}(\mathcal{N})$, $\mathcal{L}(\mathcal{N}) \neq \mathcal{L}(\mathcal{N})^{\mathbf{R}}$. Hence $\mathsf{REV}$ is undecidable.

**Alternate solution using Rice's theorem.** Let $P$ be a property on r.e.sets defined as

$$P(A) = \begin{cases} T & \text{if } A = A^{\mathbf{R}} \\ F & \text{otherwise} \end{cases}$$

Again, we consider languages over $\Sigma$ of size $> 1$ for otherwise the problem is trivially decidable. Trivially, $P(\emptyset) = T$. Also $P(\{a_1a_2\}) = F$ for some set $a_1, a_2 \in \Sigma$ with $a_1 \neq a_2$ since $(a_1a_2)^{\mathbf{R}} = a_2a_1 \notin \mathcal{L}(\mathcal{N})$. We have exhibited two sets, one for which the $P$ holds and the other for which it does not. It follows that $P$ is a non-trivial property and hence undecidable, by Rice's theorem.

5. Prove that the emptiness problem for linearly bounded automata (LBA) i.e., the set $\{\mathcal{M} \mid \mathcal{M}$ is an LBA and $\mathcal{L}(\mathcal{M}) = \emptyset\}$ is undecidable. **[10]**

**Solution:** We prove $\mathsf{E\text{-}LBA} = \{\mathcal{N} | \mathcal{N}$ is an LBA and $\mathcal{L}(\mathcal{N}) = \emptyset\}$ is undecidable via a reduction from $\mathsf{MP}$, the membership problem for TMs. Let $(\mathcal{M}, x)$ be an instance of $\mathsf{MP}$. We provide a construction of an LBA $\mathcal{N}$ that accepts $\mathsf{VALCOMPS}_{\mathcal{M},x}$. Now, $\mathcal{M}$ accepts $x$ implies $\mathsf{VALCOMPS}_{\mathcal{M},x} \neq \emptyset$ and otherwise $\mathsf{VALCOMPS}_{\mathcal{M},x} = \emptyset$. Put differently, $(\mathcal{M}, x) \in \mathsf{MP} \Leftrightarrow \mathcal{N} \in \mathsf{E\text{-}LBA}$. Since $\mathsf{MP}$ is undecidable, so is $\mathsf{E\text{-}LBA}$. All that remains to be shown is that $\mathcal{N}$ can be constructed in finite time.

The conditions for a string $z$ to be in $\mathsf{VALCOMPS}_{\mathcal{M},x}$ are:

- $z$ is of the form $\#\alpha_0\#\alpha_1\#\cdots\#\alpha_n\#$ where for each $i \in [0, n]$, $\alpha_i \in (\Delta \setminus \{\#\})^*$. Here, $\Delta$ is the alphabet for $\mathsf{VALCOMPS}_{\mathcal{M},x}$.
- Each $\alpha_i$ is a string of symbols of the form $\begin{smallmatrix} a \\ - \end{smallmatrix}$ or $\begin{smallmatrix} a \\ q \end{smallmatrix}$, with exactly one symbol containing an element of $Q$ at the bottom and the rest containing $-$ at the bottom.
- $\alpha_0$ is the starting configuration of $\mathcal{M}$ on $x$.
- Either $t$ (accept state) or $r$ (reject state) appear somewhere in $z$.
- $\alpha_i \xrightarrow[\mathcal{M}]{1} \alpha_{i+1}$ for each $i \in [0, n-1]$.

If $A_j = \{z \in \Delta^* \mid z$ satisfies condition $j\}$ for $j = 1, 2, 3, 4, 5$, then $\mathsf{VALCOMPS}_{\mathcal{M},x} = \cap_{i=1}^{5} A_i$. Conditions $A_1, \ldots, A_4$ can be checked by a DFA and so can be checked by an LBA. Design $\mathcal{N}$ to first check conditions $A_1, \ldots, A_4$. The last condition can be checked by an LBA since it can move the tape head backwards. That is, for each $i \in [0, n-1]$, the LBA checks that $\alpha_i \xrightarrow[\mathcal{M}]{1} \alpha_{i+1}$ by moving back and forth, ensuring they differ in at most 3 positions and that the symbols in these positions agree with the transition function of $\mathcal{M}$. The transition function of $\mathcal{M}$ is hardwired in $\mathcal{N}$'s finite control. Clearly, the description of the LBA $\mathcal{N}$ can be written down in finite time.

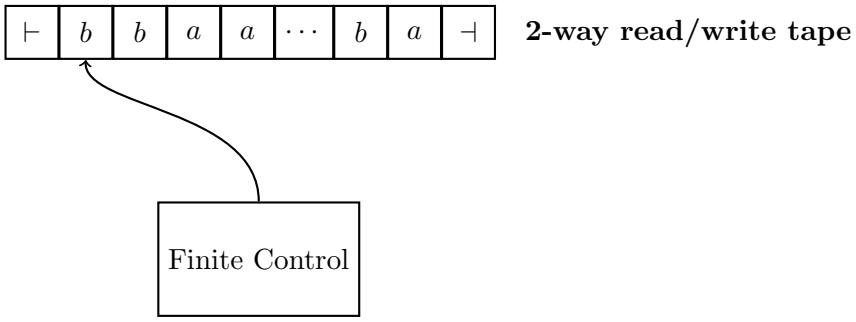
# Tutorial 1

## Turing Machines and their Variations

**Guidelines:** Solve all problems in the class. *Do not search for solutions online. You are allowed to refer to your notes and discuss among other students and TAs.*

1. Recall the definition of *counter automata.* A $k$-counter automaton is a machine equipped with a 2-way *read-only* input tape and $k$ integer counters, each of which can store an arbitrary negative integer and can be incremented, decremented or tested for 0 independently. Show how to simulate a stack using a two counters (equivalently, show how to simulate a deterministic PDA using a 2-counter automaton). Argue that counter automata can simulate Turing machines.

2. A *linear bounded automaton* (LBA) is exactly like a 1-tape TM, except that the input string $x \in \Sigma^*$ is enclosed in left and right endmarkers $\vdash$ and $\dashv$ which may not be overwritten. The machine is constrained never to move left of $\vdash$ or right of $\dashv$. It is allowed to read/write between these markers.




   (a) Give a rigorous formal definition of deterministic linearly bounded automata, including a definition of configurations and acceptance.

   **Solution:** An LBA is given by a 10-tuple, $(Q, \Sigma, \Gamma, \delta, \vdash, \dashv, \sqcup, s, t, r)$ where

   - $Q$ is the set of states
   - $\Sigma$ is the input alphabet
   - $\Gamma$ is the tape alphabet
   - $\delta : Q \times \Gamma \to Q \times \Gamma \times \{L, R\}$ is the transition function
   - $\vdash, \dashv \in \Gamma \setminus \Sigma$ are the left and right endmarkers respectively
   - $\sqcup \in \Gamma \setminus \Sigma$ is the blank symbol
   - $s, t, r$ are respectively the start, accept and reject states

   Restrictions that ensure that the Turing machine never overwrites the endmarkers and never move left or right of the left and right endmarkers are given below.

   - $\forall p \in Q$, $\exists q \in Q$ such that $\delta(p, \vdash) = (q, \vdash, R)$.
   - $\forall p \in Q$, $\exists q \in Q$ such that $\delta(p, \dashv) = (q, \dashv, L)$.

   □

(b) Let $\mathcal{M}$ be an LBA with state set $Q$ of size $k$ and tape alphabet $\Gamma$ of size $m$. How many possible configurations are there on input $x$ with $|x| = n$?

**Solution:** Let $[n_1, n_2]$ for $n_2 > n_1$ denote the set of integers $n_1, n_1 + 1, \ldots, n_2$. There are a total of $m$ possible symbols that can be written on tape cells $1, \ldots, n$. At positions 0 and $n + 1$, the symbols are fixed. The LBA can be in $k$ possible states. The tape head can be at one of $n + 2$ positions. Hence, the set of all possible configurations is given by $[0, n + 2] \times Q \times \Gamma^n$. The number of configurations possible is therefore $(n + 2) \cdot k \cdot m^n$.

(c) Argue that it is possible to detect in finite time whether an LBA loops on a given input. **Hint:** Use part (b).

**Solution:** Run the LBA for $k(n + 2)m^n + 1$ steps. If any configuration repeats, then output that the LBA loops. Correctness follows from the fact that the LBA is deterministic.

3. Show how to build a TM that inputs a string of the form $0^n$ and halts with $0^{n^2}$ written on its tape. First provide an informal description and then formally define the TM.

**Solution:** Informally, the TM copies the string $n$ times to the right. Initially the tape contents are $\vdash 0^n \sqcup \ldots$. Put a special marker, say $X$ at the end of the input string. First copy $0^n$ after the $X$ symbol. The contents will now be $\vdash 0^n X 0^n \sqcup \ldots$. Repeatedly copy the string $0^n$ at the right end replacing the blanks, each time marking the 0's in the beginning with a hat. The sequence of tape contents will be as follows.

$\vdash 0^n X 0^n \sqcup \ldots$
$\vdash \hat{0} 0^{n-1} X 0^n 0^n \sqcup \ldots$
$\vdash \hat{0}^2 0^{n-2} X 0^n 0^n 0^n \sqcup \ldots$
$\ldots$
$\vdash \hat{0}^n X \underbrace{0^n 0^n \ldots 0^n}_{n} \sqcup \ldots$

At the end, you may replace the $\hat{0}$'s with blanks. Then the tape will contain exactly $n^2$ 0's.

The copying $0^n$ to the right can be done as follows: scan from the left endmarker till the $X$ symbol each time a symbol is read (either 0 or $\hat{0}$), write a 0 at the end, that is after the last non-blank symbol. The total number of 0's and $\hat{0}$'s until $X$ will be exactly $n$.

# Tutorial 2

## Recursive and Recursively Enumerable Sets, (Un)Decidability

**Guidelines:** Solve all problems in the class. *Do not search for solutions online. You are allowed to refer to your notes and discuss among other students and TAs.*

1. Show that recursive sets are closed under complement.

   **Solution:** Let $A$ be a recursive set. Then there exists a total TM $\mathcal{M}$ deciding $A$. Construct a Turing machine $\mathcal{M}'$ that, on input $y$, does the following.

   - Simulates $\mathcal{M}$ on input $y$.
   - Accept if $\mathcal{M}$ rejects.
   - Reject if $\mathcal{M}$ accepts.

   Since $\mathcal{M}$ is total, it halts on all inputs; so does $\mathcal{M}'$. If $y \in \neg A$, then $y \notin A$. As a result, $\mathcal{M}$ rejects $y$ and hence $\mathcal{M}'$ accepts it. If $y \notin \neg A$, then $y \in A$ and $\mathcal{M}$ accepts causing $\mathcal{M}'$ to reject $y$. Therefore $\neg A$ is recursive. □

2. Prove that if $A$ and $\neg A$ are recursively enumerable, then $A$ is recursive.

   **Solution:** Let $\mathcal{M}_1, \mathcal{M}_2$ be TMs deciding $A, \neg A$ respectively. We show $A$ to be recursive by building a total TM $\mathcal{N}$ that decides membership in $A$. On input $y$, $\mathcal{N}$ does the following.

   - Copy $y$ on two separate tracks of the tape.
   - Simulate $\mathcal{M}_1$ and $\mathcal{M}_2$ on $y$ on the two tracks simultaneously.
   - If $\mathcal{M}_1$ accepts, then halt and accept.
   - If $\mathcal{M}_2$ accepts, then halt and reject.

   If $y \in A$, $\mathcal{M}_1$ eventually halts and accepts. Similary, if $y \notin A$, $\mathcal{M}_2$ eventually halts and accepts. Hence $\mathcal{N}$ halts on all inputs and accepts $y$ iff $y \in A$. Therefore $A$ is recursive.

3. It is undecidable whether two given TMs accept the same set. Justify this statement.

   **Solution:** The problem is to show that the language $\mathsf{EQUIV} = \{\mathcal{M}_1 \# \mathcal{M}_2 | \mathcal{L}(\mathcal{M}_1) = \mathcal{L}(\mathcal{M}_2)\}$ is undecidable. We show a reduction from the halting problem i.e., $\mathsf{HP} \leq_m \mathsf{EQUIV}$. Given an instance $\mathcal{M}\#x$ of $\mathsf{HP}$, construct an instance $\mathcal{M}_1\#\mathcal{M}_2$ such that $\mathcal{M}_1$ and $\mathcal{M}_2$ have the following descriptions.

   $\mathcal{M}_1$: on input $y$ does the following

   - Erase the input $y$.
   - Accept and halt.

   $\mathcal{M}_2$: on input $y$ does the following

   - Erase the input $y$.

- Run $\mathcal{M}$ on $x$.
- Accept and halt if $\mathcal{M}$ halts on $x$.

Clearly, $\mathcal{L}(\mathcal{M}_1) = \Sigma^*$ and

$$\mathcal{L}(\mathcal{M}_2) = \begin{cases} \Sigma^* & \text{if } \mathcal{M} \text{ halts on } x \\ \emptyset & \text{otherwise} \end{cases}$$

That is $\mathcal{M}\#x \in \mathsf{HP}$ iff $\mathcal{M}_1\#\mathcal{M}_2 \in \mathsf{EQUIV}$. The descriptions of $\mathcal{M}_1, \mathcal{M}_2$ can be written down by a total TM. (Note that writing down descriptions of these TMs does not require running $\mathcal{M}$.) Since $\mathsf{HP}$ is undecidable, so is $\mathsf{EQUIV}$.

# Tutorial 3
## (Un)Decidability, Rice's Theorem

**Guidelines:** Solve all problems in the class. *Do not search for solutions online. You are allowed to refer to your notes and discuss among other students and TAs.*

1. Recall the definition of linear bounded automata (LBA) from Tutorial 2. Prove by diagonalisation that there exists a recursive set that is not accepted by any LBA.

   **Solution:** Similar to TMs, we can define an encoding using bit strings for LBAs. Let $\mathcal{M}_\varepsilon, \mathcal{M}_0, \mathcal{M}_1, \mathcal{M}_{00}, \mathcal{M}_{01}, \ldots$ denote LBA's described according to this encoding. If a string $x$ is not a valid description of an LBA, set $\mathcal{M}_x$ to be a trivial LBA that accepts and halts on all inputs. Construct an infinite matrix $\mathbf{A} = (a_{x,y})_{x,y=\varepsilon,0,1,00,01,\ldots}$ as

   $$a_{x,y} = \begin{cases} 1 & \text{if } y \in \mathcal{L}(\mathcal{M}_x) \\ 0 & \text{if } y \notin \mathcal{L}(\mathcal{M}_x) \end{cases}$$

   Now consider a set $B$ defined as $B = \{x | x \notin \mathcal{L}(\mathcal{M}_x)\}$. The characteristic function for this set corresponds to the complement of the diagonal of $\mathbf{A}$. As for each $x \in \{0,1\}^*$, $B$ differs from $\mathcal{L}(\mathcal{M}_x)$ atleast on $x$, no LBA accepts $B$. We now prove that $B$ is recursive by building a total TM $\mathcal{N}$ accepting $B$. Below is the description of $\mathcal{N}$.

   $\mathcal{N}$: on input $y$

   - Construct $\mathcal{M}_y$ from $y$.
   - Simulate $\mathcal{M}_y$ on input $y$.
   - If $\mathcal{M}_y$ accepts and halts, then reject and halt.
   - If $\mathcal{M}_y$ rejects or loops, then accept and halt.

   Using the fact that it is possible to check whether an LBA loops or not in finite time, we conclude that the TM $\mathcal{N}$ halts on all inputs and is hence total.

   **Note:** Here I have indexed the automata with strings. You can use the natural numbers as index and refer to the LBA corresponding to a string $x$ as $\mathcal{M}_{\eta(x)}$ where $\eta : \{0,1\}^* \to \mathbb{N}$ is an injective map. Such a map exists since $\{0,1\}^*$ is a countable set. □

2. One of the following sets is r.e. and the other is not. Determine which one is and which one is not. Justify your answers.

   (a) $\{\mathcal{M} \mid \mathcal{L}(\mathcal{M})$ contains atleast 300 elements$\}$

   **Solution:** This set is r.e. It is possible to construct a TM that runs a given $\mathcal{M}$ all strings. If atleast 300 strings are accepted by $\mathcal{M}$, then halt and accept. Otherwise keep testing for other stings.

   (b) $\{\mathcal{M} \mid \mathcal{L}(\mathcal{M})$ contains atmost 300 elements$\}$

   **Solution:** This is a non-monotone property of r.e. sets. If an r.e. set $A$ contains atmost 300 elements, then it is not necessary that all its supersets contain atmost 300 elements. Hence, it follows by Rice's theorem that the given set is not r.e.

3. Let $\mathsf{REG} = \{\mathcal{M} \mid \mathcal{L}(\mathcal{M}) \text{ is a regular set}\}$. Show that neither $\mathsf{REG}$ nor $\neg\mathsf{REG}$ is recursively enumerable.

**Solution:** Let $P_{\mathsf{REG}}$ denote the property on r.e. sets defined as

$$P_{\mathsf{REG}}(A) = \begin{cases} \mathsf{T} & \text{if } A \text{ is regular} \\ \mathsf{F} & \text{otherwise} \end{cases}$$

Then deciding this property is equivalent to deciding $\mathsf{REG}$. If a set $A$ is regular it is not necessary that all its supersets are regular. In other words, there exist $A, B$ such that $A \subseteq B$ and $P_{\mathsf{REG}}(A) = \mathsf{T}$, $P_{\mathsf{REG}}(B) = \mathsf{F}$. For instance, we can take $A = \phi$ and $B = \{0^n1^n \mid n \geq 0\}$. This shows that $P_{\mathsf{REG}}$ is a non-monotone property. By Rice's theorem, $P_{\mathsf{REG}}$ is not r.e. or equivalently $\mathsf{REG}$ is not r.e. Similarly, we can show that $\neg\mathsf{REG}$ or equivalently,

$$P_{\neg\mathsf{REG}}(A) = \begin{cases} \mathsf{T} & \text{if } A \text{ is not regular} \\ \mathsf{F} & \text{otherwise} \end{cases}$$

is not r.e. by proving that $P_{\neg\mathsf{REG}}$ is a non-monotone property. We only need to exhibit two sets $A, B$ with $A \subseteq B$ such that $P_{\neg\mathsf{REG}}(A) = \mathsf{T}$ and $P_{\neg\mathsf{REG}}(B) = \mathsf{F}$. Taking $A = \{0^n1^n \mid n \geq 0\}$ and $B = \{0^*1^*\}$ suffices.

4. Let $\mathsf{TOT} = \{\mathcal{M} \mid \mathcal{M} \text{ halts on all inputs}\}$. Show that neither $\mathsf{TOT}$ nor its complement is r.e. is recursively enumerable.

**Solution:** We show $\mathsf{TOT}$ is not r.e. via a reduction from $\neg\mathsf{HP}$. Given an instance $\mathcal{M}\#x$ of $\neg\mathsf{HP}$, we construct a TM $\mathcal{N}$ such that $\mathcal{M}\#x \in \neg\mathsf{HP}$ iff $\mathcal{N} \in \mathsf{TOT}$. Below is a description of $\mathcal{N}$.

$\mathcal{N}$: on input $y$ does the following.

- Store $y$ on a separate track of the tape.
- Run $\mathcal{M}$ on $x$ for $|y|$ steps. ($\mathcal{M}$ and $x$ are hardwired in $\mathcal{N}$'s finite control.)
- If $\mathcal{M}$ does not halt within $|y|$ steps, then halt and either accept or reject.
- Otherwise, enter a trivial loop.

If $\mathcal{M}$ does not halt on $x$, then $\mathcal{N}$ halts on all input strings. Suppose that $\mathcal{M}$ halts on $x$ in $n$ steps. Then for any string $y$ with $|y| < n$, $\mathcal{N}$ accepts $y$ since $\mathcal{M}$ does not halt on $x$ within $|y|$ steps. On the other hand, for any string $y$ with $|y| \geq n$, $\mathcal{N}$ does not halt. That is,

$$\mathcal{M} \text{ does not halt on } x \implies \mathcal{N} \text{ halts on all input strings} \implies \mathcal{N} \in \mathsf{TOT}$$

$$\mathcal{M} \text{ halts on } x \implies \mathcal{N} \text{ does not halt on some input strings} \implies \mathcal{N} \notin \mathsf{TOT}$$

Since $\neg\mathsf{HP}$ is not r.e., $\mathsf{TOT}$ is not r.e.

Next, we prove that $\neg\mathsf{TOT}$ is not r.e. through a reduction from $\neg\mathsf{HP}$. Given an instance $\mathcal{M}\#x$ of $\neg\mathsf{HP}$, we construct a TM $\mathcal{N}$ such that $\mathcal{M}\#x \in \neg\mathsf{HP}$ iff $\mathcal{N} \in \neg\mathsf{TOT}$. Below is a description of $\mathcal{N}$.

$\mathcal{N}$: on input $y$ does the following.

- Store $y$ on a separate track of the tape.
- If $|y| \leq |x|$, halt and accept.
- Run $\mathcal{M}$ on $x$.
- If $\mathcal{M}$ halts on $y$, then halt and accept.

If $\mathcal{M}$ does not halt on $x$, then $\mathcal{N}$ halts on all input strings of length at most $|x|$. If $\mathcal{M}$ halts on $x$, then $\mathcal{N}$ accepts any string $y$. That is,

$$\mathcal{M} \text{ does not halt on } x \implies \mathcal{N} \text{ does not halt on some inputs} \implies \mathcal{N} \in \neg\mathsf{TOT}$$

$$\mathcal{M} \text{ halts on } x \implies \mathcal{N} \text{ halts on all input strings} \implies \mathcal{N} \notin \neg\mathsf{TOT}$$

Since $\neg\mathsf{HP}$ is not r.e., $\neg\mathsf{TOT}$ is not r.e.

# Tutorial 4

## Undecidable problems on Context-Free Languages

**Guidelines:** Solve all problems in the class. *Do not search for solutions online. You are allowed to refer to your notes and discuss among other students and TAs.*

1. Define a set $\mathsf{VALC\text{-}R}_{\mathcal{M},x}$ to be the set of all strings of the form $\#\alpha_0\#\alpha_1^{\mathbf{R}}\#\alpha_2\#\alpha_3^{\mathbf{R}}\#\cdots\#\alpha'_N\#$, where $\alpha'_N = \alpha_N^{\mathbf{R}}$ if $N$ is odd and $\alpha'_N = \alpha_N$ otherwise, such that $\#\alpha_0\#\alpha_1\#\cdots\#\alpha_N\#$ is a valid computation history of $\mathcal{M}$ on input $x$. That is

$$\#\alpha_0\#\alpha_1\#\cdots\#\alpha_N\# \in \mathsf{VALCOMPS}_{\mathcal{M},x} \Leftrightarrow \#\alpha_0\#\alpha_1^{\mathbf{R}}\#\alpha_2\#\alpha_3^{\mathbf{R}}\#\cdots\#\alpha'_N\# \in \mathsf{VALC\text{-}R}_{\mathcal{M},x}.$$

Show that $\mathsf{VALC\text{-}R}_{\mathcal{M},x}$ can be expressed as the intersection of two context-free languages.
**Hint:** Design two non-deterministic pushdown automata – one that checks whether $\alpha_i^{\mathbf{R}} \xrightarrow[\mathcal{M}]{1} \alpha_{i+1}$ for odd $i$ and another that checks if $\alpha_i \xrightarrow[\mathcal{M}]{1} \alpha_{i+1}^{\mathbf{R}}$ for even $i$. Argue that the intersection of the languages accepted by these two NPDAs is precisely $\mathsf{VALC\text{-}R}_{\mathcal{M},x}$.

**Solution:** Let

$$L_1 = \{\#\alpha_0\#\alpha_1\#\alpha_2\#\alpha_3\#\cdots\#\alpha_N\# \mid \alpha_i^{\mathbf{R}} \xrightarrow[\mathcal{M}]{1} \alpha_{i+1} \text{ for odd } i\}$$

and

$$L_2 = \{\#\alpha_0\#\alpha_1\#\alpha_2\#\alpha_3\#\cdots\#\alpha_N\# \mid \alpha_i \xrightarrow[\mathcal{M}]{1} \alpha_{i+1}^{\mathbf{R}} \text{ for even } i\}.$$

Clearly, $L_1 \cap L_2$ consists of strings of the form $\#\alpha_0\#\alpha_1\#\alpha_2\#\alpha_3\#\cdots\#\alpha_N\#$ with both conditions $\alpha_i^{\mathbf{R}} \xrightarrow[\mathcal{M}]{1} \alpha_{i+1}$ for odd $i$ and $\alpha_i^{\mathbf{R}} \xrightarrow[\mathcal{M}]{1} \alpha_{i+1}$ for even $i$, being satisfied. Equivalently it consists of strings $\#\alpha_0\#\alpha_1\#\alpha_2\#\alpha_3\#\cdots\#\alpha_N\#$ such that $\#\alpha_0\#\alpha_1^{\mathbf{R}}\#\alpha_2\#\alpha_3^{\mathbf{R}}\#\cdots\#\alpha'_N\# \in \mathsf{VALCOMPS}_{\mathcal{M},x}$ implying that $L_1 \cap L_2 = \mathsf{VALC\text{-}R}_{\mathcal{M},x}$. We now build NPDAs $\mathcal{N}_1, \mathcal{N}_2$ accepting $L_1, L_2$ respectively, thus showing that $L_1, L_2$ are CFLs.

$\mathcal{N}_1$, on input a string $\#\alpha_0\#\alpha_1\#\alpha_2\#\alpha_3\#\cdots\#\alpha_N\#$ repeats the following until the end of string is reached.

- Go to the next odd position $i$. Read the input string until the # symbol just before $\alpha_i$.
- $\alpha_i$ is of the form $\bar{\alpha}_i^{\mathbf{R}}$ for some TM configuration $\bar{\alpha}_i$.
- While reading $\alpha_i$, examine $\delta$ to determine the next configuration of $\mathcal{M}$ after $\alpha_i$. (This can be done using constant amount of memory in the finite control as $\alpha_i$ would differ from its next configuration in atmost three positions. Moreover, $\mathcal{M}$ is deterministic.) Push this configuration $\bar{\alpha}_{i+1}^{\mathbf{R}}$ on the stack.
- Now pop the stack one symbol at a time while reading the next symbol in the input to check whether $\alpha_{i+1} = \bar{\alpha}_{i+1}$. item Reject if there is a mismatch. Accept once the stack becomes empty.

The construction of $\mathcal{N}_2$ is similar. □

2. Prove that it is undecidable whether
   (a) the intersection of two given CFLs is empty.
   **Hint:** Think VALCOMPS or VALC-R.

**Solution:** Let $\mathsf{EI\text{-}CFL} = \{L_1, L_2 \mid L_1, L_2 \text{ are CFLs and } L_1 \cap L_2 = \emptyset\}$. We show a reduction from $\neg\mathsf{HP}$ to $\mathsf{EI\text{-}CFL}$. Let $\mathcal{M}, x$ be an instance of $\neg\mathsf{HP}$. Construct CFLs $L_1, L_2$ such that $L_1 \cap L_2 = \mathsf{VALC\text{-}R}_{\mathcal{M},x}$. If $\mathcal{M}$ does not halt on $x$, then $\mathsf{VALC\text{-}R}_{\mathcal{M},x} = \emptyset$ and hence $L_1 \cap L_2 = \emptyset$. If $\mathcal{M}$ halts on $x$, then $\mathsf{VALC\text{-}R}_{\mathcal{M},x} = L_1 \cap L_2 \neq \emptyset$. Since $\neg\mathsf{HP}$ is undecidable, $\mathsf{EI\text{-}CFL}$ is also undecidable.

**Note:** I have cheated a bit here and described the set $\mathsf{EI\text{-}CFL}$ as $\{L_1, L_2 | L_1, L_2$ are CFLs and $L_1 \cap L_2 = \emptyset\}$ assuming there are finite representations of the context-free languages. It is more appropriate to write $\mathsf{EI\text{-}CFL}$ as $\{\mathcal{M}_1, \mathcal{M}_2 \mid \mathcal{M}_1, \mathcal{M}_2$ are NPDAs and $\mathcal{L}(\mathcal{M}_1) \cap \mathcal{L}(\mathcal{M}_2) = \emptyset\}$ or define the set in terms context-free grammars generating the languages.

(b) the intersection of two given CFLs is a CFL.

**Solution:** The same reduction as in the previous problem works since $\emptyset$ is a CFL and $\mathsf{VALCOMPS}$ is not a CFL.

(c) the complement of a given CFL is a CFL.

**Solution:** We reduce $\neg\mathsf{HP}$ to $\{\mathcal{L} \mid \neg\mathcal{L}$ is a CFL$\}$. Given $\mathcal{M}, x$, construct a CFG $G$ such that $\mathcal{L}(G) = \neg\mathsf{VALCOMPS}_{\mathcal{M},x}$. If $\mathcal{M}$ does not halt on $x$, then $\mathcal{L}(G) = \Sigma^*$ and $\neg\mathcal{L}(G) = \emptyset$ both of which are CFLs. If $\mathcal{M}$ halts on $x$, then $\mathcal{L}(G)$ is a CFL but its complement $\neg\mathcal{L}(G) = \mathsf{VALCOMPS}_{\mathcal{M},x}$ is not.

# Tutorial 5
## PCP and Recursion Theorem

**Guidelines:** Solve all problems in the class. *Do not search for solutions online. You are allowed to refer to your notes and discuss among other students and TAs.*

1. Consider a *silly* variant of PCP called SPCP where corresponding strings in both lists are restricted to have the same length. Show that this variant is decidable.

   **Solution:** Let $A = \{w_1, \ldots, w_n\}$ and $B = \{x_1, \ldots, x_n\}$ denote an instance of SPCP. If there exists a solution, then there is an index $j \in [1, n]$ such that the solution starts with $w_j, x_j$. Let $|w_j| = |x_j| = \ell$. Since there is a match in the first $\ell$ positions, it must be the case that $w_j = x_j$. Also, if there is an index $j$ such that $w_j = x_j$, then $j$ is a solution to the SPCP instance $(A, B)$. Therefore, SPCP can be decided by just checking whether for each $j \in [1, n]$, $w_j = x_j$.

2. Prove that $\{G_1, G_2 \mid G_1, G_2 \text{ are CFGs and } \mathcal{L}(G_1) \cap \mathcal{L}(G_2) \neq \emptyset\}$ is undecidable via a reduction from PCP.

   **Solution:** Let $A = \{w_1, \ldots, w_n\}$ and $B = \{x_1, \ldots, x_n\}$ denote an instance of PCP over alphabet $\Sigma$. Let $\Gamma = \Sigma \cup \{a_1, \ldots, a_n\}$ for some new symbol $a_1, \ldots, a_n \notin \Sigma$. Define two CFGs $G_1 = (\{S_1\}, \Gamma, P_1, S_1)$ and $G_2 = (\{S_2\}, \Gamma, P_2, S_2)$ where $P_1$ consists of the productions

   $$S_1 \to w_1 S_1 a_1 | w_2 S_2 a_2 | \cdots | w_n S_1 a_n | w_1 a_1 | w_2 a_2 | \cdots | w_n a_n,$$

   and $P_2$ consists of

   $$S_2 \to x_1 S_2 a_1 | x_2 S_2 a_2 | \cdots | x_n S_2 a_n | x_1 a_1 | x_2 a_2 | \cdots | x_n a_n.$$

   Suppose the PCP instance $(A, B)$ has a solution $i_1, \ldots, i_k$. Let $y = w_{i_1} \cdots w_{i_k} = x_{i_1} \cdots x_{i_k}$. Then $y a_{i_k} \cdots a_{i_1} \in \mathcal{L}(G_1)$ and $y a_{i_k} \cdots a_{i_1} \in \mathcal{L}(G_2)$. As a result $\mathcal{L}(G_1) \cap \mathcal{L}(G_2) \neq \emptyset$.

   Now, suppose that $\mathcal{L}(G_1) \cap \mathcal{L}(G_2) \neq \emptyset$. Then there is a string $y a_{i_k} \cdots a_{i_1} \in \mathcal{L}(G_1) \cap \mathcal{L}(G_2)$. Since $y a_{i_k} \cdots a_{i_1} \in \mathcal{L}(G_1)$ it must be the case that $y = w_{i_1} w_{i_2} \cdots w_{i_k}$. Also, $y$ must be equal to $x_{i_1} x_{i_2} \cdots x_{i_k}$ since $y a_{i_k} \cdots a_{i_1} \in \mathcal{L}(G_2)$. Then $i_1, \ldots, i_k$ will be a solution to PCP instance $(A, B)$.

3. Show that PCP is undecidable over the binary alphabet $\{0, 1\}$.

   **Solution:** Denote PCP over alphabet $\{0, 1\}$ as BPCP. We show that PCP $\leq_{\mathsf{m}}$ BPCP. Let $A = \{w_1, \ldots, w_n\}$ and $B = \{x_1, \ldots, x_n\}$ denote an instance of PCP over some alphabet $\Sigma$. Let $s = |\Sigma|$ and $\Sigma = \{a_1, \ldots, a_s\}$. Define a map $f : \Sigma \to \{0, 1\}^*$ as $f(a_i) = 0^i 1$ for $i \in [1, s]$. Extend this map to strings over $\Sigma^*$ as $F : \Sigma^* \to \{0, 1\}^*$ where for any string $y = a_{i_1} a_{i_2} \cdots a_{i_k}$, $F(y) = f(a_{i_1}) f(a_{i_2}) \cdots f(a_{i_k})$. Observe that PCP instance $(A, B)$ has a solution iff the BPCP instance $(\{F(w_1), \ldots, F(w_n)\}, \{F(x_1), \ldots, F(x_n)\})$ has a solution, when $F$ is one-one. Suppose that $F(y) = F(z)$ for some $y, z \in \Sigma^*$. Let $y = a_{i_1} \cdots a_{i_k}$ and $z = a_{j_1} \cdots a_{j_\ell}$ for some $i_1, \ldots, i_k, j_1, \ldots, j_\ell \in [1, n]$, then $F(y) = f(a_{i_1}) f(a_{i_2}) \cdots f(a_{i_k}) = 0^{i_1} 1 0^{i_2} 1 \cdots 0^{i_k} 1 = f(a_{j_1}) f(a_{j_2}) \cdots f(a_{j_\ell}) = F(z)$. If $y \neq z$ Let $r$ be the minimum integer such that $a_{i_r} \neq a_{j_r}$. Then $f(a_{i_r}) = 0^{i_r} 1 \neq 0^{j_r} 1 = f(a_{j_r})$ but then this implies that $F(y) \neq F(x)$ contradicting our assumption. Therefore $x = z$ and as a consequence $F$ is 1-1.

4. A Turing machine is *minimal* if it has the fewest states among all TMs that accept the same set. Prove that there does not exist an infinite r.e. set of minimal TMs.
**Hint:** Use recursion theorem.

**Solution:** Suppose that $A$ of minimal TMs over a fixed alphabet (say, $\{0,1\}$) is an infinite r.e. set. (Let us restrict ourselves to TMs over input alphabet $\{0,1\}$ and stack alphabet $\Gamma = \Sigma \cup \{\vdash, \sqcup\}$.) Then there exists a machine $\mathcal{N}$ that enumerates $A$. For a TM $\mathcal{M}$, denote by $s(\mathcal{M})$ the number of states of $\mathcal{M}$. Let $\mathcal{K}$ be a machine that computes a map $\sigma : \mathbb{N} \to \mathbb{N}$ defined as: $\sigma(x)$ is the first machine $\mathcal{J}$ enumerated by $\mathcal{N}$ such that $s(\mathcal{J}) > s(\mathcal{M}_x)$ (here, $\mathcal{M}_x$ denotes the TM with description $i$). More precisely, on input $x$, $\mathcal{K}$ does the following.

- Construct $\mathcal{M}_x$ from $x$.
- Use $\mathcal{N}$ to enumerate TMs in $A$.
- Stop when a TM $\mathcal{J}$ with $s(\mathcal{J}) > s(\mathcal{M}_x)$ is enumerated. This event will occur eventually, as there are only finitely many TMs with fewer states than $\mathcal{M}_x$ and $A$ is infinite.
- Output the description/encoding of $\mathcal{J}$.

$\mathcal{K}$ is a total TM and hence $\sigma$ is a total recursive function. By recursion theorem, there exists a fixed point $x_0$ such that $\mathcal{L}(\mathcal{M}_{x_0}) = \mathcal{L}(\mathcal{M}_{\sigma(x_0)})$. But $\mathcal{M}_{\sigma(x_0)} \in A$ is a minimal TM for $\mathcal{L}(\mathcal{M}_{x_0})$ and yet $s(\mathcal{M}_{x_0}) < s(\mathcal{M}_{\sigma(x_0)})$. This contradicts the minimality of $\mathcal{M}_{\sigma(x_0)}$ and consequently, our assumption that $A$ is an infinite r.e. set.

# Tutorial 6

## Oracle TMs and the Arithmetic Hierarchy

**Guidelines:** Solve all problems in the class. *Do not search for solutions online. You are allowed to refer to your notes and discuss among other students and TAs.*

1. Show that
   (a) the language $\mathsf{NE} = \{(\mathcal{M}_1, \mathcal{M}_2) \mid L(\mathcal{M}_1) \neq L(\mathcal{M}_2)\}$ is in $\mathbf{\Sigma}_2^0$.

   **Solution:** If $(\mathcal{M}_1, \mathcal{M}_2) \in \mathsf{NE}$, then there exists a string $x$ such that either $x \in L(\mathcal{M}_1)$ and $x \notin L(\mathcal{M}_2)$, or $x \in L(\mathcal{M}_1)$ and $x \notin L(\mathcal{M}_2)$. That is, either $\mathcal{M}_1$ should accept and $\mathcal{M}_2$ should not or vice-versa. This can be expressed in terms of a $\mathbf{\Sigma}_2^0$-predicate. Define a predicate $P(\mathcal{M}, x, t)$ as

$$P(\mathcal{M}, x, t) = \begin{cases} T & \text{if } \mathcal{M} \text{ accepts } x \text{in} t \text{ steps} \\ F & \text{otherwise} \end{cases}$$

$$\mathsf{NE} = \{(\mathcal{M}_1, \mathcal{M}_2) \mid \exists n \exists x \forall t \ t \geq n \implies R(\mathcal{M}_1, \mathcal{M}_2, x, t)\},$$

   where

$$R(\mathcal{M}_1, \mathcal{M}_2, x, t) = (P(\mathcal{M}_1, x, t) \wedge \neg P(\mathcal{M}_2, x, t)) \vee (\neg P(\mathcal{M}_1, x, t) \wedge P(\mathcal{M}_2, x, t))$$

   By using the map $\tau : \mathbb{N}^2 \to \mathbb{N}$ defined as $\tau(n, x) = \binom{n+x+1}{2} + n$, we can replace $\exists n \exists x$ with a single quantified variable $y = \tau(n, x)$. Therefore $\mathsf{NE} \in \mathbf{\Sigma}_2^0$.

   (b) $\mathsf{NE}$ is $\leq_m$-hard for $\mathbf{\Sigma}_2^0$.

   **Solution:** Let $L = \{x | \exists y \forall z R(x, y, z)\}$ be a language in $\mathbf{\Sigma}_2^0$. We show $L \leq_m \mathsf{NE}$. Given an instance $x$ of $L$, construct $\mathcal{M}_1, \mathcal{M}_2$ as follows. $\mathcal{M}_1$, on any input $w$, accepts and halts. So $\mathcal{L}(\mathcal{M}_1) = \Sigma^*$. $\mathcal{M}_2$, on input $w$ does the following:

   - Enumerate all strings $y$ with length $\leq |w|$
   - For each $y$, find $z$ such that $\neg R(x, y, z)$ is true.
   - If all the trials are successful, then halt and accept.

   If $x \in L$, then $\exists y_0 \forall z R(x, y_0, z)$. If $|w| < |y_0|$, then $\mathcal{M}_2$ accepts $w$. Otherwise, it loops. Hence $\mathcal{L}(\mathcal{M}_2)$ consists of strings of length $< |y_0|$ implying that $\mathcal{M}_2$ accepts a finite language $\neq \Sigma^*$. If $x \notin L$, then for each $y$, we can find a $z$ such that $\neg R(x, y, z)$ is true, in which case $\mathcal{L}(\mathcal{M}_2) = \Sigma^*$. Thus $x \in L \Leftrightarrow (\mathcal{M}_1, \mathcal{M}_2) \in \mathsf{NE}$. As this holds for any language $L \in \mathbf{\Sigma}_2^0$, $\mathsf{NE}$ is $\leq_m$-hard for $\mathbf{\Sigma}_2^0$.

2. Let $\mathsf{HP}_1 = \mathsf{HP}$ and let $\mathsf{HP}_n$ be the halting problem for oracle TMs with oracle $\mathsf{HP}_{n-1}$, $n \geq 2$. That is

$$\mathsf{HP}_n = \{\mathcal{M}^{\mathsf{HP}_{n-1}} \# x \mid \mathcal{M}^{\mathsf{HP}_{n-1}} \text{ halts on } x\}.$$

   Show that $\mathsf{HP}_n \in \mathbf{\Sigma}_n^0 \setminus \mathbf{\Pi}_n^0$.

   **Solution:** For convenience, I omit the oracle from the superscript, when it is clear from context. First, observe that $\mathsf{HP}_n$ is r.e.in $\mathsf{HP}_{n-1}$. Define a TM $\mathcal{K}^{\mathsf{HP}_{n-1}}$ accepting $\mathsf{HP}_n$, that on input $\mathcal{M}^{\mathsf{HP}_{n-1}}, x$, simulates $\mathcal{M}^{\mathsf{HP}_{n-1}}$ on $x$, relaying all of $\mathcal{M}$'s oracle queries to its oracle $\mathsf{HP}_{n-1}$ and relaying back the

responses; accepts if $\mathcal{M}$ halts on $x$. Clearly $\mathcal{L}(\mathcal{K}) = \mathsf{HP}_n$. Since $\mathcal{K}$ is an OTM with oracle $\mathsf{HP}_{n-1} \in \mathbf{\Sigma}^0_{n-1}$, it follows that $\mathsf{HP}_n \in \mathbf{\Sigma}^0_n$.

We now show that $\mathsf{HP}_n$ is not recursive in $\mathsf{HP}_{n-1}$ (i.e., $\mathsf{HP}_n \notin \mathbf{\Delta}^0_n$ or equivalently $\mathsf{HP}_n \in \mathbf{\Sigma}^0_n \setminus \mathbf{\Pi}^0_n$). For $n = 1$, this is true. We have already proved that $\mathsf{HP}_1 = \mathsf{HP}$ is undecidable. Assume the statement is true for $\mathsf{HP}_{n-1}$. Suppose $\mathsf{HP}_n$ is recursive in $\mathsf{HP}_{n-1}$. Let $\mathcal{K}$ be a decider for $\mathsf{HP}_n$ and let $\mathcal{M}_\epsilon, \mathcal{M}_0, \mathcal{M}_1, \mathcal{M}_{00}, \mathcal{M}_{11}, \ldots$ be an enumeration of all oracle TMs with oracle $\mathsf{HP}_{n-1}$. Define a TM $\mathcal{N}$ that on input $x$ does the following:

- Construct $\mathcal{M}_x$ from $x$.
- Simulate $\mathcal{K}$ on input $(\mathcal{M}_x, x)$ (oracle queries made by $\mathcal{K}$ are relayed to $\mathcal{N}'s$ oracle).
- If $\mathcal{K}$ accepts, then reject; otherwise reject.

Note that $\mathcal{K}$ accepts $\mathcal{M}_x, x$ iff $\mathcal{M}_x$ halts on $x$. As a result, $\mathcal{M}_x$ halts on $x \Leftrightarrow x \notin \mathcal{L}(cN)$. That is, $\mathcal{N}$'s behaviour differs from each $\mathcal{M}_x$ exactly on input $x$ and hence $\mathcal{N}$ cannot be $\mathcal{M}_w$ for any string $w$. But the enumeration contains all TMs with oracle $\mathsf{HP}_{n-1}$. This contradicts our assumption that $\mathsf{HP}_n$ is recursive in $\mathsf{HP}_{n-1}$.

# Tutorial 7

## Basic Complexity Classes and NP-Completeness - I

**Submission Guidelines** Solve all problems in the class. *Do not search for solutions online. You are allowed to refer to your notes and discuss among other students and TAs.*

1. Prove that the following languages (defined over graphs) are in **P**.
   (a) BIPARTITE – the set of all bipartite graphs. That is, $G = (V, E) \in$ BIPARTITE if $V$ can be partitioned into two sets $V_1, V_2$ such that every edge in $E$ is adjacent to a vertex in $V_1$ and a vertex in $V_2$.

   **A:** Choose a vertex $v \in V$ and perform a breadth-first search (BFS) starting at $v$ as the root, colouring vertices visited in odd interations red and those visited in even iterations black. At the end, if there exists an edge whose end points are both coloured with the same colour, then $G \notin$ BIPARTITE. Else, $G \in$ BIPARTITE. This algorithm would take $O(n^2)$ time where $n = |V|$.

   (b) TRIANGLE-FREE – the set of all graphs that do not contain a triangle (where triangle is a set of three distinct vertices that are mutually connected).

   **A:** Let $n = |V|$. Check all $\binom{n}{3}$ sets of vertices and check if they are mutually connected. If none of them are, then $G \in \mathsf{TRIANGLE-FREE}$. Since $\binom{n}{3} = O(n^3)$, the algorithm runs in polynomial time.

2. Normally, we assume that numbers are represented as strings using the binary basis. That is, a number $n$ is represented by the sequence $x_0, x_1, \ldots, x_{\log n}$ such that $n = \sum_{i=0}^{\log n} x_i 2^i$. However, we could have used other encoding schemes. If $n \in \mathbb{N}$ and $b \geq 2$, then the *representation of $n$ in base $b$*, denoted by $\llcorner n \lrcorner_b$ is obtained as follows: first represent $n$ as a sequence of digits in $\{0, \ldots, b-1\}$, and then replace each digit by a sequence of zeroes and ones. The unary representation of $n$, denoted by $\llcorner n \lrcorner_1$ is the string $1^n$ (i.e., a sequence of $n$ ones).
   (a) Show that choosing a different base of representation (other than unary) will make no difference to the class **P**. That is, show that for every subset $S$ of the natural numbers, if we define $L_S^b = \{\llcorner n \lrcorner_b : n \in S\}$ then for every $b \geq 2$, $L_S^b \in \mathbf{P}$ iff $L_S^2 \in \mathbf{P}$.

   **A:** Let $D(k)$ (resp. $M(k)$) be the time complexity of division (resp. multiplication) of two $k$-bit numbers. Suppose that $L_S^b \in \mathbf{P}$. Then there exists a Turing machine $\mathcal{M}$ that runs in polynomial time recognising $L_S^b$. Any $k$-bit input $\llcorner x \lrcorner_2$ ($x \in \mathbb{N}$) can be converted to base-$b$ representation in time $D(k) \cdot \log_b x$, which is polynomial in $k$. The resulting string $\llcorner x \lrcorner_b$ can then be provided as input to $\mathcal{M}$, thereby recognising $L_S^2$. In other words, $L_S^b \in \mathbf{P}$ implies $L_S^2 \in \mathbf{P}$. Similarly, an input string of length $k$ for $L_S^b$ (i.e., a base-$b$ number) can be converted to its binary representation in $M(k) \cdot \log_b x$ time. This shows that $L_S^2 \in \mathbf{P}$ implies $L_S^b \in \mathbf{P}$.

   (b) Show that choosing the unary representation makes a difference by showing that the following language is in **P**.

   $$\mathsf{UNARYFACTORING} = \{\langle \llcorner n \lrcorner_1, \llcorner k \lrcorner_1 \rangle : \text{there is a } j \leq k \text{ dividing } n\}.$$

   **A:** Check for divisibility of $n$ by all numbers $\leq k$. This takes $k$ steps which is polynomial in the size $(n+k)$ of the input.

3. Show that the halting problem is **NP**–hard.

**A:** HP is **NP**–hard if $L \leq_p$ HP for all $L \in$ **NP**. Equivalently HP is **NP**–hard if SAT $\leq_p$ HP. We show a polynomial time computable function mapping a formula $\phi$ to a TM $\mathcal{N}$ and a string $x$ such that $\phi \in SAT$ iff $(\mathcal{N}, x) \in$ HP. Let $\mathcal{M}$ be a polynomial time deterministic machine that takes as input a formula $\phi$, an assignment $z$ and accepts iff $z$ satisfies $\phi$. Such a TM exists since SAT $\in$ **NP**. Construct $\mathcal{N}$ so that on input $x$ it does the following:

- Parse $x$ as a Boolean formula over $n$ vairables.
- For all assignments $z \in \{0,1\}^n$, run $\mathcal{M}(x, z)$ to check if $z$ satisfies $x$; accept and halt if $\mathcal{M}$ accepts.
- If $x$ is not satisfied by any $z$, enter a trivial loop.

Map $\phi$ to $\mathcal{N}, \phi$. This map can be computed in time polynomial in $|\phi|$ (the size of $\phi$) irrespective of fact that $\mathcal{N}$'s running time would be exponential in the $|\phi|$. Now, $\phi \in$ SAT iff $\exists z$ such that $\mathcal{M}(\phi, z)$ accepts iff $\mathcal{N}$ halts on input $\phi$ iff $(\mathcal{N}, \phi) \in$ HP. This completes the reduction SAT $\leq_p$ HP.

4. Let DOUBLESAT $= \{\langle\phi\rangle : \phi$ is a boolean formula in CNF having at least two satisfying assignments$\}$. Show that DOUBLESAT is **NP**–complete.

   **A:** DOUBLESAT $\in$ **NP** – the certificate is a pair of assignments of $0, 1$ values to the variables of $\phi$, each of which can be checked in polynomial time (in fact, in linear time) for whether they satisfy $\phi$ or not.

   To show that DOUBLESAT is **NP**–hard, we show that SAT $\leq_p$ DOUBLESAT. Let $\phi$ be an instance of SAT defined over $n$ variables $u_1, \ldots, u_n$. The reduction picks a new variable $u_{n+1}$ and generates $\phi' = \phi \wedge (x \vee \bar{x})$ as the instance of DOUBLESAT. If there's a satisfying assignment $z \in \{0,1\}^n$ for $\phi$, then there are two assignments $z_1 = z||0$ (indicating $u_{n+1} = 0$) and $z_2 = z||1$ (indicating $u_{n+1} = 1$) for $\phi'$.

5. Let $S$ be a set and let $C = \{X_1, \ldots, X_n\}$ be a collection of $n$ subsets of $S$ (for each $i \in [1, n]$, $X_i \subseteq S$). A set $S'$, with $S' \subseteq S$, is called a hitting set for $C$ if every subset in $C$ contains at least one element in $S'$, i.e., $|X_i \cap S'| \geq 1$ for each $i \in [1, n]$. Let HITSET be the language $\{\langle C, k\rangle : C$ has a hitting set of size $k\}$. Prove that HITSET is **NP**–complete.

   **Example** $S = \{a, b, c, d, e, f, g\}$, $C = \{\{a, b, c\}, \{d, a\}, \{d, e, f\}, \{g\}\}$

   - $k = 2$, no hitting sets exist.
   - $k = 3$, $S' = \{a, d, g\}$ (other choices exist).

   **Hint** Try reducing from VERTEXCOVER.

   **A:** Clearly, HITSET $\in$ **NP** – guess a set $S'$ of size $k$ and for each set $X_i \in C$, check for inclusion of elements of $S'$ in $X_i$. This can be done in $nk$ time.

   We now show that VERTEXCOVER $\leq_p$ HITSET. We need to transform an instance $\langle G = (V, E), k\rangle$ into an instance $\langle S, C, k\rangle$ such that $\langle G = (V, E), k\rangle \in$ VERTEXCOVER iff $\langle S, C, k\rangle \in$ HITSET. Set $S = V$ and for each edge $\{u, v\} \in E$, create a set $X_{u,v} = \{u, v\}$ and add it to $C$.

   Suppose that $V' \subseteq V$ is a vertex cover for $G$ fo size $k$, then for every edge $\{u, v\} \in E$ either $u \in V'$ or $v \in V'$. That is, $|X_{u,v} \cap V'| \geq 1$ thus giving us a hitting set $S' = V'$ of size $k$. Suppose that $S' \subseteq S$ is a hitting set of size $k$ for $C$. Then $|S' \cup X_{u,v}| \geq 1$ for every $X_{u,v}$ with $\{u, v\} \in E$, that is, choosing $V' = S'$ ensures that $V'$ contains at least one end-point of every edge.

# Tutorial 7

## Basic Complexity Classes and NP-Completeness - II

**Submission Guidelines** Solve all problems in the class. *Do not search for solutions online. You are allowed to refer to your notes and discuss among other students and TAs.*

1. Let $\phi$ be a Boolean formula in CNF. An assignment to the variables of $\phi$ is called *not-all-equal* if in each clause, at least one literal is assigned 1 and at least one literal is 0. Show that the language

$$\neq \mathsf{SAT} = \{\langle\phi\rangle : \phi \text{ is a CNF formula having a satisfying } \textit{not-all-equal} \text{ assignment}\}$$

is **NP**-complete.

**Note:** Assume no constants are allowed in the formulae. Otherwise the problem becomes trivial – just add a zero to each clause.

**A:** Clearly $\neq \mathsf{SAT}$ is in **NP**. We show that $\mathsf{SAT} \leq_p \neq \mathsf{SAT}$. Let $\phi = c_1 \wedge c_2 \wedge \cdots \wedge c_m$ be an instance of SAT. Construct a formula $\psi$ as follows: introduce $m$ new variables $w_1, \cdots, w_m, z$ and for each clause $c_i = x_1 \vee \cdots \vee x_k$ (where $x_i$'s are literals), include two clauses $c_{i,1} = x_1 \vee \cdots \vee x_{k-1} \vee w_i$ and $c_{i,2} = x_k \vee \neg w_i \vee z$ in $\psi$. Before we proceed, we state a simple claim that can be easily verified.

**Claim 1.** *If $f : \{u_1, \ldots, u_n\} \to \{0, 1\}$ is a not-all-equal satisfying assignment for a CNF formula $\phi$ over variables $u_1, \ldots, u_n$, then so is $\neg f$ ($\neg f$ just flips the assignment to each variable).*

$\phi \in \mathsf{SAT} \implies \psi \in \neq \mathsf{SAT}$**:** If $\phi \in \mathsf{SAT}$, there is an assignment of truth values to $u_1, \ldots, u_n$ such that $\phi$ is true. We show that there exists a not-all-equal satisfying assignment for $\psi$. The assignment to $u_1, \ldots, u_n$ remains unchanged. Since each clause of $\phi$ is satisfied, so is $c_i = x_1 \vee \cdots \vee x_{k-1} \vee x_k$. If $x_1 \vee \cdots \vee x_{k-1}$ true, then $c_{i,1}$ is true; setting $w_i = 0, z = 0$ ensures that $c_{i,2}$ is also satisfied and hence $\psi$ has a not-all-equal assignment. Otherwise, it must be the case that $x_k = 1$. Then setting $w_i = 1, z = 0$ leads to a not-all-equal assignment of $\psi$.

$\psi \in \neq \mathsf{SAT} \implies \phi \in \mathsf{SAT}$**:** If $\psi \in \neq \mathsf{SAT}$, i.e., there exists a satisfying not-all-equal assignment of variables $u_1, \ldots, u_n, w_1, \ldots, w_m, z$. We show that $\phi \in \mathsf{SAT}$ by exhibiting a satisfying assignment for $\psi$. Suppose that the not-all-equal assignment sets $z = 0$. Then the same assignemnt to $u_1, u_2, \ldots, u_n$ will satisfy $\phi$ – consider clause $c_i$; if $x_k = 1$ then $c_i$ is satisfied; otherwise since $c_{i,2}$ has atleast one literal assigned 1, it must be the case that $\neg w_i = 1$, i.e., $w_i = 0$ and since $c_{i,1}$ is satisfied, $x_1 \vee \cdots \vee x_{k-1}$ is true and so $c_i$ must be true. Now suppose that the not-all-equal assignment $f$ for $\psi$ sets $z = 1$, then $\neg f$ would still satisfy $\psi$ and ensure $z = 0$. Use the same reasoning as above to show a satisfying assignment for $\phi$.

□

2. Let $S$ be a finite set and $\mathcal{C} = \{C_1, C_2, ..., C_k\}$ be a collection of subsets of $S$, for some $k > 0$. We say $S$ is *2-colourable with respect to $C$* if we can colour each element of $S$ in red or blue, such that every $C_i$ contains at least one red element and at least one blue element. Show that the language

$$\mathsf{2COLOUR} = \{\langle S, \mathcal{C}\rangle : S \text{ is 2-colourable with respect to } \mathcal{C}\}$$

is **NP**-complete.

**Hint:** Reduce from $\neq$ SAT.

**A:** Let $\phi = c_1 \wedge c_2 \wedge \cdots \wedge c_m$, a CNF formula over $n$ variables $u_1, \ldots, u_n$, be an instance of $\neq SAT$. We will show how to transform $\phi$ to a set $S$ and a collection of its subsets $\mathcal{C}$ such that $\phi \in \neq$ SAT iff $\langle S, \mathcal{C} \rangle \in$ 2COLOUR. Define $S$ to contain elements $x_i, x_i'$ corresponding to literals $u_i, \neg u_i$ for $i \in [1, n]$. Add the subsets $B_i = \{x_i, x_i'\}$ (for $i \in [1, n]$) to $\mathcal{C}$. Further, include subsets $C_j$ for $j = 1, \ldots, m$ constructed as follows: for every $i = 1, \ldots, n$, if clause $c_j$ contains $u_i$, then $C_j \leftarrow C_j \cup \{x_j\}$; otherwise, if $c_j$ contains $\neg u_i$, then $C_j \leftarrow C_j \cup \{x_j'\}$.

Suppose that $\phi$ has a not-all-equal satisfying assignment. Then we show a 2-colouring of $S$ such that $\langle S, \mathcal{C} \rangle \in$ 2COLOUR. If $u_i = 1$, then colour $x_i$ red and $x_i'$ blue. Otherwise, colour $x_i$ blue and $x_i'$ red. This would ensure that each $B_i$ contains at least one blue element and one red element. Further, since each clause has at least one literal assigned 1 and at least one literal assigned 0, the red (resp. blue) elements in $C_j$ would exactly correspond to the literals assigned 1 (resp. 0). Hence each $C_j$ would contain at least one red element and at least one blue element.

For the other direction, suppose that $S$ is 2-colourable with respect to $\mathcal{C}$. Fix a 2-colouring of $S$. For each $i \in [1, n]$, if $x_i$ is coloured red, then it must be the case that $x_i'$ is coloured blue (and vice-versa), since otherwise $B_i$ would have both its entries coloured in red (resp. blue). If $x_i$ is coloured red, then set $u_i = 1$; otherwise, set $u_i = 0$. To see that this is a not-all-equal satisfying assignment for $\phi$, observe that $C_j$ contains at least one element coloured in red and at least one element coloured in blue. The literals in clause $c_j$ corresponding to red and blue elements would be assigned 1 and 0 respectively. This holds for each $j \in [1, m]$ and hence this is a satisfying not-all-equal assignment.

We have $\phi \in \neq$ SAT iff $\langle S, \mathcal{C} \rangle \in$ 2COLOUR. Further, $\neq$ SAT is **NP**-complete and the above reduction runs in polynomial time. Therefore, 2COLOUR is **NP**-complete. □

3. Prove that $\mathbf{P} = \mathbf{coP}$ and $\mathbf{P} \subseteq \mathbf{NP} \cap \mathbf{coNP}$.

**A:** Let $L \in \mathbf{P}$. Then $\neg L$ can be decided in polynomial time – run the TM deciding $L$ and flip its decision. So, $\mathbf{P} = \mathbf{coP}$.

We know that $P \subseteq \mathbf{NP}$. Now, a language $L \in \mathbf{coNP}$ iff $\neg L \in \mathbf{NP}$. Let $L \in P$. Then $\neg L \in \mathbf{coP} = \mathbf{P} \subseteq \mathbf{NP}$. So, by the definition of **coNP**, $\neg\neg L = L \in \mathbf{coNP}$. Therefore, $\mathbf{P} \subseteq \mathbf{coNP}$. □

4. Assuming $\mathbf{NP} \neq \mathbf{coNP}$, show that no **NP**-complete problem can be in **coNP**.

**A:** Suppose that $L$ is an **NP**-complete language and $L \in \mathbf{coNP}$. Then for every language $L' \in \mathbf{NP}$, we have $L' \leq_p L$ and since $L \in \mathbf{coNP}$, $L' \in \mathbf{coNP}$. So, we have $\mathbf{NP} \subseteq \mathbf{coNP}$. Now consider any $L' \in \mathbf{coNP}$. Then $\neg L' \in NP$ and $\neg L' \leq_p L$. Using the same reduction, we can say $L' \leq_p \neg L$ and $\neg L \in \mathbf{NP}$, since $L \in \mathbf{coNP}$. So, we have $L' \in \mathbf{NP}$ and hence $\mathbf{coNP} \subseteq \mathbf{NP}$, thus implying that $\mathbf{NP} = \mathbf{coNP}$ – contradiction! □

5. The following two classes are exponential time analogues of **P** and **NP**.

$$\mathbf{EXP} = \cup_{c \geq 1} \mathbf{DTIME}(2^{n^c})$$

$$\mathbf{NEXP} = \cup_{c \geq 1} \mathbf{NTIME}(2^{n^c})$$

Clearly, $\mathbf{P} \subseteq \mathbf{NP} \subseteq \mathbf{EXP} \subseteq \mathbf{NEXP}$. Show that if $\mathbf{EXP} \neq \mathbf{NEXP}$, then $\mathbf{P} \neq \mathbf{NP}$.

**Hint:** Consider padding strings in **EXP**/**NEXP** languages with exponentially sized strings in order to "scale down" to **P**/**NP**.

**A:** We show that if $\mathbf{P} = \mathbf{NP}$, then $\mathbf{EXP} = \mathbf{NEXP}$. Let $L \in \mathbf{NTIME}(2^{n^c})$ and let $\mathcal{M}$ be a Turing machine deciding $L$. Define language $L'$ as follows.

$$L' = \{\langle x \| 1^{2^{|x|^c}} \rangle : x \in L\},$$

where $\|$ denotes concatenation. We show that $L' \in \mathbf{NP}$ by constructing an NDTM $\mathcal{M}'$ deciding $L'$. Given input $y$, $\mathcal{M}'$ checks whether $y$ is of the form $z||1^{2^{|z|^c}}$ for some string $z$. If not, reject. Otherwise, run $\mathcal{M}$ on $z$ for $2^{|z|^c}$ steps and output its decision. The running time of $\mathcal{M}'$ is $O(|z| + 2^{|z|^c})$ which is polynomial in $|y|$. Hence, $L' \in \mathbf{NP}$. If $\mathbf{P} = \mathbf{NP}$, then $L' \in \mathbf{P}$ i.e., there is a deterministic TM $\mathcal{M}''$ deciding $L$ in polynomial time. If $L' \in \mathbf{P}$, then $L \in \mathbf{EXP}$ – given any $x \in L$, pad $x$ with $1^{2^{|x|^c}}$ and provide the resulting string as input to $\mathcal{M}''$. If $\mathcal{M}''$ accepts, then accept; else, reject. Thus we can decide membership in $L$ deterministically in exponential time. Therefore, $\mathbf{EXP} \subseteq \mathbf{NEXP}$ and as a result $\mathbf{EXP} = \mathbf{NEXP}$. □

6. (Scaling recource bounds.) Let $\mathbf{CL}_1, \mathbf{CL}_2$ denote some time/space complexity classes. Show that if $\mathbf{CL}_1(f(n)) \subseteq \mathbf{CL}_2(g(n))$, then $\mathbf{CL}_1(f(n^c)) \subseteq \mathbf{CL}_2(g(n^c))$.

   **Hint:** Use a padding technique, as in the previous problem.

   **A:** Let $L \in \mathbf{CL}_1(f(n^c))$ and let $\mathcal{M}$ be a TM deciding $L$ with resource bound $f(n^c)$ i.e., $\mathcal{M}$ decides $L$ in $f(n^c)$ time/space. The statement trivially holds for $c = 1$. For $c \geq 2$, define

   $$L' = \{\langle x \| 0 \| 1^{|x|^c - |x| - 1} \rangle : x \in L\}.$$

   Define $\mathcal{M}'$ that accepts input $y$ iff $y$ is of the form $x\|0\|1^{|x|^c-|x|-1}$ and $\mathcal{M}$ accepts $x$. $\mathcal{M}'$ runs in time/space $O(f(|x|^c))$. But $O(f(|x|^c)) = O(f(|y|))$ implying that $L' \in \mathbf{CL}_1(f(n)) \subseteq \mathbf{CL}_2(g(n))$. Hence there is a TM $\mathcal{N}'$ deciding $L'$ using $g(n)$ time/sapce. Define a machine $\mathcal{N}$ deciding $L$ as follows: $\mathcal{N}(x) = \mathcal{N}'(x\|0\|1^{|x|^c-|x|-1})$. Clearly $\mathcal{N}$ runs in time/space $g(|x01^{|x|^c-|x|-1}|) = g(|x|^c)$. Therefore $L \in \mathbf{CL}_2(g(n^c))$. □

# Tutorial 9
## Space Complexity

**Submission Guidelines** Solve all problems in the class. *Do not search for solutions online. You are allowed to refer to your notes and discuss among other students and TAs.*

1. Show that
   (a) **PSPACE** is closed under union, intersection, complement and Kleene Closure.

   **A:** Let $L_1, L_2 \in$ **PSPACE** and let $\mathcal{M}_1, \mathcal{M}_2$ be deterministic TMs deciding $L_1, L_2$ respectively in polynomial space. We describe constructions of DTMs running in polynomial space for the following languages.

   $L_1 \cup L_2$: On input $x$, run $\mathcal{M}_1$ on $x$. If it accepts, then accept. Otherwise, run $\mathcal{M}_2$ on $x$. If it accepts, then accept; else reject.

   $L_1 \cap L_2$: On input $x$, run $\mathcal{M}_1$ on $x$ and then run $\mathcal{M}_2$ on $x$. If both accept, then accept; otherwise reject.

   $L_1^c$: On input $x$, run $\mathcal{M}_1$ on $x$ and negate its output. (This can be done since $\mathcal{M}$ is deterministic).

   $L_1^*$: Observe that $x \in L_1^*$ iff one of the following holds: $x = \varepsilon$, $x \in L_1$, $x = wz$ such that $w \in L_1^*$ and $z \in L_1^*$. Let $x = x_1 x_2 \cdots x_n$ and let $x_{i,j}$ (for $1 \le i \le j \le n$) denote the substring $x_i x_{i+1} \cdots x_j$ of $x$. We will use dynamic programming to build a matrix $T = (t_{i,j})$ with $t_{i,j} = 1$ if $x_{i,j} \in L_1^*$ and 0 otherwise. Essentially we consider all substrings of $x$, starting with substrings of length 1 and ending with all substrings of length $n$. Below is the pseudocode:

```
Input: x = x_1 ··· x_n

  Initialise t_{i,j} to 0.
  If x = ε, then accept;
  For k = 1,...,n:
    For i = 1,...,n−k+1:
      j = i+k−1;
      Run M_1 on x_{i,j};
      If M_1 accepts, then set t_{i,j} = 1;
      Else
        For ℓ = i,...,j−1:
          If t_{i,ℓ} = 1, and t_{ℓ+1,j} = 1, then set t_{i,j} = 1;
  If t_{1,n} = 1, then accept; Else reject;
```

   We now analyze the space complexity of the above algorithm. In each inner loop, we run $\mathcal{M}_1$ once which takes polynomial space (since $L_1 \in$ **PSPACE**). Storing $T$ requires $O(n^2)$-space and each execution of the loop requires constant number of cells to store the indices. Hence the algorithm runs in polynomial space.

   Alternatively, using the fact that **PSPACE** = **NPSPACE**, we can define a poly-space NDTM that decides $L_1^*$ that guesses an index in $x$; checks whether $x_{1,i} \in L_1$ using $\mathcal{M}_1$ and then recursively checks whether $x_{i+1,n} \in L_1^*$.

   It is easy to verify that the first three run in polynomial space.

(b) **NL** is closed under union, intersection and Kleene Closure.

**A:** Union and intersection can be dealt with similar to the previous question.

For any language $L \in \mathbf{NL}$, we show that $L^* \in \mathbf{NL}$. Let $\mathcal{M}$ be an NDTM deciding $L$ in logarithmic space. We show how to construct an NDTM deciding $L^*$. Let $x = x_1 \cdots x_n \in \Sigma^*$. If $x = \varepsilon$, then accept. Otherwise, non-deterministically choose $i \in \{1, \ldots, n\}$ and let $w = x_1 \cdots x_i$, $z = x_{i+1} \cdots x_n$. Run $\mathcal{M}$ with $w$ as the input. If $\mathcal{M}$ accepts, then recursively check if $z \in L^*$. Else, reject.

The above algorithm always correctly determines if $x \in L^*$. If $x \in L^*$, then $x$ can be written as $w_1 w_2 \cdots w_k$ for some $k$ such that $|w_j| \geq 1$ for each $j$. Therefore, there exists a sequence of guesses leading to $w_1, \ldots, w_k$ in that order. Suppose that $x \notin L^*$. Then for any $w$ with $|w| \geq 1$ such that $x = wz$, either $w \notin L$ or $z \notin L^*$ and so all branches of computation of the NDTM are rejecting. Furthermore, guessing a prefix only takes logarithmic space, that is to index into a position in $x$. Checking whether $w \in L$ takes logarithmic space since $\mathcal{M}$ is a logspace machine. Once $w$ is checked, it does not have to be remembered for the recursive calls. Hence, $x \in L^*$ can be checked in log space and as a consequence $L^* \in \mathbf{NL}$.

2. Show that the language consisting of strings with properly nested parentheses is in **L**.

**A:** Initialise a counter to zero. While reading the input string from left to right, increment the counter whenever a left paranthesis is encountered. Decrement the counter whenever a right paranthesis is read. Reject the string if the counter becomes negative or does not become zero after reading the end of the input string. Otherwise accept. It requires log-space to store the counter.

3. A *ladder* is a sequence of strings $s_1, s_2, \ldots, s_k$, wherein every string differs from the preceding one in exactly one character. For example, the following is a ladder of English words, starting with "head" and ending with "free": head, hear, near, fear, bear, beer, deer, deed, feed, feet, fret, free. Let

$$\mathsf{LADDER}_{DFA} = \{\langle \mathcal{M}, s, t \rangle : \mathcal{M} \text{ is a DFA and } L(\mathcal{M}) \text{ consists of a ladder of strings starting with } s \text{ and ending with } t\},$$

where $s, t \in \Sigma^*$ and $\mathcal{M}$ is defined over the input alphabet $\Sigma$. Show that $\mathsf{LADDER}_{DFA}$ is in **PSPACE**.

**Hint** Use the fact that **PSPACE** = **NPSPACE**.

**A:** Since **NPSPACE** = **PSPACE**, it suffices to show that $\mathsf{LADDER}_{DFA} \in \mathbf{NPSPACE}$. We describe a non-deterministic algorithm that decides $\mathsf{LADDER}_{DFA}$. Given an instance $\langle \mathcal{M}, s, t \rangle$, reject if $|s| \neq |t|$. Otherwise, consider a graph $G$ whose vertices are labeled with strings in $\Sigma^{|s|}$. There is a directed edge from vertex $u$ to vertex $v$ if $u$ and $v$ differ in exactly one character and $u, v \in L(\mathcal{M})$. Then, $\langle \mathcal{M}, s, t \rangle \in \mathsf{LADDER}_{DFA}$ iff there is a path from $s$ to $t$ in $G$. This can be checked non-deterministically in polynomial space (although $G$ has exponential number of vertices) – guess the path, storing at each step only the label of the current vertex (which is of size $|s|$). At step, say starting at vertext $u$, non-deterministically choose a new vertext $v$ connected to $u$ via an edge and check if $\mathcal{M}$ accepts $v$.

To ensure that the machine always halts, maintain a counter which is incremented after each guess. Reject and halt when the counter crosses $|\Sigma|^{|s|}$. Since any path from $s$ to $t$ (without loops) can be of length at most $|\Sigma|^{|s|}$.

4. In the generalised version of the game Tic-Tac-Toe, 2 players places marks $X$ (crosses) and $O$ (noughts) on an $m \times n$ grid. A player wins if she is the first to place $k$ marks in a row, column

or diagonal. The game ends in a draw if no such sequence is present when all the $mn$ cells of the grid are filled. Assuming that $X$ always starts, show that the language

$$\mathsf{GTICTACTOE} = \{\langle m, n, k, c\rangle : c \text{ is an intermediate configuration on the } m \times n \text{ board with next move by } X \text{ and } \exists \text{ a winning strategy for } X\}$$

is in **PSPACE**.

**A:** Given an instance $\langle m, n, k, c\rangle$, we will describe a recursive procedure to check if there exists a winning strategy for $X$. The validity of $c$ may be checked in polynomial space by checking that there are equal number of crosses and noughts. Further, it is possible to check that the game has not already ended by determining whether there exists a series of $k$ crosses or noughts in a row, column or diagonal (this would take $O(nm)$-space since there are $nm$ possible cells and $k < \min\{m, n\}$). This as well can be done in polynomial space. If there are $k$ consecutive $X$ marks, then accept. If there are $k$ consecutive marks of both $X$ and $O$ or only $O$, reject. If there is no unoccupied cell, then reject. Otherwise, repeat the following for each unoccupied cell $u$ in configuration $c$: *Put the marker $X$ on $u$. If the resulting configuration $d$ is a winning configuration for $X$ then accept. If the grid is completely filled, then reject. Otherwise, generate configurations $e_1, e_2, \ldots, e_\ell$ obtained by marking exactly one unoccupied cell in $d$ with $O$. Make a recursive call on each $e_i$, reusing the space for each call. If all the calls return 'accept', then accept. Otherwise, continue with checking a different unoccupid cell in $c$.* If all the configurations $d$ lead to rejection, then reject. Each recursion requires remembering $c, d, e_i$ at a time; checking whether there are $k$ consecutive $X$ marks ina row/column/diagonal requires $O(nm)$-space; the depth of the recursion is $O(mn)$. Based on these observations it is straightforward to see that the procedure requires space $O(m^2n^2)$-space thus implying that $\mathsf{GTICTACTOE} \in$ **PSPACE**.

5. Let $\mathbf{polyL} = \cup_{c>0}\mathbf{DSPACE}(\log^c n)$. Let **SC** (named after Stephen Cook) be the class of languages that can be decided by deterministic machines that run in polynomial time and $\log^c n$ space for some $c > 0$.

   (a) It is an open problem whether $\mathsf{PATH} \in \mathbf{SC}$. Why does Savitch's theorem not resolve this question?

   **A:** We know that $\mathsf{PATH} \in \mathbf{P}$ – simply use breadth-first search. We also know that it is in **NL** – nondeterministically guess a path from the source to the target vertex, keeping track of the current vertex at each step, thus taking logarithmic space. By Savitch's theorem, we have $\mathbf{NSPACE}(S(n)) \subseteq \mathbf{DSPACE}(S(n)^2)$ and hence $\mathsf{PATH} \in \mathbf{NL} = \mathbf{NSPACE}(\log n) \subseteq \mathbf{DSPACE}(\log^2 n) \subseteq \mathbf{polyL}$. The property that $\mathbf{NSPACE}(\log n) \subset \mathbf{DTIME}(2^{\log n}) = \mathbf{DTIME}(n)$ is not preserved by Savitch's theorem. That is, the transformation used in Savitch's theorem to transform a non-deterministic $\log n$-space algorithm to a deterministic $\log^2 n$-space algorithm does not ensure that the resulting algorithm runs in polynomial time. Since $\mathbf{DSPACE}(\log^2 n) \subset \mathbf{DTIME}(2^{\log^2 n}) = \mathbf{DSPACE}(n^{\log n})$, all we can say about a language in $\mathbf{DSPACE}(\log^2 n)$ is that there exists a TM deciding it running in time at most $n^{\log n}$ which is not polynomial in $n$.

   (b) Is $\mathbf{SC} = \mathbf{polyL} \cap \mathbf{P}$?

   **A:** Clearly, it holds that $\mathbf{SC} \subseteq \mathbf{polyL} \cap \mathbf{P}$. The other direction is not necessarilily true. For any language $L \in \mathbf{polyL} \cap \mathbf{P}$, we can say that there exist Turing machines $\mathcal{M}_1, \mathcal{M}_2$ running in polylogarithmic space and polynomial time respectively that decide membership in $L$. This does not imply that there exists a single Turing machine that runs (simultaneously) in polynomial time and polylogarithmic space.

6. Show that 2SAT is in **NL**.

**A:** Consider a 2SAT instance $\phi = \wedge_{i=1}^{m}(x_i \vee y_i)$ over variables $u_1, \ldots, u_n$. Let $G$ be a graph with $2n$ vertices corresponding to the literals $u_i, \neg u_i$ for each $i$. For each clause $x \vee y$, add directed edge $(\neg x, y)$ and $(\neg y, x)$ to $G$, so that $(v, w)$ is an edge in $G$ iff $\neg v \vee w$ is a clause in $\phi$. The edge $(\neg x, y)$ indicates that if $x = 1$ then $y$ must be 1 in order to satisfy the clause $x \vee y$. Note that $G$ can be constructed in logarithmic space.

Let $v, w$ be arbitrary vertices in $G$. If $G$ contains a path from $v$ to $w$, the there must exist one from $\neg w$ to $\neg v$. Let the path be $w \to z_1 \to \cdots \to z_k \to v$. By our construction, edges $\neg v \to \neg z_k$, $\neg z_j \to \neg z_{j-1}$ (for $j \in [2, k]$) and $\neg z_1 \to w$ also belong to $G$. Hence there is a path from $\neg w$ to $\neg v$.

We now claim that $\phi$ is unsatisfiable iff there exists a variable $u$ such that there are paths from $u$ to $\neg u$ and from $\neg u$ to $u$. Suppose, for the sake of contradiction that, there exist paths $u \to \neg u$ and $\neg u \to u$ and $\phi$ has a satisfying assignment with $u = 1$ (the case $u = 0$ can be analysed similarly). Let $u \to \cdots v \to w \cdots \to \neg u$ be the path from $u$ to $\neg u$. Since $u = 1$, all literals in the path $u$ to $w$ must be true. Similarly, since $\neg u = 0$, all literals in the path from $w$ to $\neg u$ must be false. Since there is an edge $(v, w)$ and $v = 1$ and $w = 0$, the clause $\neg v \vee w$ is not satisfied, thus contradicting our assumption that $\phi$ is satisfiable.

Hence checking for the existence of paths $u \to \neg u$ and $\neg u \to u$ will determine whether or not $\phi$ is satisfiable. We know that PATH is in **NL** and we need polynomially many queries to PATH. Therefore, 2SAT $\in$ **NL**. Note that this simultaneously shows that 2SAT and $\overline{\text{2SAT}}$ are in **NL** since **NL** = **coNL**.

# Tutorial 2

## Recursive and R.E. Sets, (Un)Decidability, Rice's Theorem

**Guidelines:** Solve all problems in the class. *Do not search for solutions online.*

1. Show that the class of recursively enumerable sets is closed under union and intersection.

   **A:** Let $A, B$ be *r.e.* sets and let $\mathcal{M}_A, \mathcal{M}_B$ be the respective Turing machines recognising them. Consider the TM $\mathcal{N}$ that does the following on input $x$.

   - Run $\mathcal{M}_A$ on $x$ and $\mathcal{M}_B$ on $x$ in a round-robin fashion.
   - Accept if either $\mathcal{M}_A$ or $\mathcal{M}_B$ accepts.

   Clearly if $x \in A \cup B$, either $\mathcal{M}_A$ or $\mathcal{M}_B$ accepts it causing $\mathcal{N}$ to accept. $\mathcal{N}$ is a valid TM and hence $A \cup B$ is *r.e.*

   Next, let $\mathcal{K}$ be a TM that, on input $x$, does the following.

   - Run $\mathcal{M}_A$ on $x$.
   - If $\mathcal{M}_A$ accepts, then run $\mathcal{M}_B$ on $x$.
   - Accept if $\mathcal{M}_B$ accepts.

   $\mathcal{K}$ accepts $x$ only if both $\mathcal{M}_A$ and $\mathcal{M}_B$ accept $x$ and hence $L(\mathcal{K}) = A \cap B$. Note that $\mathcal{M}_A$ could loop on $x$ in which case $x$ is clearly not in $A$ or $A \cap B$. So, it does not matter if we do not check whether or not $\mathcal{M}_B$ accepts $x$.

2. Prove that a language $L$ is recursive if and only if there is an enumeration machine enumerating the strings of $L$ in lexicographic order.

   **A:** Suppose that $L \subseteq \Sigma^*$ is a recursive language. Then there exists a total TM $\mathcal{K}$ deciding $L$. Construct an enumeration machine $\mathcal{M}$ that does the following for each string $y$ in $\Sigma^*$ according to the lexicographic order

   - Simulate $\mathcal{K}$ on $y$.
   - Enumerate $y$ if $\mathcal{K}$ accepts $y$
   - If $\mathcal{K}$ rejects $y$, then do not enumerate.

   Since $\mathcal{K}$ is total, the enumeration occurs after finitely many steps.

   Conversely, suppose that there is an enumeration machine $\mathcal{M}$ enumerating strings of a language $L$ in lexicographic order. Define a TM $\mathcal{K}$ that does the following on input $y \in \Sigma^*$,

   - Run $\mathcal{M}$ until it enumerates $y$ or a string that comes after $y$ in a lexicographic order.
   - If $y$ is enumerated, then accept and halt.
   - If the previous string enumerated is different from $y$, then reject and halt.

   Since $\mathcal{M}$ enumerates all strings of $L$ in lexicographic order, it should eventually enumerate $y$ if $y \in L$ or a string that follows $y$ in the lexicographic order. The latter happens when $y \notin L$. As a result, in finite time $\mathcal{K}$ determines if $y \in L$ or not. Therefore, $\mathcal{K}$ is total and $L$ is recursive.

3. One of the following sets is *r.e.* and the other is not. Determine which one is and which one is not. Justify your answers.

(a) $\{\mathcal{M} \mid \mathcal{L}(\mathcal{M})$ contains atleast 300 elements$\}$

**A:** This set is *r.e.* It is possible to construct a TM that runs a given $\mathcal{M}$ all strings in a round-robin fashion. If atleast 300 strings are accepted by $\mathcal{M}$, then halt and accept. Otherwise keep testing for other stings.

(b) $\{\mathcal{M} \mid \mathcal{L}(\mathcal{M})$ contains atmost 300 elements$\}$

**A:** This is a non-monotone property of *r.e.* sets. If an *r.e.* set $A$ contains atmost 300 elements, then it is not necessary that all its supersets contain atmost 300 elements. Hence, it follows by Rice's theorem that the given set is not *r.e.*

4. True or False? It is decidable whether two given TMs accept the same set.

**A:** The problem is to show that the language $\mathsf{EQUIV} = \{\mathcal{M}_1\#\mathcal{M}_2 | \mathcal{L}(\mathcal{M}_1) = \mathcal{L}(\mathcal{M}_2)\}$ is undecidable. We show a reduction from the halting problem i.e., $\mathsf{HP} \leq_m \mathsf{EQUIV}$. Given an instance $\mathcal{M}\#x$ of $\mathsf{HP}$, construct an instance $\mathcal{M}_1\#\mathcal{M}_2$ such that $\mathcal{M}_1$ and $\mathcal{M}_2$ have the following descriptions.

$\mathcal{M}_1$: on input $y$ does the following

- Erase the input $y$.
- Accept and halt.

$\mathcal{M}_2$: on input $y$ does the following

- Erase the input $y$.
- Run $\mathcal{M}$ on $x$.
- Accept and halt if $\mathcal{M}$ halts on $x$.

Clearly, $\mathcal{L}(\mathcal{M}_1) = \Sigma^*$ and

$$\mathcal{L}(\mathcal{M}_2) = \begin{cases} \Sigma^* & \text{if } \mathcal{M} \text{ halts on } x \\ \emptyset & \text{otherwise} \end{cases}$$

That is $\mathcal{M}\#x \in \mathsf{HP}$ iff $\mathcal{M}_1\#\mathcal{M}_2 \in \mathsf{EQUIV}$. The descriptions of $\mathcal{M}_1, \mathcal{M}_2$ can be written down by a total TM. (Note that writing down descriptions of these TMs does not require running $\mathcal{M}$.) Since $\mathsf{HP}$ is undecidable, so is $\mathsf{EQUIV}$.

5. Show that none of the following languages or their complements are *r.e.*

(a) $\mathsf{REG} = \{\mathcal{M} \mid \mathcal{L}(\mathcal{M})$ is a regular set$\}$.

**A:** We use Rice's theorem to show that $\mathsf{REG}$ is undecidable. Let $P_{\mathsf{REG}}$ denote the property on *r.e.* sets defined as

$$P_{\mathsf{REG}}(A) = \begin{cases} \mathsf{T} & \text{if } A \text{ is regular} \\ \mathsf{F} & \text{otherwise} \end{cases}$$

Then deciding this property is equivalent to deciding $\mathsf{REG}$. If a set $A$ is regular it is not necessary that all its supersets are regular. In other words, there exist $A, B$ such that $A \subseteq B$ and $P_{\mathsf{REG}}(A) = \mathsf{T}$, $P_{\mathsf{REG}}(B) = \mathsf{F}$. For instance, we can take $A = \phi$ and $B = \{0^n1^n \mid n \geq 0\}$. This shows that $P_{\mathsf{REG}}$ is a non-monotone property. By Rice's theorem, $P_{\mathsf{REG}}$ is not *r.e.* or equivalently $\mathsf{REG}$ is not *r.e.* Similarly, we can show that $\neg\mathsf{REG}$ or equivalently,

$$P_{\neg\mathsf{REG}}(A) = \begin{cases} \mathsf{T} & \text{if } A \text{ is not regular} \\ \mathsf{F} & \text{otherwise} \end{cases}$$

is not *r.e.* by proving that $P_{\neg\mathsf{REG}}$ is a non-monotone property. We only need to exhibit two sets $A, B$ with $A \subseteq B$ such that $P_{\neg\mathsf{REG}}(A) = \mathsf{T}$ and $P_{\neg\mathsf{REG}}(B) = \mathsf{F}$. Taking $A = \{0^n1^n \mid n \geq 0\}$ and $B = \{0^*1^*\}$ suffices.

(b) $\mathsf{TOT} = \{\mathcal{M} \mid \mathcal{M}$ halts on all inputs$\}$.

**A:** We show TOT is not *r.e.* via a reduction from ¬HP. Given an instance $\mathcal{M}\#x$ of ¬HP, we construct a TM $\mathcal{N}$ such that $\mathcal{M}\#x \in \neg\mathsf{HP}$ iff $\mathcal{N} \in \mathsf{TOT}$. Below is a description of $\mathcal{N}$.
$\mathcal{N}$: on input $y$ does the following.

- Store $y$ on a separate track of the tape.
- Run $\mathcal{M}$ on $x$ for $|y|$ steps. ($\mathcal{M}$ and $x$ are hardwired in $\mathcal{N}$'s finite control.)
- If $\mathcal{M}$ does not halt within $|y|$ steps, then halt and either accept or reject.
- Otherwise, enter a trivial loop.

If $\mathcal{M}$ does not halt on $x$, then $\mathcal{N}$ halts on all input strings. Suppose that $\mathcal{M}$ halts on $x$ in $n$ steps. Then for any string $y$ with $|y| < n$, $\mathcal{N}$ accepts $y$ since $\mathcal{M}$ does not halt on $x$ within $|y|$ steps. On the other hand, for any string $y$ with $|y| \geq n$, $\mathcal{N}$ does not halt. That is,

$$\mathcal{M} \text{ does not halt on } x \implies \mathcal{N} \text{ halts on all input strings} \implies \mathcal{N} \in \mathsf{TOT}$$

$$\mathcal{M} \text{ halts on } x \implies \mathcal{N} \text{ does not halt on some input strings} \implies \mathcal{N} \notin \mathsf{TOT}$$

Since ¬HP is not *r.e.* , TOT is not *r.e.*
Next, we prove that ¬TOT is not *r.e.* through a reduction from ¬HP. Given an instance $\mathcal{M}\#x$ of ¬HP, we construct a TM $\mathcal{N}$ such that $\mathcal{M}\#x \in \neg\mathsf{HP}$ iff $\mathcal{N} \in \neg\mathsf{TOT}$. Below is a description of $\mathcal{N}$.
$\mathcal{N}$: on input $y$ does the following.

- Store $y$ on a separate track of the tape.
- If $|y| \leq |x|$, halt and accept.
- Run $\mathcal{M}$ on $x$.
- If $\mathcal{M}$ halts on $y$, then halt and accept.

If $\mathcal{M}$ does not halt on $x$, then $\mathcal{N}$ halts on all input strings of length at most $|x|$. If $\mathcal{M}$ halts on $x$, then $\mathcal{N}$ accepts any string $y$. That is,

$$\mathcal{M} \text{ does not halt on } x \implies \mathcal{N} \text{ does not halt on some inputs} \implies \mathcal{N} \in \neg\mathsf{TOT}$$

$$\mathcal{M} \text{ halts on } x \implies \mathcal{N} \text{ halts on all input strings} \implies \mathcal{N} \notin \neg\mathsf{TOT}$$

Since ¬HP is not *r.e.* , ¬TOT is not *r.e.*

# Tutorial 1

## Turing Machines and their Variations

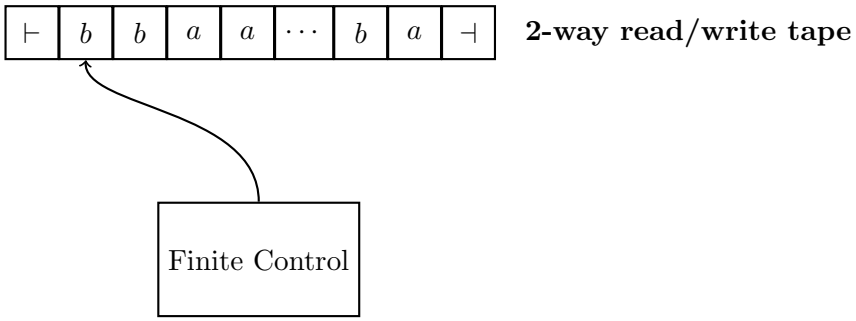
**Guidelines:** Solve all problems in the class. *Do not search for solutions online.*

1. Informally describe a TM that
   (a) inputs a string $\Sigma^*$ and outputs the length of the string in binary.
   (b) accepts the language $\{ww \mid w \in \Sigma^*\}$.

2. Recall the definition of counter automata. A $k$-counter automaton is a machine equipped with a 2-way read-only input tape and $k$ integer counters, each of which can store an arbitrary negative integer and can be incremented, decremented or tested for 0 independently. Show how to simulate a stack using a two counters. Argue that counter automata can simulate Turing machines.

   **A:** Refer to lecture 30 (page 224) of Automata and Computability by Dexter Kozen for a discussion on this. □

3. A linear bounded automaton (LBA) is exactly like a 1-tape TM, except that the input string $x \in \Sigma^*$ is enclosed in left and right endmarkers $\vdash$ and $\dashv$ which may not be overwritten. The machine is constrained never to move left of $\vdash$ or right of $\dashv$. It is allowed to read/write between these markers.




   (a) Give a rigorous formal definition of deterministic linearly bounded automata, including a definition of configurations and acceptance.

   **A:** An LBA is given by a 10-tuple, $(Q, \Sigma, \Gamma, \delta, \vdash, \dashv, \sqcup, s, t, r)$ where

   - $Q$ is the set of states
   - $\Sigma$ is the input alphabet
   - $\Gamma$ is the tape alphabet
   - $\delta : Q \times \Gamma \to Q \times \Gamma \times \{L, R\}$ is the transition function
   - $\vdash, \dashv \in \Gamma \setminus \Sigma$ are the left and right endmarkers respectively
   - $\sqcup \in \Gamma \setminus \Sigma$ is the blank symbol
   - $s, t, r$ are respectively the start, accept and reject states

   Restrictions that ensure that the Turing machine never overwrites the endmarkers and never move left or right of the left and right endmarkers are given below.

   - $\forall p \in Q$, $\exists q \in Q$ such that $\delta(p, \vdash) = (q, \vdash, R)$.
   - $\forall p \in Q$, $\exists q \in Q$ such that $\delta(p, \dashv) = (q, \dashv, L)$.

   □

(b) Let $\mathcal{M}$ be an LBA with state set $Q$ of size $k$ and tape alphabet $\Gamma$ of size $m$. How many possible configurations are there on input $x$ with $|x| = n$?

**A:** Let $[n_1, n_2]$ for $n_2 > n_1$ denote the set of integers $n_1, n_1 + 1, \ldots, n_2$. There are a total of $m$ possible symbols that can be written on tape cells $1, \ldots, n$. At positions 0 and $n + 1$, the symbols are fixed. The LBA can be in $k$ possible states. The tape head can be at one of $n + 2$ positions. Hence, the set of all possible configurations is given by $[0, n + 2] \times Q \times \Gamma^n$. The number of configurations possible is therefore $(n + 2) \cdot k \cdot m^n$.

(c) Argue that it is possible to detect in finite time whether an LBA loops on a given input.
**Hint:** Use part (b).

**A:** Run the LBA for $k(n + 2)m^n + 1$ steps. If any configuration repeats, then output that the LBA loops. Correctness follows from the fact that the LBA is deterministic.

(d) Prove by diagonalisation that there exists a recursive set that is not accepted by any LBA.

**A:** Similar to TMs, we can define an encoding using bit strings for LBAs. Let $\mathcal{M}_\varepsilon, \mathcal{M}_0, \mathcal{M}_1, \mathcal{M}_{00}, \mathcal{M}_{01}, \ldots$ denote LBA's described according to this encoding. If a string $x$ is not a valid description of an LBA, set $\mathcal{M}_x$ to be a trivial LBA that accepts and halts on all inputs. Construct an infinite matrix $\mathbf{A} = (a_{x,y})_{x,y=\varepsilon,0,1,00,01,\ldots}$ as

$$a_{x,y} = \begin{cases} 1 & \text{if } y \in \mathcal{L}(\mathcal{M}_x) \\ 0 & \text{if } y \notin \mathcal{L}(\mathcal{M}_x) \end{cases}$$

Now consider a set $B$ defined as $B = \{x \mid x \notin \mathcal{L}(\mathcal{M}_x)\}$. The characteristic function for this set corresponds to the complement of the diagonal of $\mathbf{A}$. As for each $x \in \{0, 1\}^*$, $B$ differs from $\mathcal{L}(\mathcal{M}_x)$ atleast on $x$, no LBA accepts $B$. We now prove that $B$ is recursive by building a total TM $\mathcal{N}$ accepting $B$. Below is the description of $\mathcal{N}$.

$\mathcal{N}$: on input $y$

- Construct $\mathcal{M}_y$ from $y$.
- Simulate $\mathcal{M}_y$ on input $y$.
- If $\mathcal{M}_y$ accepts and halts, then reject and halt.
- If $\mathcal{M}_y$ rejects or loops, then accept and halt.

Using the fact that it is possible to check whether an LBA loops or not in finite time, we conclude that the TM $\mathcal{N}$ halts on all inputs and is hence total.

**Note:** Here I have indexed the automata with strings. You can use the natural numbers as index and refer to the LBA corresponding to a string $x$ as $\mathcal{M}_{\eta(x)}$ where $\eta : \{0, 1\}^* \to \mathbb{N}$ is an injective map. Such a map exists since $\{0, 1\}^*$ is a countable set. □

# Tutorial 5
## Recursion Theorems

**Guidelines:** Solve all problems in the class. *Do not search for solutions online.*

**Quick Recap of Notation:** $f_i(\cdot)$ denotes the function computed by the TM with encoding $i \in \mathbb{N}$. For a TM $\mathcal{M}$, $\langle \mathcal{M} \rangle$ denotes its index.

1. Prove that there exists $x_0 \in \mathbb{N}$ such that for all $y$,

$$f_{x_0}(y) = \begin{cases} y^2 & \text{if } y \text{ is even} \\ f_{x_0+1}(y) & \text{otherwise} \end{cases}$$

**Solution:** There exists a partial recursive function $g$ in two variables such that

$$g(x,y) = \begin{cases} y^2 & \text{if } y \text{ is even} \\ f_{x+1}(y) & \text{otherwise} \end{cases}$$

Let $\mathcal{M}$ be a TM that does the following on input $x, y$: check if $y$ is even; if so write $y^2$ on the tape and halt; otherwise simulate the TM with index $x+1$ on input $y$. Clearly $\mathcal{M}$ computes $g(x,y)$.

By $S_{mn}$-theorem, there exists a total recursive function $\sigma : \mathbb{N} \to \mathbb{N}$ such that for all $y$,

$$f_{\sigma(x)}(y) = g(x,y).$$

$\sigma$, being total recursive, has a fixed point $x_0$ guaranteed by the recursion theorem. We therefore have

$$f_{x_0} = f_{\sigma(x_0)}(y) = g(x_0,y) = \begin{cases} y^2 & \text{if } y \text{ is even} \\ f_{x_0+1}(y) & \text{otherwise} \end{cases},$$

for all $y$.

2. Define any fixed point for the total recursive function $\sigma : \mathbb{N} \to \mathbb{N}$ defined as follows: for $x \in \mathbb{N}$, the TM with description $\sigma(x)$ computes the function

$$f_{\sigma(x)}(y) = \begin{cases} 1 & \text{if } y = 0 \\ f_x(y+1) & \text{otherwise} \end{cases}$$

Describe a fixed point for $\sigma$.

**Solution:** Let $\mathcal{M}$ be a TM that on input $y \in \mathbb{N}$ outputs 1 if $y = 0$ and outputs a constant $a \in \mathbb{N}$ otherwise. Then $\hat{x} = \langle \mathcal{M} \rangle$ is a fixed point for $\sigma$, as justified below.

For $y = 0$, we have

$$f_{\hat{x}}(y) = 1 = f_{\sigma(\hat{x})}(y)$$

and otherwise, we have

$$f_{\hat{x}}(y) = a = f_{\hat{x}}(y+1) = f_{\sigma(\hat{x})}(y).$$

3. Let $\sigma : \mathbb{N} \to \mathbb{N}$ be any total recursive function. Prove that $\sigma$ has infinitely many fixed points i.e., there are infinitely many $w \in \mathbb{N}$ such that $f_w(y) = f_{\sigma(w)}(y)$ for all $y$.
**Hint:** Recursion theorem ensures there is atleast one fixed point for 'any' total recursive function. If the set of fixed points is finite, does it contradict recursion theorem?

**Solution:** Suppose there exists a total recursive function $\sigma$ with finitely many fixed points. Let the set of fixed points be denoted $\mathcal{F}$. Let $g$ be a partial recursive function such that the indices of all TMs computing $g$ are outside $\mathcal{F}$. That is for all TMs $\mathcal{M}$ computing $g$, $\langle \mathcal{M} \rangle \notin \mathcal{F}$. In other words, for all TMs $\mathcal{M}$ computing $g$, $f_{\langle \mathcal{M} \rangle} \neq f_w$ for every $w \in \mathcal{F}$.

Let $u$ be an index of some TM computing $g$. Now, define a function $\tau : \mathbb{N} \to \mathbb{N}$ implicitly so that

$$\tau(x) = \begin{cases} u & \text{if } x \in \mathcal{F} \\ \sigma(x) & \text{otherwise} \end{cases}$$

Observe that $\tau$ is total recursive:

- For any $x \in \mathbb{N}$, check whether $x \in \mathcal{F}$. This can be done in finite time since $\mathcal{F}$ is a finite set.
- If $x \in \mathcal{F}$, then set $\tau(x) = u$; otherwise compute $\sigma(x)$ (which is total recursive) and assign the resulting value to $\tau(x)$.

We now argue that $\tau$ has no fixed point. If $x \in \mathcal{F}$, then $\tau(x) = u$ and since $f_u \neq f_w$ for every $w \in \mathcal{F}$, we have (in particular) $f_{\tau(x)} \neq f_x$. Now suppose $x \notin \mathcal{F}$. Then $f_{\tau(x)} = f_{\sigma(x)} \neq f_x$. Combining the two, we have $f_{\tau(x)} \neq f_x$ for every $x \in \mathbb{N}$ thus implying that $\tau$ has no fixed points. This contradicts the recursion theorem. Hence, any total recursive function must have inifinitely many fixed points.

4. Let $\mathcal{M}_x$ denote the Turing machine with index $x \in \mathbb{N}$. Here's a statement of the recursion theorem, specialised to language recognisers:

"Let $\sigma : \mathbb{N} \to \mathbb{N}$ be any total recursive function. Then there exists $x_0 \in \mathbb{N}$ such that $L(\mathcal{M}_{x_0}) = L(\mathcal{M}_{\sigma(x_0)})$."

Use it to provide an alternate proof of Rice's theorem (part I).

**Solution:** Let $P$ be any non-trivial property of *r.e.* sets. Then there exist indices $u, v$ such that $P(L(\mathcal{M}_u)) = \mathsf{T}$ and $P(L(\mathcal{M}_v)) = \mathsf{F}$. Assume, for the sake of contradiction, that $P$ is decidable. Define a function $\sigma : \mathbb{N} \to \mathbb{N}$ as follows:

$$\sigma(x) = \begin{cases} u & \text{if } P(L(\mathcal{M}_x)) = \mathsf{F} \\ v & \text{otherwise} \end{cases}$$

By our assumption, $\sigma$ is a total recursive function. The recursion theorem implies that $\sigma$ has a fixed point $x_0$ with $L(\mathcal{M}_{x_0}) = L(\mathcal{M}_{\sigma(x_0)})$. Now, if $P(L(\mathcal{M}_{x_0})) = \mathsf{F}$, we have

$$\mathsf{T} = P(L(\mathcal{M}_{x_0})) = P(L(\mathcal{M}_{\sigma(x_0)})) = P(L(\mathcal{M}_v)) = \mathsf{F},$$

thus contradicting our assumption that $P$ is decidable. Therefore, $P$ is undecidable.

**Food for thought:** Does a generalisation of Rice's theorem hold for partial recursive functions? That is, can you show that any non-trivial property of the set of partial recursive functions is undecidable?

# Tutorial 3

## Valid and Invalid Computations of a Turing Machine

**Guidelines:** Solve all problems in the class. *Do not search for solutions online.*

1. Define a set $\mathsf{VALC\text{-}R}_{\mathcal{M},x}$ to be the set of all strings of the form $\#\alpha_0\#\alpha_1^{\mathbf{R}}\#\alpha_2\#\alpha_3^{\mathbf{R}}\#\cdots\#\alpha'_N\#$, where $\alpha'_N = \alpha_N^{\mathbf{R}}$ if $N$ is odd and $\alpha'_N = \alpha_N$ otherwise, where $\#\alpha_0\#\alpha_1\#\cdots\#\alpha_N\#$ is a valid computation history of $\mathcal{M}$ on input $x$. That is

$$\#\alpha_0\#\alpha_1\#\cdots\#\alpha_N\# \in \mathsf{VALCOMPS}_{\mathcal{M},x} \Leftrightarrow \#\alpha_0\#\alpha_1^{\mathbf{R}}\#\alpha_2\#\alpha_3^{\mathbf{R}}\#\cdots\#\alpha'_N\# \in \mathsf{VALC\text{-}R}_{\mathcal{M},x}.$$

Show that $\mathsf{VALC\text{-}R}_{\mathcal{M},x}$ can be expressed as the intersection of two context-free languages.
**Hint:** Consider checking two possibilities for $\alpha_i^{\mathbf{R}} \xrightarrow[\mathcal{M}]{1} \alpha_{i+1}$ – odd $i$ and even $i$.

**A:** Let

$$L_1 = \{\#\alpha_0\#\alpha_1^{\mathbf{R}}\#\alpha_2\#\alpha_3^{\mathbf{R}}\#\cdots\#\alpha'_N\# \mid \alpha_i \xrightarrow[\mathcal{M}]{1} \alpha_{i+1} \text{ for odd } i\}$$

and

$$L_2 = \{\#\alpha_0\#\alpha_1^{\mathbf{R}}\#\alpha_2\#\alpha_3^{\mathbf{R}}\#\cdots\#\alpha'_N\# \mid \alpha_i \xrightarrow[\mathcal{M}]{1} \alpha_{i+1} \text{ for even } i\}.$$

Clearly, $L_1 \cap L_2$ consists of strings of the form $\#\alpha_0\#\alpha_1^{\mathbf{R}}\#\alpha_2\#\alpha_3^{\mathbf{R}}\#\cdots\#\alpha'_N\#$ with both conditions $\alpha_i \xrightarrow[\mathcal{M}]{1} \alpha_{i+1}$ for odd $i$ and $\alpha_i \xrightarrow[\mathcal{M}]{1} \alpha_{i+1}$ for even $i$, being satisfied. That is, $\alpha_0 \xrightarrow[\mathcal{M}]{1} \alpha_1 \xrightarrow[\mathcal{M}]{1} \alpha_2 \xrightarrow[\mathcal{M}]{1} \cdots \xrightarrow[\mathcal{M}]{1} \alpha_N$ and so we have $\#\alpha_0\#\alpha_1\#\alpha_2\#\alpha_3\#\cdots\#\alpha_N\# \in \mathsf{VALCOMPS}_{\mathcal{M},x}$ or equivalently $\#\alpha_0\#\alpha_1^{\mathbf{R}}\#\alpha_2\#\alpha_3^{\mathbf{R}}\#\cdots\#\alpha'_N\# \in \mathsf{VALC\text{-}R}_{\mathcal{M},x}$ implying that $L_1 \cap L_2 = \mathsf{VALC\text{-}R}_{\mathcal{M},x}$. We now build NPDAs $\mathcal{N}_1, \mathcal{N}_2$ accepting $L_1, L_2$ respectively, thus showing that $L_1, L_2$ are CFLs.

$\mathcal{N}_1$, on input a string $\#\alpha_0\#\alpha_1^{\mathbf{R}}\#\alpha_2\#\alpha_3^{\mathbf{R}}\#\cdots\#\alpha'_N\#$ does the following.

1. Go to the next odd position $i$, scanning (and ignoring) the input string until the # symbol just before $\alpha_i^{\mathbf{R}}$.
2. While reading $\alpha_i^{\mathbf{R}}$, examine $\delta$ to determine (reverse of) the next configuration of $\mathcal{M}$ following $\alpha_i$. (This can be done using constant amount of memory in the finite control as $\alpha_i$ would differ from its next configuration in atmost three positions. Moreover, $\mathcal{M}$ is deterministic and hence there is atmost one possibility for the next configuration.) Let $\bar{\alpha}_{i+1}$ denote the subsequent configuration. Push $\bar{\alpha}_{i+1}^{\mathbf{R}}$ on the stack. This can be done as $\bar{\alpha}_{i+1}$ is determined in the reverse order, when corresponding symbols in $\alpha_i^{\mathbf{R}}$ are read from the tape.
3. Start reading $\alpha_{i+1}$ – pop the stack one symbol at a time for each input symbol read in order to verify that $\alpha_{i+1} = \bar{\alpha}_{i+1}$.
4. Reject and if there is a mismatch.
5. If end of string is reached, accept. Otherwise, go back to step 1.

The construction of $\mathcal{N}_2$ is similar. □

2. Prove that it is undecidable whether

   (a) the intersection of two given CFLs is empty.
   **Hint:** Think VALCOMPS or VALC-R.

   **A:** Let EI-CFL $= \{G_1, G_2 \mid G_1, G_2$ are CFGs and $L(G_1) \cap L(G_2) = \emptyset\}$. We show a reduction from $\neg$HP to EI-CFL. Let $\mathcal{M}, x$ be an instance of $\neg$HP. Construct CFGs $G_1, G_2$ such that $L(G_1) \cap L(G_2) =$ VALC-R$_{\mathcal{M},x}$. If $\mathcal{M}$ does not halt on $x$, then VALC-R$_{\mathcal{M},x} = \emptyset$ and hence $L(G_1) \cap L(G_2) = \emptyset$. If $\mathcal{M}$ halts on $x$, then VALC-R$_{\mathcal{M},x} = L(G_1) \cap L(G_2) \neq \emptyset$. Since $\neg$HP is undecidable, EI-CFL is also undecidable.

   (b) the intersection of two given CFLs is a CFL.

   **A:** The same reduction as in the previous problem works since $\emptyset$ is a CFL and VALCOMPS is not a CFL.

   (c) the complement of a given CFL is a CFL.

   **A:** We reduce $\neg$HP to $\{G \mid G$ is a CFG and $\neg L(G)$ is a CFL$\}$. Given $\mathcal{M}, x$, construct a CFG $G$ such that $L(G) = \neg$VALCOMPS$_{\mathcal{M},x}$. If $\mathcal{M}$ does not halt on $x$, then $L(G) = \Sigma^*$ and $\neg L(G) = \emptyset$ both of which are CFLs. If $\mathcal{M}$ halts on $x$, then $L(G)$ is a CFL but its complement $\neg L(G) =$ VALCOMPS$_{\mathcal{M},x}$ is not.

# Tutorial 4

## PCP, Undecidable Problems in Language Theory and More

**Guidelines:** Solve all problems in the class. *Do not search for solutions online.*

1. Consider a silly variant of PCP called SPCP where corresponding strings in both lists are restricted to have the same length. Show that this variant is decidable.

   **Solution:** Let $A = \{w_1, \ldots, w_n\}$ and $B = \{x_1, \ldots, x_n\}$ denote an instance of SPCP. If there exists a solution, then there is an index $j \in [1, n]$ such that the solution starts with $w_j, x_j$. Let $|w_j| = |x_j| = \ell$. Since there is a match in the first $\ell$ positions, it must be the case that $w_j = x_j$. Also, if there is an index $j$ such that $w_j = x_j$, then $j$ is a solution to the SPCP instance $(A, B)$. Therefore, SPCP can be decided by just checking whether for each $j \in [1, n]$, $w_j = x_j$.

2. Prove that $\{G_1, G_2 \mid G_1, G_2 \text{ are CFGs and } \mathcal{L}(G_1) \cap \mathcal{L}(G_2) \neq \emptyset\}$ is undecidable via a reduction from PCP.

   **Solution:** Let $A = \{w_1, \ldots, w_k\}$ and $B = \{x_1, \ldots, x_k\}$ denote an instance of PCP over alphabet $\Sigma$. Let $\Sigma' = \Sigma \cup \{a_1, \ldots, a_k\}$ for some new symbols $a_1, \ldots, a_k \notin \Sigma$. Define two CFGs $G_A = (\{S_A\}, \Sigma', P_A, S_A)$ and $G_B = (\{S_B\}, \Sigma', P_B, S_B)$ where $P_A$ consists of the productions

   $$S_A \to w_i S_A a_i \mid w_i a_i \text{ for } 1 \leq i \leq k,$$

   and $P_B$ consists of

   $$S_B \to x_i S_B a_i \mid x_i a_i \text{ for } 1 \leq i \leq k.$$

   Suppose the PCP instance $(A, B)$ has a solution $i_1, \ldots, i_m$. Let $y = w_{i_1} \cdots w_{i_m} = x_{i_1} \cdots x_{i_m}$. Then $y a_{i_m} \cdots a_{i_1} \in \mathcal{L}(G_A)$ and $y a_{i_m} \cdots a_{i_1} \in \mathcal{L}(G_B)$. As a result $\mathcal{L}(G_A) \cap \mathcal{L}(G_B) \neq \emptyset$.

   Now, suppose that $\mathcal{L}(G_A) \cap \mathcal{L}(G_B) \neq \emptyset$. Then there is a string $y a_{i_m} \cdots a_{i_1} \in \mathcal{L}(G_A) \cap \mathcal{L}(G_B)$. Since $y a_{i_m} \cdots a_{i_1} \in \mathcal{L}(G_A)$ it must be the case that $y = w_{i_1} w_{i_2} \cdots w_{i_m}$. Also, $y$ must be equal to $x_{i_1} x_{i_2} \cdots x_{i_m}$ since $y a_{i_m} \cdots a_{i_1} \in \mathcal{L}(G_B)$. Then $i_1, \ldots, i_m$ forms a solution to PCP instance $(A, B)$.

3. Show that PCP is undecidable over the binary alphabet $\{0, 1\}$.

   **Solution:** Denote PCP over alphabet $\{0, 1\}$ as BPCP. We show that PCP $\leq_{\mathsf{m}}$ BPCP. Let $A = \{w_1, \ldots, w_n\}$ and $B = \{x_1, \ldots, x_n\}$ denote an instance of PCP over some alphabet $\Sigma$. Let $s = |\Sigma|$ and $\Sigma = \{a_1, \ldots, a_s\}$. Define a map $f : \Sigma \to \{0, 1\}^*$ as $f(a_i) = 0^i 1$ for $i \in [1, s]$. Extend this map to strings over $\Sigma^*$ as $F : \Sigma^* \to \{0, 1\}^*$ where for any string $y = a_{i_1} a_{i_2} \cdots a_{i_k}$, $F(y) = f(a_{i_1}) f(a_{i_2}) \cdots f(a_{i_k})$. Observe that PCP instance $(A, B)$ has a solution iff the BPCP instance $(\{F(w_1), \ldots, F(w_n)\}, \{F(x_1), \ldots, F(x_n)\})$ has a solution, when $F$ is one-one. Suppose that $F(y) = F(z)$ for some $y, z \in \Sigma^*$. Let $y = a_{i_1} \cdots a_{i_k}$ and $z = a_{j_1} \cdots a_{j_\ell}$ for some $i_1, \ldots, i_k, j_1, \ldots, j_\ell \in [1, n]$, then $F(y) = f(a_{i_1}) f(a_{i_2}) \cdots f(a_{i_k}) = 0^{i_1} 1 0^{i_2} 1 \cdots 0^{i_k} 1 = f(a_{j_1}) f(a_{j_2}) \cdots f(a_{j_\ell}) = F(z)$. If $y \neq z$ Let $r$ be the minimum integer such that $a_{i_r} \neq a_{j_r}$. Then $f(a_{i_r}) = 0^{i_r} 1 \neq 0^{j_r} 1 = f(a_{j_r})$ but then this implies that $F(y) \neq F(x)$ contradicting our assumption. Therefore $x = z$ and as a consequence $F$ is 1-1.

4. Show that the language $\mathsf{PF} = \{\mathsf{G} \mid \mathsf{G} \text{ is a CFG and } \mathsf{L}(\mathsf{G}) \text{ is prefix-free}\}$ is undecidable.

**Solution:** We know that PCP is undecidable. This implies ¬PCP is undecidable as well.

We describe a reduction $\neg\mathsf{PCP} \leq_{\mathsf{m}} \mathsf{PF}$. Let $A = (w_1, w_2, \ldots, w_k)$ and $B = (x_1, x_2, \ldots, x_k)$ be an instance of ¬PCP defined over alphabet $\Sigma$. Let $a_1, a_2, \ldots, a_k, \#, \dashv \notin \Sigma$ be $k+1$ new distinct symbols and let $\Sigma' = \Sigma \cup \{a_1, \ldots, a_k, \#, \dashv\}$. Define a context-free grammar $G = (N = \{S, S_A, S_B\}, \Sigma', P, S)$ where $P$ consists of the following productions:

$$S \to S_A \# \dashv \mid S_B \#,$$

$$S_A \to w_i S_A a_i \mid w_i a_i \quad \text{for } 1 \leq i \leq k,$$

$$S_B \to x_i S_A a_i \mid x_i a_i \text{ for } 1 \leq i \leq k.$$

Observe that all strings derived from $S \to S_A \# \dashv$ end with $\# \dashv$ and all strings derived from $S \to S_B \#$ end with $\#$. Suppose there are distinct strings $u, v \in L(G)$ such that $u$ is a prefix of $v$. Then all symbols in $u$ and $v$ upto and including $\#$ must match. That is, we can write $u = u'\#$, $v = v'\# \dashv$ such that $u' = v'$, $S_A \xrightarrow[G]{*} v'$ and $S_B \xrightarrow[G]{*} u'$.

We now show that $(A, B) \in \neg\mathsf{PCP}$ iff $L(G)$ is prefix-free. Suppose that $(A, B) \notin \neg\mathsf{PCP}$. For a solution $i_1, i_2, \ldots, i_m$, we have $w_{i_1} w_{i_2} \cdots w_{i_m} = x_{i_1} x_{i_2} \cdots x_{i_m}$. Let $z = w_{i_1} w_{i_2} \cdots w_{i_m} a_{i_m} \cdots a_{i_2} a_{i_1} = x_{i_1} x_{i_2} \cdots x_{i_m} a_{i_m} \cdots a_{i_2} a_{i_1}$. By definition, $L(G)$ contains both $z\# \dashv$ and $z\#$ and hence is not prefix-free. Suppose that $\exists u, v \in L(G)$ such that $u$ is a prefix of $v$. Then, $u = u'\#$, $v = v'\# \dashv$ such that $u' = v'$, $S_A \xrightarrow[G]{*} v'$ and $S_B \xrightarrow[G]{*} u'$. The string $v'$, derived from $S_A$, must be of the form $w_{i_1} w_{i_2} \cdots w_{i_m} a_{i_m} \cdots a_{i_2} a_{i_1}$. Similarly, $u'$ has the form $x_{i_1} x_{i_2} \cdots x_{i_m} a_{i_m} \cdots a_{i_2} a_{i_1}$. The $a_i$'s at the end must all match since $u' = v'$. As a result, $w_{i_1} w_{i_2} \cdots w_{i_m} = x_{i_1} x_{i_2} \cdots x_{i_m}$ implying that $i_1, i_2, \ldots, i_m$ is a solution for $(A, B)$ i.e., $(A, B) \notin \neg\mathsf{PCP}$. We have shown that $(A, B) \notin \neg\mathsf{PCP} \Leftrightarrow \mathsf{G} \notin \mathsf{PF}$ from which it follows that $(A, B) \in \neg\mathsf{PCP} \Leftrightarrow \mathsf{G} \in \mathsf{PF}$ and therefore PF is undecidable.

5. For $A, B \subseteq \Sigma^*$, define

$$A/B = \{x \in \Sigma^* \mid \exists y \in B \quad xy \in A\}.$$

(a) Show that if $A$ and $B$ are recursively enumerable, then so is $A/B$.

**Solution:** Let $\mathcal{M}_A, \mathcal{M}_B$ be Turing machines accepting $A, B$ respectively. Define a TM $\mathcal{N}$ that on input $x$ does the following.

- For each $y \in \Sigma^*$, simulate $\mathcal{M}_B$ on $y$ on a time-shared basis. That is, simulate $\mathcal{M}_B$ on $y_1$ for one step and then simulate it on $y_2$ for one step and continue simulations for some fixed ordering $y_1, y_2, \ldots$ of strings in $\Sigma^*$.
- If $\mathcal{M}_B$ accepts, then simulate $\mathcal{M}_A$ on $xy$.
- Halt and accept if $\mathcal{M}_A$ accepts.

If $x \in A/B$, then for some $y \in \Sigma^*$, $\mathcal{M}_B$ accepts $y$ eventually and $\mathcal{M}_A$ accepts $xy$. Hence $A/B$ is recursively enumerable.

(b) Show that every *r.e.* set can be represented as $A/B$ with $A$ and $B$ being context-free languages.

**Solution:** Let $R$ be an *r.e.* set and let $\mathcal{M} = (Q, \Sigma, \Gamma, \delta, \vdash, \sqcup, s, t, r)$ be a Turing machine accepting it. Recall that we defined $\mathsf{VALCOMPS}_{\mathcal{M},c}$ over the alphabet $\Delta = \{\#\} \cup (\Gamma \times (Q \cup \{-\}))$. For $x = a_1 a_2 \cdots a_n$, let $S_{\mathcal{M},x}$ be the starting configuration, given by

$$\begin{array}{cccccc} \vdash & a_1 & a_2 & \cdots & a_n \\ s & - & - & & - \end{array}.$$

We now define the sets $A$ and $B$ over the alphabet $\Sigma \cup \Delta$ as follows:

$$A = \left\{ x\#S_{\mathcal{M},x}\#\alpha_1^{\mathbf{R}}\#\alpha_2\#\cdots\#\alpha'_N \mid \alpha_i \xrightarrow[\mathcal{M}]{1} \alpha_{i+1} \text{ for all odd } i \right\}$$

$$B = \left\{ \#\alpha_0\#\alpha_1^{\mathbf{R}}\#\alpha_2\#\cdots\#\alpha'_N \mid \alpha_i \xrightarrow[\mathcal{M}]{1} \alpha_{i+1} \text{ for all even } i \text{ and } \alpha_N \text{ contains } t \right\}$$

Here, $\alpha'_N = \alpha_N^{\mathbf{R}}$ if $N$ is odd and $\alpha'_N = \alpha_N$ otherwise.
Convince yourself that $R = A/B$!

6. Let

$$f(x) = \begin{cases} 3x+1 & \text{if } x \text{ is odd} \\ x/2 & \text{if } x \text{ is even} \end{cases}$$

for any natural number $x$. Define $\mathsf{C}(x)$ as the sequence $x, f(x), f(f(x)), \ldots$, which terminates if and when it hits 1. For example, if $x = 7$, then

$$\mathsf{C}(x) = (7, 22, 11, 34, 17, 52, 26, 13, 40, 20, 10, 5, 16, 8, 4, 2, 1).$$

Computer tests have shown that $\mathsf{C}(x)$ hits 1 eventually for $x$ ranging from 1 to $87 \times 2^{60}$ (as of 2017). But, the question of whether $\mathsf{C}(x)$ ends at 1 for all $x \in \mathbb{N}$ is not proven. This is believed to be true and known as the Collatz conjecture. Suppose that $\mathsf{MP}$ were decidable by a Turing machine $\mathcal{K}$. Use $\mathcal{K}$ to describe a TM that is guaranteed to prove or disprove Collatz conjecture.

**Solution:** Let $\mathcal{N}$ be a TM, that on input $x$, does the following.

- while $x \neq 1$,
  - if $x$ is even, $x \leftarrow x/2$
  - otherwise $x \leftarrow 3x + 1$
- accept and halt

Let $\mathcal{M}$ be a TM, that on input $y$, does the following.

- erase y
- set $x \leftarrow 1$ and repeat the following
  - use $\mathcal{K}$ to determine whether $\mathcal{N}$ accepts $x$
  - if not, accept and halt
  - otherwise, $x \leftarrow x + 1$

If the Collatz conjecture is true, then $\mathcal{M}$ runs forever; otherwise $\mathcal{M}$ halts after finding a counter example (in fact, the smallest counter example).

We now use $\mathcal{K}$ to determine whether or not $\mathcal{M}$ accepts an arbitrary input $y$ to decide whether or not Collatz conjecture is true.