

Computability, exam preparation

June 18, 2010

By Nerya Or.

Version 0.93

Based on lectures by Orna Kupferman and TA classes by Ezra Hoch.

This is in no way a complete document, and contains just general highlights of what we've seen in the course.

The contents herein may contain errors, use at your own responsibility...

Contents

I	Highlights of the lectures and TA sessions	2
1	Automatons and formal languages	2
1.1	Automatons and regular languages	2
1.2	Context-free languages	3
2	Computability	4
3	Complexity	6
3.0.1	Time complexity	6
3.0.2	Space complexity	7
3.0.3	The classes <i>LOGSPACE</i> , <i>NLOGSPACE</i>	8
II	Counter examples, etc.	10

Part I

Highlights of the lectures and TA sessions

1 Automata and formal languages

1.1 Automata and regular languages

- A DFA is a tuple $\langle Q, \Sigma, \delta, q_0, F \rangle$ where Q are the states, Σ is the alphabet, $\delta : Q \times \Sigma \rightarrow Q$, q_0 is the start state, and $F \subseteq Q$ is the end state set.
- An NFA is a tuple $\langle Q, \Sigma, \delta, Q_0, F \rangle$ where Q are the states, Σ is the alphabet, $\delta : Q \times (\Sigma \cup \{\varepsilon\}) \rightarrow 2^Q$, $Q_0 \subseteq Q$ is the start state set, and $F \subseteq Q$ is the end state set.
- Any NFA A has an equivalent NFA A' , with no ε steps. (Tirgul, 24.2).
- **The class REG is closed under union, intersection, complement, concatenation and star operations.**
- Regular expressions: $\emptyset, \sigma, \varepsilon$ are all valid, basic, regular expressions. In addition, $+, \cdot, *$ are defined recursively as regular expressions.
- Theorem: $NFA = DFA = REG$.
 - For proving that $REG \subseteq NFA$, we built automata for regular expressions.
 - To prove the other direction ($DFA \subseteq REG$), we've used a $GNFA$, which has one single start state with no incoming edges, one single accept state with no exiting edges, and in between **every** two states (including a state and itself), there's an edge with a regular expression on it. We turn a DFA to a GNFA, and then reduce its number of states iteratively until we have a GNFA with 2 states, and the desired regular expression is written on the edge between them. The exact procedure was shown in the Tirgul, 10.3.
- **Pumping lemma** for regular languages:
 - Let L be a regular language. Thus, there exists $p \geq 1$ such that for all $w \in L$, if $|w| \geq p$, there exists a partition $w = xyz$ such that:
 - * $|xy| \leq p$
 - * $|y| > 0$
 - * $\forall i \geq 0, xy^i z \in L$
- **Myhill-Nerode** theorem:
 - Given a language L , we define $x \sim_L y \Leftrightarrow \forall z \in \Sigma^* [xz \in L \Leftrightarrow yz \in L]$.
 - L is regular **iff** the number of equivalence classes in \sim_L is finite.
 - **Very important result:** For a DFA A , if we let $|\sim_L|$ be the number of equivalence classes in \sim_L and Q be the set of states in A , it holds that $|\sim_L| \leq |Q|$.
- It is possible to build a “minimal” automaton A' from a DFA A , such that the number of states in A' is equal to the number of equivalence classes in $\sim_{L(A)}$. The procedure for building such a minimized automaton is polynomial (about $\mathcal{O}(|Q|^3 \cdot |\Sigma|)$). As seen in Tirgul on 17.3.
- Decision problems:
 - Emptiness: Given an NFA A , we ask, $L(A) = \emptyset$? The answer can be obtained by running a BFS on the automaton to check reachability of the accept state. This is polynomial.

- Universality: Given an NFA A , we ask, $L(A) = \Sigma^*$? The answer can be obtained by calculating \overline{A} (convert to DFA and switch accept/non accept states. This is exponential), and then checking if $L(\overline{A}) = \emptyset$. This is, as we've seen, exponential.
- Containment: Given two NFAs A_1, A_2 , we ask, $L(A_1) \subseteq L(A_2)$? It's at least as hard as both the emptiness problem (take $L_2 = \emptyset$) and the universality problem (take $L_1 = \Sigma^*$). Since we know $X \subseteq Y \Leftrightarrow X \cap \overline{Y} = \emptyset$, we can check if $L(A_1) \cap \overline{L(A_2)} = \emptyset$. We do this by getting $\overline{A_2}$ (exp. time) and calculating the $A_1 \times \overline{A_2}$ automaton. This is polynomial in $|A_1|$ and exponential in $|A_2|$.
- Let A be an NFA, and let $w \in \Sigma^*$. We wish to know if $w \in L(A)$.
 - Create a DFA B whose language is only w (easy to do).
 - Calculate the intersection $A \cap B$ (the product automaton).
 - Check emptiness of the new automaton.
 - This is all done in poly-time.
- For a language L , define:
 - $Pref(L) = \{x \mid \text{there exists } y \in \Sigma^* \text{ such that } x \cdot y \in L\}$
 - $Suff(L) = \{x \mid \text{there exists } y \in \Sigma^* \text{ such that } y \cdot x \in L\}$
 - If L is regular, then so are $Pref(L)$ and $Suff(L)$.
 - Proof: To prove $Pref(L) \in REG$, we can build an automaton that accepts it, reversing all arrows and running BFS from the accept state, marking all reachable states as accepting states in the new automaton, which has the edges back in their original direction. To prove $Suff(L) \in REG$, we note that $[Pref(L^R)]^R = Suff(L)$, and remember that if $L \in REG$ then $L^R \in REG$.
- A DFA with n states accepts an infinite language iff it accepts a word w such that $n \leq |w| \leq 2n$.
 - Proof: Ex3q6. Basically, we have repeating states for a word that's too long, which means we can play with the run loops like in the pumping lemma proof. Need to show two directions...

1.2 Context-free languages

- A CFG (context-free grammar) is a tuple $\langle V, \Sigma, R, S \rangle$, where V is the set of variables, Σ is the alphabet/terminals, R are the rules, and $S \in V$ is the start variable.
- Theorem: $REG \subseteq CFL$.
 - Given a DFA A , we can create a grammar G such that $L(G) = L(A)$. For every state q_i we create a variable V_i . If $\delta(q_i, a) = q_j$, we add a rule $V_i \rightarrow aV_j$, and if $q_i \in F$, we add a rule $V_i \rightarrow \varepsilon$. The start variable is V_0 , representing q_0 .
- **The class of CFL is closed under union, concatenation, and star.**
- If C is context-free and R is regular, $C \cap R$ is context-free.
- **Linear-right grammar:** All rules are of the form $A \rightarrow aB$ or $A \rightarrow \varepsilon$.
- **Leftmost derivation:** A derivation in which on every step we replace the leftmost variable.
- **Ambiguity:** An ambiguous grammar is defined as a grammar that may generate two different derivation trees for a single word.
 - Alternative definition: A grammar is ambiguous if it contains a word that has two different leftmost derivations.
- **Chomski normal form:**
 - The start variable S is never on the right side of any rule.

- All rules are of the form:
 - * $S \rightarrow \varepsilon$
 - * $A \rightarrow a$
 - (For $A \in V$, $a \in \Sigma$, and remember that $\varepsilon \notin \Sigma$ and so $a \neq \varepsilon$).
 - * $A \rightarrow BC$
 - (For $A, B, C \in V$, such that $B, C \neq S$).
- Theorem: For any CFG G , there exists a CFG G' in Chomski normal form, such that $L(G) = L(G')$. This is done in polynomial time (Tirgul, 24.3).
 - Membership check: Given a CFG G (in Chomski form) and a word w , we can answer the question of $w \stackrel{?}{\in} L(G)$ in polynomial time - using something like dynamic programming.
 - Emptiness check: Given a CFG G (in Chomski form), we can check whether or not $L(G) = \emptyset$ using a backwards-chaining algorithm, to see if the start variable can lead to a terminal. This is polynomial.
 - * (Both are detailed in the Tirgul, 24.3).
- A pushdown automaton is a tuple $\langle Q, \Sigma, \Gamma, \delta, Q_0, F \rangle$. The function: $\delta : Q \times (\Sigma \cup \{\varepsilon\}) \times (\Gamma \cup \{\varepsilon\}) \rightarrow 2^{Q \times (\Gamma \cup \{\varepsilon\})}$.
 - A PDA's configuration: State + stack contents. $c \in Q \times \Gamma^*$. $c = \langle q, s \rangle$.
 - $c' = \langle q', s' \rangle$ is σ -following from $c = \langle q, s \rangle$, if the PDA A in configuration c reads the letter σ and may go to configuration c' .
- **Pumping lemma** for context-free languages:
 - Let L be a context-free language. There exists $p \geq 1$ such that for every word $w \in L$, if $|w| \geq p$, there exists a partition $w = uvxyz$ such that:
 - * $|vy| > 0$
 - * $|vxy| \leq p$
 - * $\forall i \geq 0, uv^i xy^i z \in L$
 - Generally useful claim from its proof: If the branching factor in a tree is bounded by b , and the tree has n leaves, then the tree's height is **at least** $\log_b n$.
- **Equivalence of CFL and PDA:**
 - Given $L \in CFL$, we can create a PDA A such that $L(A) = L$. It will "guess" a leftmost derivation. Proof in lecture, 24.3.
 - Given a PDA A , we can construct an equivalent grammar. Proof in lecture, 24.3, but it was not very clear to me - the same proof can be found in Sipser.

2 Computability

- A Turing machine is a tuple $\langle Q, \Sigma, \Gamma, \delta, q_0, q_{acc}, q_{rej} \rangle$ where Q is the set of states, Σ is the input alphabet ($_ \notin \Sigma$), Γ is the working alphabet ($\Sigma \subseteq \Gamma$, and $_ \in \Gamma$), $\delta : Q \times \Gamma \rightarrow Q \times \Gamma \times \{L, R\}$, and $q_0, q_{acc}, q_{rej} \in Q$ are the start, accept, reject states .
 - A configuration of a TM:
 - * The state, contents of the tape, and the location of the reading head.
 - * \Rightarrow Therefore, if the machine's tape is bounded by some number n , we have no more than $|Q| \cdot |\Gamma|^n \cdot n$ configurations - select the state, contents, and head location, respectively.
 - * Each configuration can be represented as uqv , for a state $q \in Q$ and $u, v \in \Sigma^*$. This representation means the tape contains $u \cdot v$, and the machine head points at the first character in v .
- Basic decideability classes:

- A language L is **recursively enumerable (RE)** if there exists a TM M such that $L(M) = L$.
- A language L is **recursive (R)** if there exists a TM M that recognizes L , and in addition stops on all inputs.
- $L \in CO - RE \iff \bar{L} \in RE$
- Theorem: $R = RE \cap CO - RE$.
- An **enumerator**: A TM with a “printer”, that prints out a list of words. Might never halt, and might repeat words more than once.
 - $L \in RE \iff$ There exists an enumerator E such that $L(E) = L$.
- Both R and RE are **closed under union, intersection, concatenation, and star operations**.
- A **TM with k tapes** is equivalent in power to a regular TM. The simulation of k tapes on a single tape is done by writing the contents of different tapes on a single tape, separated by a special character. We mark on each tape the location of its “read head”. To simulate a single step, we go from left to right, realizing in what state the machine is, then going back over the tape, making the needed changes to each simulated tape, and pushing the other tapes forward in case a tape in the middle got too long to fit in its allocated memory.
 - If the runtime of the original k tape machine was $f(n)$, we know that each tape can’t exceed $f(n)$ space usage, and therefore each simulated step will cost us at most $\mathcal{O}(f^2(n))$ calculation steps. Therefore, the simulation from k tapes to 1 tape is polynomial.
- A **non-deterministic TM** M is defined in a way such that $\delta : Q \times \Gamma \rightarrow 2^{Q \times \Gamma \times \{L, R\}}$. A word w is in $L(M)$ if *there exists* an accepting run of M on w .
 - In terms of computing power, non-deterministic TMs are equivalent to deterministic TMs. We can prove this by simulating a non-deterministic TM’s “run tree” with a deterministic TM. We use a BFS procedure to explore the tree, and each node represents a run of the non-deterministic TM on the word. The deterministic TM will accept its input, if it finds an accepting run in this tree.
 - Cost of the simulation: If the original accepting run was of length T , then the run-tree is of height T , and therefore has b^T leaves (where b is the branching factor, the maximum number of non-deterministic choices possible in any state). Each leaf requires at most T steps, and therefore the simulation costs $\mathcal{O}(T \cdot b^T)$ - exponential time!!
- A RAM machine is a machine that has a memory tape, an accumulator tape, input tape, and PC-tape. It works much like a CPU and accepts commands similar to Assembly language. It can do anything a regular real-life computer can do. We’ve seen that it can be simulated by a Turing machine. (Tirgul, 21.4).
- Note that since they may be represented by finite strings, the set of all Turing machines is countable (via lexicographic ordering).
- **Mapping reduction**: We say that there is a mapping reduction from L_1 to L_2 , written: $L_1 \leq_m L_2$, if there exists a computable function $f : \Sigma^* \rightarrow \Sigma^*$ such that $x \in L_1 \iff f(x) \in L_2$.
 - A computable function is a function $f : \Sigma^* \rightarrow \Sigma^*$ such that there exists a TM that halts for all inputs x with $f(x)$ written on its tape.
- Useful theorems: Assume $L_1 \leq_m L_2$.
 - If $L_2 \in R$ then $L_1 \in R$.
 - If $L_1 \notin R$ then $L_2 \notin R$.
 - Same as above, for $RE, CO - RE$.
- Claim: $A \leq_m B$ iff $\bar{A} \leq_m \bar{B}$. Proof directly from the definitions.
- Claim: For every two languages $L_1, L_2 \in R$ such that $L_1, L_2 \neq \emptyset, \Sigma^*$, it holds that $L_1 \leq_m L_2$.

- Proof: Since $L_2 \neq \Sigma^*, \emptyset$, there exists $w_{not} \notin L_2$ and $w_{yes} \in L_2$. Since $L_1 \in R$ we can decide whether or not an input x is in L_1 it using a TM, and return either w_{not} or w_{yes} according to the TM's answer.

- **Rice's theorem:**

- A property is a set P of TMs. A *semantic* property is a property, such that for all M_1, M_2 such that $L(M_1) = L(M_2)$, it holds that either $M_1, M_2 \in P$, or $M_1, M_2 \notin P$.
- A property is *non-trivial* if there exist $M \in P$ and $M' \notin P$.
- Rice's theorem: Let P be a semantic, non-trivial property, and $L_P = \{\langle M \rangle \mid M \in P\}$. Then, $L_P \notin R$.
- **Important, stronger lemma** (under the previous assumptions): Let M_\emptyset be a machine such that $L(M_\emptyset) = \emptyset$. If $M_\emptyset \in P$, then $L_P \notin RE$.
 - * Proof, in Turgul, 28.4.

- **The Recursion Theorem:**

- Let T be a TM that computes some function $t : \Sigma^* \times \Sigma^* \rightarrow \Sigma^*$. There is a TM R that computes a function $r : \Sigma^* \rightarrow \Sigma^*$ such that for all w it holds that $r(w) = t(\langle R \rangle, w)$.
- This basically means that any TM can use the command "obtain an encoding of yourself"!

3 Complexity

Note that in this part we only discuss languages which are in R , so all TMs are deciders.

3.0.1 Time complexity

- Non-deterministic TMs: The runtime of a **decider** non-deterministic TM M is defined as the longest path in the tree which depicts all possible runs of M . Since it is a decider, all branches are finite. We look at the longest branch, even if it ends in rejection.

- If L is decidable by a non-deterministic TM which runs in time $t(n)$, L is decidable by a deterministic TM that runs in $2^{\mathcal{O}(t(n))}$.

- Time complexity classes:

- $TIME(t(n)) = \{L \mid L \text{ is decidable by a single tape deterministic TM on an input of length } n, \text{ in time } \mathcal{O}(t(n))\}$.

- $NTIME(t(n)) = \{L \mid L \text{ is decidable by a single tape non-deterministic TM on an input of length } n, \text{ in time } \mathcal{O}(t(n))\}$.

-

$$P = \bigcup_{k \geq 0} TIME(n^k)$$

-

$$NP = \bigcup_{k \geq 0} NTIME(n^k)$$

- Alternate definition for NP :

- * A **verifier** for a language L is a deterministic TM V such that $L = \{w \mid V \text{ accepts the pair } \langle w, c \rangle \text{ for some } c\}$. This c is the "certificate". A verifier's complexity is measured in relation to the length of w . A polynomial verified runs in time polynomial in $|w|$ (and so c also must be polynomial in $|w|$).
- * $L \in NP \Leftrightarrow L$ has a polynomial verifier.

- L is NP-complete if $L \in NP$ and L is NP-Hard: For all $L' \in NP$, $L' \leq_p L$.

- * L is NP-complete \Leftrightarrow if $L \in P$ then $P = NP$.

- Open questions!

- $P \stackrel{?}{=} NP$. Clearly $P \subseteq NP$, but the other direction is yet unknown.

- $NP \cap CO = NP \stackrel{?}{=} P$.
- $NP \stackrel{?}{=} CO = NP$.

- **Theorem:** If $L \in TIME(n)$, then L is regular. (We didn't prove this...?)

- Claim: For every two languages $L_1, L_2 \in P$ such that $L_1, L_2 \neq \emptyset, \Sigma^*$, it holds that $L_1 \leq_p L_2$.

- Proof: Since $L_2 \neq \Sigma^*, \emptyset$, there exists $w_{not} \notin L_2$ and $w_{yes} \in L_2$. Since $L_1 \in P$ we can decide polynomially whether or not an input x is in L_1 it using a TM, and return either w_{not} or w_{yes} according to the TM's answer.

3.0.2 Space complexity

- Space complexity classes:

- $SPACE(s(n)) = \{L \mid L \text{ is decidable by a deterministic TM with space complexity of } \mathcal{O}(s(n))\}$
- $NSPACE(s(n)) = \{L \mid L \text{ is decidable by a non-deterministic TM with space complexity of } \mathcal{O}(s(n))\}$

$$PSPACE = \bigcup_k SPACE(n^k)$$

$$NPSPACE = \bigcup_k NSPACE(n^k)$$

- It is worth noting that:

- $TIME(f(n)) \subseteq SPACE(f(n))$
 - * Naturally, if a machine runs for $f(n)$ time, it can never write more than one character per 'cycle'.
- $SPACE(f(n)) \subseteq TIME(2^{\mathcal{O}(f(n))})$
 - * The total number of possible configurations for a TM with memory bounded by $f(n)$ is $|Q| \cdot |\Gamma|^{f(n)}$. $f(n) = 2^{\mathcal{O}(f(n))}$, and of course runtime \leq number of configurations, otherwise we'd hit a 'loop'.

- **$PSPACE$ is closed under union, complementation, and star operations.**

- **Savitch's theorem:**

- For any function $s : \mathbb{N} \rightarrow \mathbb{N}$ such that $s(n) \geq n$, $NSPACE(s(n)) \subseteq SPACE(s^2(n))$
 - * **Important:** As a direct result, we obtain that $PSPACE = NPSPACE$.
 - Since any complexity class identified by a *deterministic* TM is closed under negation, we get $(CO - NPSPACE) = CO - PSPACE = PSPACE = NPSPACE$.

- ALL_{NFA} :

- An automaton A is **empty** if $L(A) = \emptyset$. It is **universal** if $L(A) = \Sigma^*$.
- $\{\langle A \rangle \mid A \text{ is an NFA that is not empty}\} \in PTIME$ (by running a BFS, etc.).
- $\overline{ALL_{NFA}} = \{\langle A \rangle \mid A \text{ is an NFA that is not universal} (\Leftrightarrow \exists w \notin L(A))\} \in PSPACE$
 - * Proven in lecture, 31.5. The idea: guess a word of length $\leq 2^n$ where A has n states. If you find a set S of states such that $S \cap F = \emptyset$, accept. Otherwise reject. Memory used is indeed polynomial because a counter up to 2^n takes n space, and the set of states maintained in memory is linear in size.
- From Savitch: $ALL_{NFA} \in PSPACE$.
- From the Tiring (9.6): $ALL_{NFA} \in PSPACE - Complete$

- A function $f : \mathbb{N} \rightarrow \mathbb{N}$ (assume $f(n) \geq \log(n)$) is **space constructible** if there exists a TM M that given an unary input 1^n , computes $f(n)$ and writes down the result in binary, under a memory limit of $f(n)$.

- **The memory hierarchy theorem:**
 - For any space constructible function $f(n)$, there exists a language L_f that is decidable under a memory limit of $\mathcal{O}(f(n))$, but **not** decidable under a memory limit of $o(f(n))$.
 - Conclusions:
 - * For all $k_1 > k_2$, it holds that $SPACE(\mathcal{O}(n^{k_2})) \subsetneq SPACE(n^{k_1})$.
 - * For all $k \in \mathbb{N}$, $SPACE(n^k) \subsetneq SPACE(2^n)$
 - * $PSPACE = \bigcup_{k \in \mathbb{N}} SPACE(n^k) \subseteq SPACE(2^n) \subsetneq SPACE(2^{2^n}) \subseteq EXPSPACE = \bigcup_{k \in \mathbb{N}} SPACE(2^k)$
· So, $PSPACE \subsetneq EXPSPACE$
 - Proof in Tirlgul, 2.6
- A function $f : \mathbb{N} \rightarrow \mathbb{N}$ is **time constructible** if $f(n) \geq n \log n$ and there exists a TM that given an unary input 1^n , computes $f(n)$ in binary representation, in time $\mathcal{O}(f(n))$.

- **The time hierarchy theorem:**
 - For any time constructible function $f(n)$, there exists a language L_f that is decidable in time $\mathcal{O}(f(n))$, but **not** decidable in time $o\left(\frac{f(n)}{\log(f(n))}\right)$.
 - Conclusion: $P \subsetneq EXPTIME$
 - This is true because $TIME(n^k) \subsetneq TIME(2^n)$, and so $P = \bigcup_k TIME(n^k) \subseteq TIME(2^n) \subsetneq TIME(2^{2^n}) \subseteq EXPTIME$.
 - **Open problem** (yay!!): We don't know which of the following is a “ \subset ” rather than a “ \subseteq ”:

$$P \subseteq NP \subseteq PSPACE \subseteq EXP$$

- A language L is PSPACE-Complete if:
 - $L \in PSPACE$
 - $L \in PSPACE$ – *Hard*: For any $L' \in PSPACE$, $L' \leq_p L$.
- Theorem: If $L \in PSPACE$ – *Complete*, and $L \in NP$, then $NP = PSPACE$.
 - Proof: For any $L' \in PSPACE$, polynomially calculate the reduction to L , and run L 's TM on the result.
 - Similarly proven: If $L \in P$, then $P = PSPACE$.

3.0.3 The classes LOGSPACE, NLOGSPACE

Note that for time complexity these classes are not interesting! We can't even read the entire input.

- $L = SPACE(\log(n))$
- $NL = NSPACE(\log(n))$
- We have an input tape: read only, and a working tape: read/write.
- Open question: $NL \stackrel{?}{=} L$.
 - Savitch's theorem gives us $NL = NSPACE(\log(n)) \subseteq SPACE(\log^2(n)) \neq L$, so it's not helpful here.
- A **log-space transducer** is a deterministic TM M with three tapes: Input (read only), work (read/write), output (write only). The working tape **contains only** $\mathcal{O}(\log n)$ cells (for an input of length n).
 - Definition: $f : \Sigma^* \rightarrow \Sigma^*$ is computable in logarithmic space, if there exists a log-space transducer that on input w outputs $f(w)$.
- NL completeness: A language A is NL – *Complete* if:

- $A \in NL$
- $A \in NL - \text{Hard}$: For any $B \in NL$, $B \leq_L A$.
 - * $B \leq_L A$ (“ B is logspace reducible to A ”) if there exists a function f that is computable in logarithmic space such that for all $w \in \Sigma^*$, $w \in B \Leftrightarrow f(w) \in A$.
- Theorem: if $A \in L$ and $B \leq_L A$, then $B \in L$.
 - Proof: Lecture, 9.6. Note that we couldn’t do it in the naive way so we had to “pipeline” the calculation of the transducer.
 - Conclusion: If A is $NL - \text{Complete}$ and $A \in L$ then $NL = L$.
- Theorem: $NL \subseteq P$ (It’s easy to see that also $NL \subseteq NP$, but this is trivial).
 - Proof: Since $PATH = \{\langle G, s, t \rangle \mid G \text{ is a directed graph, and there’s a path from } s \text{ to } t\} \in NL - \text{Complete}$, we can do the following: Given $B \in NL$, we reduce it to $PATH$ (in log-space and therefore in $PTIME$), and solve the reachability problem in $PTIME$, since $PATH \in P$.
- To conclude:

$$L \subseteq NL \subseteq P \subseteq NP \subseteq PSPACE \subseteq EXPTIME$$
 - It is known that $L \subsetneq PSPACE$.
 - It is known that $P \subsetneq EXPTIME$ (From the time hierarchy theorem).
 - It is known that $PSPACE \subsetneq EXPSPACE$ (From the space hierarchy theorem).

Part II

Counter examples, etc.

Reduction strategies

Some useful techniques we've encountered to create mapping reductions:

- Given $\langle M, w \rangle$ on an input x (Or sometimes just $\langle M \rangle$ and input x):
 - Run M on w and act the same as M .
 - Run M on w for $|x|$ steps, and act the same (or the opposite) as M .
 - Run through a “filter”, for example accept all words that start with 0 or are of the form $0^n 1^n$. The filter can either accept words immediately or reject them immediately. If x passed the filter, run M on w and accept/reject as needed.
 - Change something about M (for example change the reject state to an infinite loop in the reduction $A_{TM} \leq HALT_{TM}$) and return the modified M .
 - Return a machine that writes a constant word on its tape and then runs M on it.
 - Useful claim: for two sets A, B , it holds that $A \subseteq B \Leftrightarrow A \cap \overline{B} = \emptyset$.
 - Any more? TODO

Languages and problems

- L is regular iff L^R is regular.
 - Proof in ex2. Reverse all arrows in the DFA graph and switch accept/start states.
- $\{0^n 1^n \mid n \geq 0\} \notin REG$
 - Proof: Pump $0^p 1^p$.
 - It is in CFL. A CFG for it: $S \rightarrow 0S1 \mid \varepsilon$
 - This language is in class L . Count the number of 0s and 1s in binary and compare. The counting takes up $O(\log n)$ memory.
- $\{w \mid w = w^R\} \notin REG$
 - Proof: Take $w = 0^p 10^p$, and use $i = 2$ with the pumping lemma.
- $\{0^m 1^n \mid m > n\} \notin REG$
 - Proof: Take $w = 0^{p+1} 1^p$, and use $i = 0$ with the pumping lemma.
- $\{w \in \{0, 1\}^* \mid \text{The number of 0's in } w \text{ is greater than the number of 1's in } w\} \notin REG$
 - Since $0^* 1^*$ is regular, and the intersection of it and this language is $\{0^m 1^n \mid m > n\}$, we get that this language can't be regular.
- $\{1^a + 1^b = 1^{a+b}\} \notin REG$
 - Proof: Take $w = 1^p + 1^p = 1^{2p}$, and use $i = 2$ with the pumping lemma.
- Grammar G for all legal parentheses, where $'(= a$, and $)' = b$:
 - $S \rightarrow aSb \mid SS \mid \varepsilon$
 - Proof that it is not regular: $L(G) \cap a^* b^* = \{a^n b^n \mid n \geq 0\} \notin REG$
- Grammar for the language of palindromes over $\Sigma = \{a, b\}$:

- $S \rightarrow aSa | bSb | \varepsilon | a | b$
- $\{w \cdot w^R \mid w \in (0+1)^*\} \in CFL$. Easy to build a PDA for it.
- A language **not in CFL**: $L = \{a^n b^n c^n \mid n \geq 0\} \notin CFL$.
 - Note: $L = \{a^n b^n c^k \mid n, k \geq 0\} \cap \{a^k b^n c^n \mid n, k \geq 0\}$, and both of those languages are CFL. Thus, once we prove that $L \notin CFL$, we have proof that CFL is **not closed under intersection**.
 - Proof that it's not in CFL: $a^p b^p c^p$, pump with $i = 2$. (In any partition $uvxyz$ good for the pumping lemma, vxy must contain at most 2 different characters).
- Another language not in CFL: $\{w \cdot w \mid w \in (0+1)^*\} \notin CFL$
 - Proof: $0^p 1^p 0^p 1^p$, and pump.
 - Bad proof: $0^p 10^p 1$, won't work! We can divide to $uvxyz = (0^{p-1})(0)(1)(0^{p-1})(01)$.
- An **ambiguous grammar**: $E \rightarrow E + E \mid E \times E \mid 0 \mid 1 \mid \dots \mid 9$. The word “ $3 + 5 \times 8$ ” can be parsed either as $(3 + 5) \times 8$ or $3 + (5 \times 8)$.
 - One way to make it non-ambiguous but still equivalent: $E \rightarrow T + E \mid T \times E \mid T$, and also $T \rightarrow 0 \mid 1 \mid \dots \mid 9$. It won't obey standard algebra priority rules, but will nevertheless be non-ambiguous.
- $A_{TM} = \{\langle M, w \rangle \mid w \in L(M)\} \in RE \setminus R \quad (\Rightarrow \overline{A_{TM}} \in CO - RE \setminus R)$.
 - $A_{TM} \in NP - Hard$. Proof: If $L \in NP$, there is a TM M that decides it. For the poly mapping, we take $f(x) = \langle M, x \rangle$. It holds that $x \in L \Leftrightarrow f(x) \in A_{TM}$.
 - $A_{TM} \notin NP$ (it's not even in R !) and therefore $A_{TM} \notin NP - Complete$.
- $HALT_{TM} = \{\langle \langle M, \cdot \rangle, w \rangle \mid M \text{ halts in its run on } w\} \in RE \setminus R$
 - Proof that $HALT_{TM} \notin R$: Assume by contradiction that it is, then we could build a decider for A_{TM} that first checks for halting, and then acts accordingly. This is of course a contradiction.
 - $HALT_{TM}^{\varepsilon} = \{\langle M \rangle \mid M \text{ halts on the empty input}\} \in RE \setminus R$. By reduction $HALT_{TM} \leq_m HALT_{TM}^{\varepsilon}$: Given a machine M and a word w , map them to a machine M' that writes w on its tape and runs M on it.
- $INF_{TM} = \{\langle M \rangle \mid L(M) \text{ is infinite}\} \notin RE \cup CO - RE$. By reductions $A_{TM} \leq_m INF_{TM}$ and $HALT_{TM} \leq_m INF_{TM}$.
- **The tiling problem**: $TILE = \{\langle T, H, V, t_{init} \rangle \mid \text{There exists a legal tiling } n \times n \text{ for all } n \in \mathbb{N}\} \in CO - RE \setminus RE$.
 - This is equivalent to asking whether or not there exists a legal tiling for the infinite quarter of the plain. **Konig's lemma** says that in an infinite tree, if the branching factor in each node is finite, then there exists an infinite path in the tree. This lemma shows that the equivalence is indeed correct, by looking at a tree of legal tilings, in which a tile is a descendant of another tile if it expands on it.
 - In $CO - RE$ by simple construction.
 - Not in RE by reduction $\overline{HALT_{TM}^{\varepsilon}} \leq_m TILE$. Given a TM $\langle M \rangle$, we build an instance of the tiling problem that has a legal tiling for all $n \times n$ in the quarter of the plain, iff M does not halt on ε .
- $REG_{TM} = \{\langle M \rangle \mid L(M) \text{ is regular}\} \notin RE \cup CO - RE$.
 - Reductions: $A_{TM} \leq_m REG_{TM}$ and $\overline{A_{TM}} \leq_m REG_{TM}$. (Using a “filter” of checking whether or not $x = 0^n 1^n$ - one direction needs the filter before running M on w , the other direction needs the filter after M accepts w).
 - Similarly (from ex8): $L = \{\langle M \rangle \mid L(M) \text{ is context-free}\} \notin RE \cup CO - RE$, using reductions from A_{TM} and $\overline{A_{TM}}$ and filtering with $a^n b^n c^n$.

- $MAJ_k = \{\langle M \rangle \mid M \text{ is a TM over } \Sigma = \{0, 1\} \text{ that accepts at least half of all words of length } k\} \in RE \setminus R$
 - Proof: Run iteratively on all words of length k for $i = 0, 1, 2, \dots$ iterations, and find out. The language is not in R by Rice's theorem.
- $HALF_k = \{\langle M \rangle \mid M \text{ is a TM over } \Sigma = \{0, 1\} \text{ that accepts exactly half of all words of length } k\} \notin RE \cup CO - RE$
 - Proof by reductions $A_{TM} \leq_m HALF_k$ and $\overline{A_{TM}} \leq HALF_k$. The idea: Reject all words of length different than k , and then check if the word starts with a 0, and afterwards check M 's answer on w . Accept/reject in the last two stages according to the direction of the reduction we prove...
- $MIN_{TM} = \{\langle M \rangle \mid M \text{ is a 'minimal' machine for the language } L(M)\} \notin RE$
 - (Minimality is defined by the number of states, and comparison between two machines with the same number of states is lexicographic).
 - Proof by contradiction, using an enumerator for $L(MIN_{TM})$. Full proof in Turgul, 11.5.
- $E_{TM} = \{\langle M \rangle \mid M \text{ is a TM and } L(M) = \emptyset\} \in CO - RE \setminus R$
 - Go over all $i=1,2,3,\dots$ words in lexicographic ordering and run for i steps, to see if any are accepted, to identify $\overline{E_{TM}}$.
- $ALL_{TM} = \{\langle M \rangle \mid M \text{ is a TM and } L(M) = \Sigma^*\} \notin RE \cup CO - RE$
- $EQ_{TM} = \{\langle M_1, M_2 \rangle \mid M_1, M_2 \text{ are TMs and } L(M_1) = L(M_2)\} \notin RE \cup CO - RE$
- $ONE_{TM} = \{\langle M \rangle \mid M \text{ is a TM and } |L(M)| = 1\} \notin RE \cup CO - RE$
- $SAMESIZE_{TM} = \{\langle M_1, M_2 \rangle \mid M_1, M_2 \text{ are TMs and } |L(M_1)| = |L(M_2)|\} \notin RE \cup CO - RE$
 - Reduction $ONE_{TM} \leq_m SAMESIZE_{TM}$, given a TM M we return $\langle M, M^1 \rangle$ for M^1 such that $|L(M^1)| = 1$.
- $\{\langle M_1, M_2 \rangle \mid L(M_1) \subseteq L(M_2)\} \notin RE \cup CO - RE$
 - By reduction from ALL_{TM} . Return $\langle M^*, M \rangle$ for some $M^* \in ALL_{TM}$.
- $U = \{\langle M, w, 1^n \rangle \mid M \text{ is a nondeterministic TM that accepts } w \text{ within } n \text{ steps}\} \in NP - Complete$
 - In NP because we can run M on w for n steps and see if it accepts.
 - In $NP - Hard$: Let $L' \in NP$ and let M be a TM for L' , and $p(n)$ its polynomial. Given an input w for L' , the function $f : \Sigma^* \rightarrow \Sigma^*$ returns $\langle M, w, 1^{p(|w|)} \rangle$. Indeed, $\langle M, w, 1^{p(|w|)} \rangle \in U \Leftrightarrow w \in L'$, and f is polynomial.
 - We've seen a reduction: $U \leq_p BT$ (Bounded Tiling), which means BT is also $NP - Complete$. (Lecture, 17.5)
- $3SAT \in NP - Complete$. Shown in lecture, by $BT \leq_p SAT \leq_p 3SAT$. (Lecture, 24.5).
 - Also: $SAT \in PSPACE$
- $CLIQUE = \{\langle G, k \rangle \mid G \text{ is an undirected graph with a } k\text{-clique}\} \in NP - Complete$
 - Reduction: $3SAT \leq_p CLIQUE$.
- $IS = \{\langle G, k \rangle \mid G = \langle V, E \rangle \text{ is an undirected graph and it has an independent set of size } k\} \in NP - Complete$
 - $S \subseteq V$ is an independent set if for all $u, v \in S$ there's no edge between u and v .
 - $CLIQUE \leq_p IS$ by taking the graph G and replacing its edges with $E' = (V \times V) \setminus E$.

- $VC = \{\langle G, k \rangle \mid G \text{ is an undirected graph, and } G \text{ has a vertex cover of size } k\} \in NP - Complete$
 - As seen on ex11.
 - Reduction 1 (from Sipser's book): $3SAT \leq_p VC$. Given an input φ , create a graph: Add two vertices for each variable and add three vertices for each clause, in which every literal is represented by a vertex. Then add edges in between each clause, and from each literal to its appropriate node from the first stage. Return this graph, with $k = 2r + m$, where r is the number of clauses, and m is the number of variables, in the original input φ .
 - Reduction 2: $IS \leq_p VC$. For $\langle G, k \rangle$ return $\langle G, |V| - k \rangle$.
- $SUBSETSUM = \{\langle y_1, y_2, \dots, y_l, t \rangle \mid \exists I \subseteq [l] : \sum_{i \in I} y_i = t\} \in NP - Complete$
 - Proof by reduction $3SAT \leq_p SUBSETSUM$, using a big table of 1's and 0's. (Tirgul, 26.5).
- $HAMPATH = \{\langle G, s, t \rangle \mid \text{There is a directed Hamiltonian path from } s \text{ to } t \text{ in } G\} \in NP - Complete$
 - Proof by reduction $3SAT \leq_p HAMPATH$, by building a graph with diamond-like shapes in it... (Tirgul, 26.5).
- $KNAPSACK = \{\langle (v_1, w_1), (v_2, w_2), \dots, (v_n, w_n), V, W \rangle \mid \exists I \subseteq [n] : \sum_{i \in I} v_i \geq V \wedge \sum_{i \in I} w_i \leq W\} \in NP - Complete$
 - Via $SUBSETSUM \leq_p KNAPSACK$, in Tirgul, 2.6, and also similar to exercise 10's question with the meals.
- $TSP = \{\langle G, w, t \rangle \mid G \text{ is a complete graph, and in } G \text{ there is a directed circle path } P \text{ that goes through each vertex once such that } \sum_{e \in P} w(e) \leq t\} \in NP - Complete$
 - Reduction: $HAMCYCLE \leq_p TSP$. For a graph G the reduction returns $\langle G', w, t \rangle$ such that G' is a complete graph, and w evaluates as 1 for edges that were in G and $n + 1$ for edges that were not in G , and $t = |E|$.
- $BT = \{\langle T, H, B, t_{init}, t_{fin}, 1^n \rangle \mid \text{There's a legal } n \times n \text{ tiling}\} \in NP - Complete$
- $SBT = \{\langle T, H, V, t_{init}, t_{fin}, 1^n \rangle \mid \text{There's a legal } n \times m \text{ tiling for } 1 \leq m \leq |T|^n\} \in PSPACE - Complete$.
- $TQBF = \{\langle \varphi \rangle \mid \varphi \text{ is a true sentence}\} \in PSPACE - Complete$, by reduction $SBT \leq_p TQBF$.
- $ALL_{NFA} = \{\langle A \rangle \mid A \text{ is an NFA and } L(A) = \Sigma^*\} \in PSPACE - Complete$.
- $PATH = \{\langle G, s, t \rangle \mid G \text{ is a directed graph, and there's a path from } s \text{ to } t\} \in NL - Complete$.
 - Proof of being in NL : A nondeterministic TM will guess a path - it has to be of length $\leq |V|$, and at any moment we hold only info about the number of steps ($\log|V|$ cells) and the current node ($\log|V|$ cells).
 - It's also $NL - Hard$. Proof in lecture, 9.6.
 - Note that $PATH \in P$ by using a BFS search.
 - Open problem: $PATH \stackrel{?}{\in} L$.