Computability and Computational Complexity, A.Y. 2024–2025

Guide to the answers

Wednesday, June 11, 2025

Exercise 1

1.1) Prove that the two following problems belong to **NP**:

P_1: Given a finite list L of unordered pairs of persons, where $\{a, b\} \in L$ means "a and b know each other", and a positive integer k, is there an individual who knows at least k other people?

P_2: Given a finite list L of unordered pairs of persons, where $\{a, b\} \in L$ means "a and b know each other", and a positive integer k, is there a group of k people who all know each other?

1.2) Prove the two following statements:

If $P_2 \in$ **P** then **P** $=$ **NP**.

If P_1 is **NP**-complete then $P_2 \in$ **P**.

Hint — *Here is a list of languages that you can assume to be **NP**-complete without having to prove it: SATISFIABILITY, 3-SATISFIABILITY, CLIQUE, INDEPENDENT SET, INTEGER LINEAR PROGRAMMING, VERTEX COVER, 3-VERTEX COLORING, SUBSET SUM, KNAPSACK, HAMILTONIAN PATH, DIRECTED HAMILTONIAN CYCLE, HAMILTONIAN CYCLE, TRAVELING SALESMAN PROBLEM.*

Solution 1

1.1)

- "$P_1 \in$ **NP**": a polynomial certificate is simply the ID x of the individual who knows at least k others. To check the certificate, we just need to count the number of distinct pairs in L that contain x. This can be solved polynomially with list scans (the exact complexity depending on what guarantees we have on the list, e.g.: are any pairs repeated?).
  Otherwise, we can directly prove that ¶$_1 \in$ **P** (see the second proposition in point 1.2, where we need to prove it anyway).
- "$P_2 \in$ **NP**": this time, a polynomial certificate can be given as a list of IDs of k individuals $x_1, \dots, x_k$. After checking that all IDs are distinct, we verify that $\{x_i, x_j\} \in L$ for $i, j = 1 \dots k$ (with many possible, but irrelevant, optimizations).

1.2)

- "If $P_2 \in$ **P** then **P** $=$ **NP**": P_2 is clearly equivalent to CLIQUE. In particular, CLIQUE $\leq_P$ P_2 Therefore, $P_2 \in$ **P** $\Rightarrow$ CLIQUE $\in$ **P**. However, CLIQUE is **NP**-complete, therefore any other problem in **NP** is polynomially reducible to it.
- "If P_1 is **NP**-complete then $P_2 \in$ **P**": we can easily prove that $P_1 \in$ **P** by providing an algorithm for it: set a counter $c_i = 0$ for each individual i, scan L and for every $\{i, j\} \in L$ increment both c_i and c_j. As soon as a counter get to k, accept; if the scan terminates, reject. If P_1 were **NP**-complete, then every problem in $P \in$ **NP** would be reducible to it, and would therefore be polynomial.

Exercise 2

For each of the following properties of Turing machines $\mathcal{M}$, prove whether it is recursive or not. Whenever possible, use Rice's theorem.

2.1) $\mathcal{M}$ either performs less than 100 steps or runs forever when executed on an empty tape;

2.2) $\mathcal{M}$ never visits any state more than ten times when executed on an empty tape;

2.3) $\mathcal{M}$ recognizes Turing machines with more states than alphabet symbols.

Solution 2

2.1) Non-recursive. The property is not trivial because clearly there are machines with that property and machines without it, however it is not semantic (e.g., a machine might recognize the empty language and reject immediately, or run 101 dummy states and then reject), therefore we cannot use Rice's theorem. We could use a TM computing $\mathcal{P}_1$ to test for $\mathcal{M} \in \text{HALT}_\varepsilon$ in two ways:

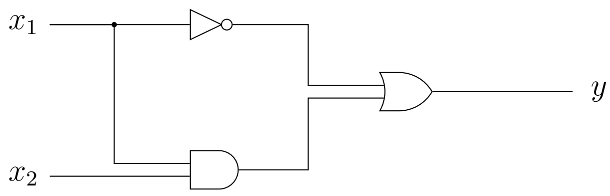
- create $\mathcal{M}'$ by adding 100 dummy states at the beginning of $\mathcal{M}$, so that $\mathcal{M}'$ must run for at least 100 steps and behaves exactly like $\mathcal{M}$ in every other aspect, then test $\mathcal{M}' \in \mathcal{P}_1$;
- or test $\mathcal{M} \in \mathcal{P}_1$ and, if yes, simulate a run of $\mathcal{M}(\varepsilon)$ for at most 100 steps to see whether it halts within 100 steps; if not, it will run forever.

2.2) Recursive. Again, the property is neither trivial nor semantic, so Rice's Theorem cannot be applied. However, to check whether $\mathcal{M} \in \mathcal{P}_2$ we just need to maintain a counter for every state of $\mathcal{M}$ and simulate the computation $\mathcal{M}(\varepsilon)$ increasing a counter whenever the computation visits the corresponding state. As soon as one counter exceeds 10, we reject (if $\mathcal{M}$ runs forever, we are guaranteed that this will eventually happer, because the number of states is finite). If the computation halts before any counter exceeds 10, then we accept.

2.3) Non-recursive. The definition clearly defines a language (the actual meaning of the definition is "$\mathcal{M}$ recognizes the language of all TM descriptions that…"), but it is not trivial (it is possible to build a TM with the property of recognizing TMs with more states than symbols). Rice's theorem applies.

Exercise 3

Consider the following Boolean circuit representing a Boolean function $y = f(x_1, x_2)$:



3.1) Write the function f in terms of the Boolean operators $\wedge$ (and), $\vee$ (or) and $\neg$ (not) on the two variables x_1 and x_2.

3.2) Write a 3CNF formula on the three variables x_1, x_2 and y (and, if needed, other auxiliary variables for gate outputs) that is satisfiable if and only if $y = f(x_1, x_2)$ (i.e., if x_1, x_2 and y have values that are compatible with the given Boolean circuit).

Hint — *Point 3.2 can be solved in two ways: by directly writing the dependency as $y \Leftrightarrow f(x_1, x_2)$ and applying Boolean algebraic rules to work out a 3CNF formula, or by writing down a 3CNF formula for each gate and requiring them all to be true. The second way is the one discussed in the course.*

Solution 3

3.1) Just translate the circuit into a Boolean formula:

$$f(x_1, x_2) = \neg x_1 \vee (x_1 \wedge x_2).$$

The formula can actually be simplified (but non requested in the exercise) by distributing the "or", then removing the first clause, that is always true:

$$\begin{aligned} f(x_1, x_2) &= (\neg x_1 \vee x_1) \wedge (\neg x_1 \vee x_2) \\ &= \neg x_1 \vee x_2. \end{aligned}$$

3.2) We can answer this in at least three ways (any method is acceptable):

- As suggested in the exercise text:

$$\begin{aligned} & y \Leftrightarrow (\neg x_1 \vee x_2) \\ \equiv\ & \big(y \Rightarrow (\neg x_1 \vee x_2)\big) \wedge \big((\neg x_1 \vee x_2) \Rightarrow y\big) \\ \equiv\ & (\neg y \vee \neg x_1 \vee x_2) \wedge \big(\neg(\neg x_1 \vee x_2) \vee y\big) \\ \equiv\ & (\neg y \vee \neg x_1 \vee x_2) \wedge \big((x_1 \wedge \neg x_2) \vee y\big) \\ \equiv\ & (\neg y \vee \neg x_1 \vee x_2) \wedge (x_1 \vee y) \wedge (\neg x_2 \vee y). \end{aligned}$$

- By following the second suggestion: define a variable for the outputs of the two "internal" gates (e.g., $g_\neg$ for the "not", $g_\wedge$ for the "and" gate), then write a conjunction of the CNFs for the single gates:

$$\begin{aligned} & (g_\neg \Leftrightarrow \neg x_1) \wedge (g_\wedge \Leftrightarrow (x_1 \wedge x_2)) \wedge (y \Leftrightarrow (g_\neg \vee g_\wedge)) \\ \equiv\ & (\neg g_\neg \vee \neg x_1) \wedge (g_\neg \vee x_1) \\ & \wedge (\neg g_\wedge \vee x_1) \wedge (\neg g_\wedge \vee x_2) \wedge (g_\wedge \vee \neg x_1 \vee \neg x_2) \\ & \wedge (\neg y \vee g_\neg \vee g_\wedge) \wedge (y \vee \neg g_\neg) \wedge (y \vee \neg g_\wedge). \end{aligned}$$

  This is the standard, "foolproof" way to do it, but it is much more cumbersome and requires more variables.

- Another method, mentioned during the course but not in the notes, uses the circuit's truth table:

x_1	x_2	y
F	F	T
F	T	T
T	F	F
T	T	T

Therefore, the requested CNF would have the following truth table, where the "true" rows are the ones that appear in the table above:

x_1	x_2	y	CNF
F	F	F	F
F	F	T	T
F	T	F	F
F	T	T	T
T	F	F	T
T	F	T	F
T	T	F	F
T	T	T	T

Finally, a disjunctive clause can be used to exclude one line. For example, $\neg x_1 \vee x_2 \vee y$ is true for all lines with the exception of the fifth one (TFF). Therefore, our CNF can be described by the following:

$$
\begin{aligned}
& (x_1 \vee x_2 \vee y) \quad \text{(exclude the 1st line)} \\
\wedge \quad & (x_1 \vee \neg x_2 \vee y) \quad \text{(exclude the 3rd line)} \\
\wedge \quad & (\neg x_1 \vee x_2 \vee \neg y) \quad \text{(exclude the 6th line)} \\
\wedge \quad & (\neg x_1 \vee \neg x_2 \vee y) \quad \text{(exclude the 7th line)}
\end{aligned}
$$

Note that with further manipulation this formula can be reduced to the first one (collect $x_1 \vee y$ from the first two clauses, and collect $\neg x_2 \vee y$ from the second and the fourth clause).