CS Theory (Spring '25)

Test 2 Review Solutions

Instructor: *Toniann Pitassi and William Pires*

1 Concepts to review

Be comfortable answering the following:

(a) What is P ?

(b) What is NP ?

(c) What is a decidable language ?

(d) What is a recognizable language ?

(e) What's the language of a TM ?

(f) What's the language of a NTM ?

2 True or False

(a) Every NP-hard language is NP-complete. False

(b) Every decidable language is in NP. False

(c) Every language in P is in NP. True

(d) There exists a finite language that is undecidable. False

(e) There exists an infinite language that is in P. True

(f) There is an algorithm for $3SAT$ that runs in exponential-time. True

(g) The following problem is decidable: on input $< M >$, accept if and only if M halts in polynomial-time on all inputs. False

(h) A polynomial-time mapping reduction from 3SAT to $L = \{0^n 1^n \mid n \geq 0\}$ implies $P = NP$. True

(i) Let $\overline{HALT}$ be the set of inputs $< M, w >$ such that M does not halt on w. $\overline{HALT}$ is infinite. True

(j) $\overline{HALT}$ is recognizable. False

(k) Let A be polynomial-time mapping reducible to B. Then it is possible that A is in P but B is not in NP. True

(l) The set of languages over $\Sigma = \{0, 1\}$ is countable. False

(m) The set of all strings $\langle M \rangle$ where M is a TM, is countable. True

3 Recognizable languages

Prove the following languages are recognizable. In each problem you can assume the Turing machines take as inputs strings over $\Sigma = \{0, 1\}$.

(a) $L = \{\langle M \rangle \mid M$ accepts at least one string w such that w has odd length and ends with a 1 $\}$

Solution.

Let $w_1, w_2, \ldots$ be the lexicographic enumeration of all strings over $\Sigma = \{0, 1\}$.

Algorithm 1 Recognizer for L

Input: $\langle M \rangle$ where M is a TM

```
1: for i = 1, 2, 3, ... do
2:     for 0 ≤ j ≤ i do
3:         Run M on w_j for i steps.
4:         if M accepted and w_j has odd length and w_j ends with a 1 then
5:             Accept ⟨M⟩
6:         end if
7:     end for
8: end for
```

For correctness, assume first that there is a string of odd length and ending with a 1 that M accepts. Let w be the first such string, and suppose that w is the ℓ^{th} string in our enumeration. And suppose that M accepts w in t steps. Let t' be the max of t and ℓ. Then our simulation will accept w when $i = t'$ and $j = \ell$, and therefore our algorithm accepts all strings in the language. Conversely if M does not accept any string w ending with 1 and of odd length, then our simulation will never halt and accept since our algorithm only halts and accepts when M halts and accepts on a string of this form.

(b) $L = \{< M >| \ M \text{ accepts some string ending with 11 or 01}\}$.

Solution.

Algorithm 2 Recognizer for L

Input: $\langle M \rangle$ where M is a TM

1: **for** $i = 0, 1, 2, 3, \ldots$ **do**
2: Run M on all strings w over $\{0, 1\}$ with $|w| \leq i$ for i steps.
3: Look at the strings M accepted in the above step.
4: **if** M accepted some string w and w ends with 11 or 01. **then**
5: Accept $\langle M \rangle$
6: **end if**
7: **end for**

For correctness, assume first that there is a string ending with 11 or 01 that M accepts. Let w be the lexicographically first such string, and suppose that $|w| = \ell$. And suppose that M accepts w in t steps. Let t' be the max of t and ℓ. Then our simulation will accept w in the step where $i = t'$, and therefore our algorithm accepts all strings in the language. Conversely if M does not accept any string w ending with 11 or 01, then our simulation will never halt and accept since our algorithm only halts and accepts when M halts and accepts on a string of this form.

(c) $L = \{\langle M_1, M_2\rangle \mid \text{such that there exists a string } w \text{ accepted by } M_1 \text{ and rejected by } M_2\}$

Solution.

Let $w_1, w_2, \ldots$ be the lexicographic enumeration of all strings over $\Sigma = \{0, 1\}$.

Algorithm 3 Recognizer for L

Input: $\langle M_1, M_2\rangle$ where M_1, M_2 are TMs

```
1: for i = 1, 2, 3, ... do
2:     for 0 ≤ j ≤ i do
3:         Run M_1 on w_j for i steps.
4:         Run M_2 on w_j for i steps.
5:         if M accepted and w_j and M_2 rejected w_j then
6:             Accept ⟨M_1, M_2⟩
7:         end if
8:     end for
9: end for
```

For correctness, suppose M_1, accepts some string that M_2 rejects, and let w be the lexicographically smallest string that M_1 accepts and M_2 rejects. Assume that w is the ℓ^{th} string in lexicographic order, and assume that M_1 accepts w in t_1 steps and M_2 rejects w in t_2 steps. Then when we run the loop for $i = max(\ell, t_1, t_2)$, when $j = \ell$ will simulate both M_1 and M_2 on w and M_1 will accept, and M_2 will reject w and therefore we will accept. On the other hand, if the strings accepted by M_1 and the strings rejected by M_2 are disjoint sets, then we will never accept since the only way our simulation accepts is if it finds some string w that was accepted by M_1 and rejected by M_2.

4 Decidable languages

For each of the following languages, state whether or not it is decidable and prove your answer. To prove a language is undecidable using a Turing reduction. Use A_{TM} or Halt or Empty for your reduction. To prove decidable give pseudocode to describe a decider. In each problem you can assume the Turing machines take as inputs strings over $\Sigma = \{0, 1\}$.

(a) $L = \{\langle M \rangle \mid M \text{ accepts } 00\}$.

Solution.

We show a Turing reduction $A_{TM} \leq_t L$, in class we've shown that A_{TM} is undecidable so this proves L is not decidable. Let N be a decider for L. We can construct a decider for A_{TM} as follows:

Algorithm 4 A decider D for A_{TM}

Input: $\langle M, w \rangle$ where M is a TM and w is a string

1: Consider the TM M' defined as follows:
"On input x:
1. If $x \neq 00$: reject.
2. Run M on w
3. If M accepted w, accept x.
4. If M rejected w, reject."
2: Run N on $\langle M' \rangle$. ▷ N always halts because it *decides* L
3: If N accepted, accept $\langle M, w \rangle$.
4: If N rejected, reject $\langle M, w \rangle$.

We now show D is a decider for A_{TM}. The only string x that M' can accept is 00. If M accepts w then M' accepts 00, and otherwise M' doesn't accept 00.

(1) If $\langle M, w \rangle \in A_{TM}$, then M accepts w, so M' accepts 00 and $\langle M' \rangle \in L$. So N accepts $\langle M' \rangle$ on line 2, and D accepts $\langle M, w \rangle$ on line 3.
(2) If $\langle M, w \rangle \notin A_{TM}$, then M doesn't accept w, and M' doesn't accept 00 so $\langle M' \rangle \notin L$. So N rejects $\langle M' \rangle$ on line 2, and D rejects $\langle M, w \rangle$ on line 4.

So D always halts and accepts $\langle M, w \rangle$ iff $\langle M, w \rangle \in A_{TM}$. So D decides A_{TM}, meaning $A_{TM} \leq_t L$.

(b) $L = \{\langle M \rangle \mid M \text{ accepts at least 3 strings } \}$

Solution.

We show a Turing reduction $A_{TM} \leq_t L$, in class we've shown that A_{TM} is undecidable so this proves L is not decidable. Let N be a decider for L. We can construct a decider for A_{TM} as follows:

Algorithm 5 A decider D for A_{TM}

Input: $\langle M, w \rangle$ where M is a TM and w is a string

1: Consider the TM M' defined as follows:
"On input x:
1. Run M on w
2. If M accepted w, accept x.
3. If M rejected w, reject."
2: Run N on $\langle M' \rangle$. ▷ N always halts because it *decides* L
3: If N accepted, accept $\langle M, w \rangle$.
4: If N rejected, reject $\langle M, w \rangle$.

We now show D is a decider for A_{TM}. First note that of if M accepts w then $L(M') = \Sigma^*$ so M' accepts more than 3 strings, and otherwise $L(M') = \emptyset$, so M' accepts strictly less than 3 strings.

(1) If $\langle M, w \rangle \in A_{TM}$, then M accepts w, so M' accepts more than 3 strings and $\langle M' \rangle \in L$. So N accepts $\langle M' \rangle$ on line 2, and D accepts $\langle M, w \rangle$ on line 3.
(2) If $\langle M, w \rangle \notin A_{TM}$, then M doesn't accept w, and M' accepts 0 strings so $\langle M' \rangle \notin L$. So N rejects $\langle M' \rangle$ on line 2, and D rejects $\langle M, w \rangle$ on line 4.

So D always halts and accepts $\langle M, w \rangle$ iff $\langle M, w \rangle \in A_{TM}$. So D decides A_{TM}, meaning $A_{TM} \leq_t L$.

(c) $L = \{< M_1, M_2 > |$ there exists some string w such that M_1 accepts w and M_2 rejects w. $\}$

Solution.

Recall that $E_{TM} = \{\langle M \rangle \mid L(M) = \emptyset\}$ is undecidable. We show a Turing reduction $E_{TM} \leq_t L$, in class we've shown that E_{TM} is undecidable so this proves L is not decidable. Let N be a decider for L. We can construct a decider for E_{TM} as follows:

Algorithm 6 A decider D for E_{TM}

Input: $\langle M \rangle$ where M is a TM

1: Consider the TM $M_\emptyset$ which rejects every string.
2: Run N on $\langle M, M_\emptyset \rangle$. ▷ N always halts because it *decides* L
3: If N accepted, reject $\langle M \rangle$.
4: If N rejected, accept $\langle M \rangle$.

We now show D is a decider for E_{TM}. First note that $M_\emptyset$ rejects every string. So if M accepts some string w then $\langle M, M_\emptyset \rangle \in L$. If M doesn't accept any string, then $\langle M, M_\emptyset \rangle \notin L$

(1) If $\langle M \rangle \in E_{TM}$, then M doesn't accept any string. So $\langle M, M_\emptyset \rangle \notin L$. So N rejects $\langle M, M_\emptyset \rangle$ on line 2, and D accepts $\langle M \rangle$ on line 4.
(2) If $\langle M \rangle \notin E_{TM}$, then M accepts some string w. So $\langle M, M_\emptyset \rangle \in L$. So N accepts $\langle M, M_\emptyset \rangle$ on line 2, and D rejects $\langle M, w \rangle$ on line 3.

So D always halts and accepts $\langle M \rangle$ iff $\langle M \rangle \in E_{TM}$. So D decides E_{TM}, meaning $E_{TM} \leq_t L$.

Good practice: Try to show $A_{TM} \leq_T L$?

5 NP Completeness

Prove the following languages are NP-complete. For proving the language is in NP, you can either give a verifier or an NTM. For proving the language is NP-hard, you must use one of the following NP-complete languages in your reduction: 3SAT, Graph-Color, Clique, Independent-Set, Hitting-Set, Subset-Sum. (Graph-Color is a generalization of 3Color: The input is (G, k) where G is an undirected graph. The input is accepted iff there is a proper k-coloring of G.)

(a) You are given a k tables and a list of n guests. For each guest you must place at one of the tables (some tables can be left empty).

You're also given a list $C = \{c_1, \ldots, c_\ell\}$ of constraints of the form (g_i, g_j) which says that guests i and j cannot sit at the same table.

Floor-Plan $= \{\langle k, n, C\rangle \mid$ we can place the n guests using k tables and respect the constraints in $C.\}$

Solution.

We first give a verifier to show the problem is in NP.

Algorithm 7 A polytime verifier for Floor-Plan

Input: $\langle k, n, C\rangle, \mathsf{cert}\rangle$, where C is a list of constraints

1: Check $\mathsf{cert} = (t_1, \ldots, t_n)$ where each t_i is an integer between 1 and k (in binary).
2: We interpret t_i has meaning guest i is assigned to table t_i.
3: **for** each constraint (g_i, g_j) in C **do**
4: If $t_i = t_j$ reject. ▷ This means guest i and j are assigned to the same table
5: **end for**
6: Accept. ▷ This means that all constraints are respected

Tip: In your verifiers, it should be clear what the certificate format is and what it means. For instance don't write "Check cert is a floor plan".
Similarly, if you give an NTM you should write something like *"Non-deterministically assign a table $1 \le t_i \le k$ to each guest i."*. Do not write "Non-deterministically pick a floor-plan".

Solution.

We now show Floor-Plan is NP-hard. We show Graph-Color $\le_P$ Floor-Plan.
We're given $\langle G, k\rangle$ where G is an undirected graph with vertices $\{v_1, \ldots, v_n\}$ and k an integer.
We output $f(\langle G, k\rangle) = \langle k, n, C_G\rangle$ where $C_G = \{(i, j) \mid (v_i, v_j)$ is an edge in $G\}$.

Here are some details so that you can see why this is correct.

First, it's easy to see f can be computed in polynomial time, you only need to write down n, k and C which is (almost) the same thing as the list of edges in G.

In the reduction, an edge (v_i, v_j) of G becomes a conflict (i, j) in C. So (v_i, v_j) can have the same color iff guests i and j can be at the same table.
So assume there is a k-coloring of G. Then pick such a coloring α, we can build a floor-plan as follows: if v_i is colored with color ℓ, we assign guest i to table ℓ. Since adjacent vertices in G have different colors in α, no conflicting guests are at the same table.
Similarly if $\langle n, k, C_G\rangle \in$ Floor-Plan. Pick a valid assignment α of guests to table. We can build a k coloring of G as follows: if guest i is at table ℓ we color v_i with ℓ in G. Since conflicting guests (i, j) are at different tables, in the coloring adjacent vertices (v_i, v_j) must have a different color.

(b) You are given a set S of n items each with a value v_i and a weight w_i. You're also given a target t and a weight limit W. (all numbers are positive integers given in binary). Does there exists $T \subseteq S$ such that $\sum_{i \in T} v_i = t$ and $\sum_{i \in T} w_i \leq W$?

$$\mathsf{Knapsack} = \{\langle S, t, W\rangle \mid \exists T \subseteq S \text{ such that } \sum_{i \in T} v_i = t \text{ and } \sum_{i \in T} w_i \leq W\}$$

Solution.

We first give an NTM to show the problem is in NP.

Algorithm 8 A polytime NTM for Knapsack

Input: $\langle S, t, w\rangle$

1: Non deterministically pick $T \subseteq S$.
2: If $\sum_{i \in T} v_i \neq t$: Reject.
3: If $\sum_{i \in T} w_i > W$: Reject.
4: Accept.

Solution.

We now show KnapSack is NP-hard. We show Subset-Sum $\leq_P$ Knapsack.
We're given $\langle S, t\rangle$ where $S = \{x_1, \ldots, x_n\}$ is a set of integers and t a positive integer. We output $f(\langle S, t\rangle) = \langle S', t, n\rangle$ where $S' := \{(x_1, 1), \ldots, (x_n, 1)\}$.

Here are some details so that you can see why this is correct.

First, it's easy to see f can be computed in polynomial time, you only need to write down S', t, n. We can build S' in $O(|S|)$ time given S.

In the reduction, we assign every item in S a weight of 1. We then ask if we can pick items such that their values sums to t and the total weight is $\leq n = |S|$. In particular, no matter how we pick items, the total weight is always $\leq n$.

So assume $\langle S, t\rangle \in$ Subset-Sum. Then there exists $T \subseteq S$ such that $\sum_{i \in T} s_i = t$. For $\langle S', t, n\rangle$ the same subset T works in showing $\langle S', t, n\rangle \in$ KnapSack. We have $\sum_{i \in T} v_i = t, \sum_{i \in T} w_i \leq n$.

If $\langle S', t, n\rangle \in$ Knapsack, then exists $T \subseteq S$ such that $\sum_{i \in T} s_i = t$. So in S we have $\sum_{i \in T} s_i = t$. So $\langle S, t\rangle \in$ Subset-Sum.

(c) Double-Clique $= \{\langle G, k\rangle \mid G$ is a graph that contains two disjoint cliques of size $k\}$ (the two cliques are not allowed to have any vertices in common).

Solution.

We first give a verifier to show the problem is in NP.

Algorithm 9 A polytime verifier for Double-Clique

Input: $\langle G, k, c\rangle$

1: Check $c = (S, S')$ where S, S' are sets of k of vertices from G.
2: If $S \cap S' \neq \emptyset$: Reject.
3: **for** each pair of vertices u, v in S **do**
4: If (u, v) isn't an edge in G: reject.
5: **end for**
6: **for** each pair of vertices u, v in S' **do**
7: If (u, v) isn't an edge in G: reject.
8: **end for**
9: Accept.

Solution.

We now show Double-Clique is NP-hard. We show Clique $\leq_P$ Double-Clique.
We're given $\langle G, k\rangle$ where G is an undirected graph and k a positive integer.

We output $f(\langle G, k\rangle) = \langle G', k\rangle$ where G' is a new graph which is made of two copies of G called G_1 and G_2. I.e. for every vertex v of G we now have two copies v_1, v_2. If (u, v) is an edge of G we now have edges (u_1, v_1) and (u_2, v_2).

Here are some details so that you can see why this is correct.

First, it's easy to see f can be computed in polynomial time, we just make two copies of G, so this takes $O(|G|)$ time.

First if G has a clique S of size k, then G' has at least two cliques of size: the copy of S in G_1 and the copy in G_2. So if $\langle G, k\rangle \in$ Clique then $\langle G', k\rangle \in$ Double-Clique.
Now if $\langle G', k\rangle \in$ Double-Clique then G' has a clique S of size k [a]. Then S is either a subset of the vertices of G_1 or of G_2 (there's no edges between the two copies of G we made). But if S is a subset of vertices in G_1, since G_1 is a copy of G: the vertices in S must form a clique in G. The same is true if S is a subset of vertices in G_2. So if $\langle G', k\rangle \in$ Double-Clique then $\langle G', k\rangle \in$ Clique.

[a] G' must have at least two, but we only need to care about one

(d) VertexCover = $\{\langle G, k\rangle \mid \exists$ a subset S of k vertices such that every edge has at least one endpoint in $S\}$. (I.e. for every edge (u, v) in the graph u or v (or both) is in S)

Solution.

We first give an NTM to show the problem is in NP.

Algorithm 10 A polytime NTM for VertexCover

Input: $\langle G, k, c\rangle$

1: Non-deterministically pick a subset S of k vertices of G.
2: **for** each vertex (u, v) of G **do**
3: If $u \notin S$ and $v \notin S$: Reject.
4: **end for**
5: Accept.

Solution.

We now show VertexCover is NP-hard. We show IndependentSet $\leq_P$ VertexCover.
We're given $\langle G, k\rangle$ where G is an undirected graph with n vertices and k a positive integer.We output $f(\langle G, k\rangle) = \langle G, n - k\rangle$.

Here are some details so that you can see why this is correct.

First, it's easy to see f can be computed in polynomial time.

For the correctness. First, let V be the set of vertices of G.
Suppose that G has an independent set S of size k. Then, for every edge (u, v) of G at most one of u or v is in S. So for every edge (u, v) at least of the endpoints is in $V \setminus S$. So $V \setminus S$ is a vertex cover of size $n - k$.
Conversely, suppose that G has an independent set S' of size $n - k$. Then for every edge (u, v) at least one of u or v is in S'. So for every edge (u, v) at most of the endpoints is in $V \setminus S'$. So $V \setminus S'$ is an independent set of size k.
Hence we can see that $\langle G, k\rangle \in$ IndependentSet $\iff f(\langle G, k\rangle) = \langle G, n-k\rangle \in$ VertexCover.

(e) 3SAT-under-$\rho = \{\langle \phi, j \rangle \mid \phi$ is a 3-CNF formula and it has a satisfying assignment α which sets $x_j = 1\}$. Here ϕ has n variables $x_1, \ldots, x_n$ and you can assume $\langle \phi, j \rangle$ always has $1 \leq j \leq n$.

Solution.

We first give an NTM to show the problem is in NP.

Algorithm 11 A polytime NTM for 3SAT-under-ρ

Input: $\langle \phi, j \rangle$

1: Non deterministically pick an assignment $\alpha \in \{0,1\}^n$ to the variables of ϕ.
2: If α doesn't set $x_j = 1$: reject.
3: Check α satisfies ϕ: If not reject.
4: Accept.

Solution.

We now show 3SAT-under- ρ is NP-hard. We show 3-Sat $\leq_P$ 3SAT-under- ρ.
We're given $\langle \phi \rangle$ where ϕ is a 3-CNF formula with variables $x_1, \ldots, x_n$. We output $f(\langle \phi \rangle) = \langle \phi \wedge x_{n+1}, n+1 \rangle$ where x_{n+1} is a new variable.

Here are some details so that you can see why this is correct.

First, it's easy to see f can be computed in polynomial time, you only need to write down $\phi \wedge x_{n+1}$ and $n+1$. So this is $O(|\phi|)$ time.

Assume $\langle \phi \rangle \in$ 3-SAT. Then there exists some assignment $\alpha \in \{0,1\}^n$ which satisfies ϕ. Then clearly, $\phi \wedge x_{n+1}$ is satisfied by $\alpha' := \alpha \circ 1$. So $\phi \wedge x_{n+1}$ has a satisfying assignment which sets $x_{n+1} = 1$. So $f(\langle \phi \rangle) \in$ 3SAT-under- ρ

If $\langle \phi \wedge x_{n+1}, n+1 \rangle \in$ 3SAT-under- ρ. Then there's some assignment α' which satisfies the formula $\phi \wedge x_{n+1}$ and sets $x_{n+1} = 1$. So α' must satisfy ϕ. So we can restrict α' to $x_1, \ldots, x_n$ to get an assignment which satisfies ϕ. So $\langle \phi \rangle \in$ 3-SAT.

(f) $\text{FIND2} - \text{3SAT} = \{\langle\phi\rangle \mid \phi \text{ is a 3-CNF formula and it has at least 2 satisfying assignments}\}$

Solution: See HW5 exercise 2 (it's the same problem, but called DOUBLE-SAT).[1]

(g) William needs to solve the following ExamDesign problem. He has a list of problems, and he knows for each problem which students will really enjoy that problem. He needs to choose a subset of problems for the exam such that for each student in the class, the exam includes at least one question that student will really enjoy. On the other hand, he does not want to spend the entire summer grading an exam with dozens of questions, so the exam must also contain as few questions as possible. Prove that the ExamDesign problem is NP-hard.

Solution: ask William.

[1] Technically, DOUBLE-SAT is about any CNF formula, while here the problem is about 3-CNF formulas. However, the verifier we gave in HW5 still works. You can check the mapping reduction works too.