Computability and Computational Complexity, A.Y. 2025–2026

Guide to the answers
(revised version based on student answers to the test)

Wednesday January 7, 2026 — revised January 14, 2026

Exercise 1

1.1) Is the HALT language **NP**-hard?

1.2) Is HALT **NP**-complete?

Hint — *Motivate your answer starting from the relevant definitions. While this question has a definitely correct answer, you are allowed to express well-motivated doubts!*

Solution 1

1.1) To prove that HALT is **NP**-hard we must show that for every language $L \in \mathbf{NP}$ there is a polynomial-time reduction from L to HALT (i.e., $L \leq_{\mathbf{P}}$ HALT).
Let $\mathcal{M}_L$ be a TM that accepts L (it exists because L is decidable):

$$\mathcal{M}_L(x) = \begin{cases} 1 & \text{if } x \in L \\ \infty & \text{if } x \notin L. \end{cases}$$

Therefore, $x \in L$ if and only if $(\mathcal{M}_L, x) \in$ HALT. Hence, given a string x we can transform it into an instance of HALT by appending it to a description of $\mathcal{M}_L$. In other words, define f as the following function:

$$\begin{array}{rcl} f : \Sigma^* & \to & \Sigma^* \\ x & \mapsto & (\lfloor \mathcal{M}_L \rfloor, x) \end{array}$$

Function f just adds a fixed-length string to the existing input x, so it runs in $O(|x|)$ time (it may need to traverse the input before adding the TM description). Therefore it runs in polynomial time.
Moreover, $x \in L \Leftrightarrow f(x) \in$ HALT.
Hence, f is a polynomial-time reduction from L to HALT.

1.2) Since HALT is not recursive, it is not in **NP**, therefore it cannot be **NP**-complete.

Remarks

- Something in the lines of "HALT is not even computable, therefore the question makes no sense" is *almost* acceptable, and earns a reasonably high mark. However, one should notice that the definition of **NP** hardness does not refer to any complexity class for the target language.
- Observe that the same reduction applies to any recursively enumerable language, not only to **NP**.
- Pay attention to the direction of the reduction: we need to reduce any $L \in \mathbf{NP}$ to HALT, not vice versa (which would be impossible).

Exercise 2

An instance (k, G) of the CLIQUE language (where k is the required clique size and G is an undirected graph) is encoded with alphabet $\Sigma = \{\text{␣}, \texttt{0}, \texttt{1}, \texttt{+}\}$ as follows:

- the binary encoding of k,
- followed by a blank cell,
- followed by the adjacency matrix of G, where every row is represented as a sequence of `0`'s and `1`'s, and rows are separated by a + symbol.

2.1) Does the instance encoded by the following string belong to the CLIQUE language or not?

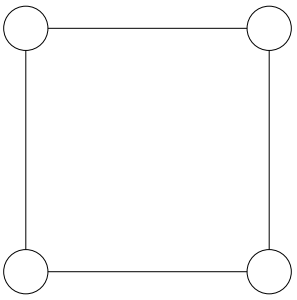
`11␣0101+1010+0101+1010`

2.2) Write a Turing machine that performs a reduction from the INDEPENDENT SET to the CLIQUE language, using the same encoding for the instances (k, G) of both languages. What is the time complexity of the reduction?

Solution 2

2.1) The string encodes the integer $k = 3$ and the adjacency matrix defines a 4-node loop (a square):

$$E = \begin{pmatrix} 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 \end{pmatrix}$$

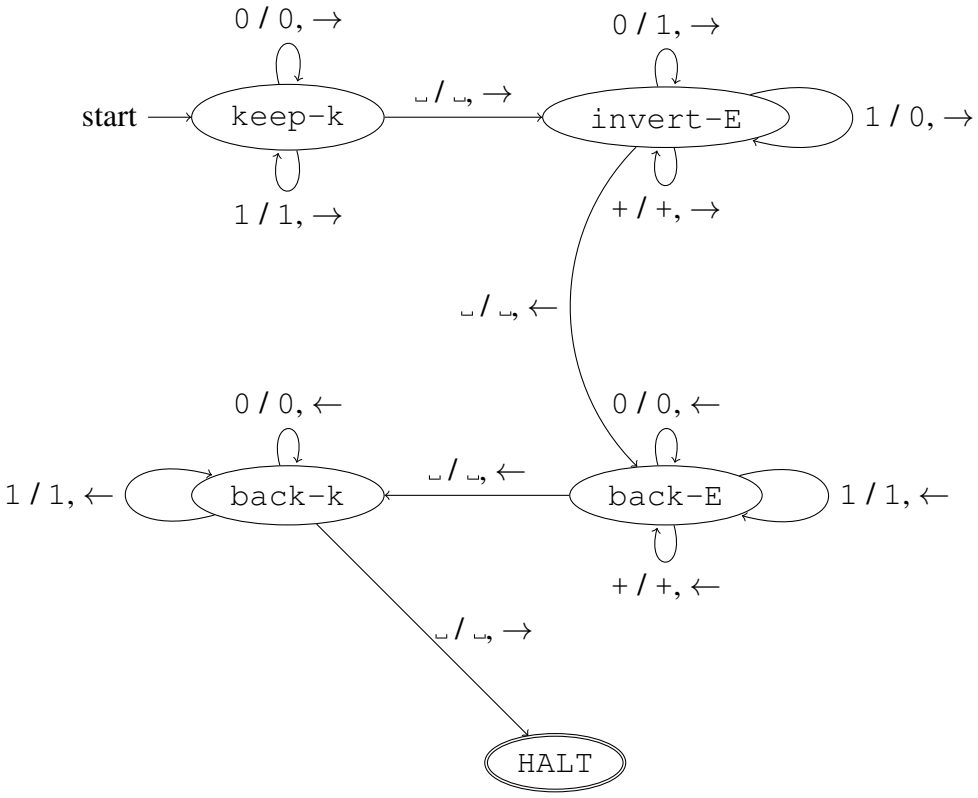


The graph doesn't contain a 3-clique, therefore the encoded instance does not belong to CLIQUE.

2.2) An independent set in $G = (V, E)$ corresponds to a clique in the complementary graph $\bar{G} = (V, \bar{E})$. Therefore the TM just needs to keep the same value for k and invert the adjacency matrix. Assuming that the initial position is the leftmost symbol of the input, the machine only needs to leave the input unchanged until the first blank (state `keep-k`), then invert all 0's and 1's until the second blank (state `invert-E`). Then we can complete the work by resuming the initial position (back two blanks, states `back-E` and `back-k`):

	␣	0	1	+
`keep-k`	␣, →, `invert-E`	0, →, `keep-k`	1, →, `keep-k`	—
`invert-E`	␣, ←, `back-E`	1, →, `invert-E`	0, →, `invert-E`	+, →, `invert-E`
`back-E`	␣, ←, `back-k`	0, ←, `back-E`	1, ←, `back-E`	+, ←, `back-E`
`back-k`	␣, →, `HALT`	0, ←, `back-k`	1, ←, `back-k`	—

Observe that the inverted adjacency matrix contains loops (arcs from a node to itself), however their presence is irrelevant to the definitions of clique and independent set.
Here is the corresponding state transition diagram:



Remarks

- Observe that the first question isn't just asking if the given string is a valid encoding (it is), but if the string encodes a *positive* instance of the CLIQUE problem, i.e., a graph G and a number k such that G has a clique of size k.
- The proposed TM looks more complex than strictly necessary because it also moves back to the initial position. Halting immediately after the graph inversion (so, a TM having just the `keep-k` and `invert-E` states) is acceptable and earns full marks.

Exercise 3

For each of the following properties of Turing machines $\mathcal{M}$, prove whether it's computable or not. Invoke Rice's theorem whenever it applies.

$$
\begin{aligned}
\mathcal{P}_1 &= \{\mathcal{M} : \mathcal{M} \text{ accepts at least one input string}\} \\
\mathcal{P}_2 &= \{\mathcal{M} : \mathcal{M} \text{ accepts all input strings}\} \\
\mathcal{P}_3 &= \{\mathcal{M} : \mathcal{M} \text{ computes the Kolmogorov complexity of the input string}\} \\
\mathcal{P}_4 &= \{\mathcal{M} : \mathcal{M} \text{ halts on all input strings of length} \leq 10\} \\
\mathcal{P}_5 &= \{\mathcal{M} : \mathcal{M} \text{ halts in no more than 10 steps on all input strings}\}.
\end{aligned}
$$

Solution 3

- $\mathcal{P}_1 = \{\mathcal{M} : \mathcal{M}$ accepts at least one input string$\}$ — **not computable**.
  The property refers to the accepted language: it can be rephrased as $L(\mathcal{M}) \neq \emptyset$, therefore it is semantic.
  More clearly, if two TMs accept the same language L, then either $L = \emptyset$ (thus neither machine has the property cP_1) or $L \neq \emptyset$ (and both machines have the property).
  The property is not trivial (there are machines that accept at least one string, and machines that do not accept any string). Therefore, Rice's theorem applies and the property is not recursive.

- $\mathcal{P}_2 = \{\mathcal{M} : \mathcal{M}$ accepts all input strings$\}$ — **not computable**.
  Again, the property is semantic (it can be rephrased as $L(\mathcal{M}) = \Sigma^*$) and not trivial (there are machines that accept all strings and machines that don't). Thus Rice's theorem applies.

- $\mathcal{P}_3 = \{\mathcal{M} : \mathcal{M}$ computes the Kolmogorov complexity of the input string$\}$ — **computable**.
  We know that Kolmogorov complexity is not computable; therefore the property is false for all machines. In other words, $\mathcal{P}_3 = \emptyset$ is trivially computed by any machine that rejects all inputs.

- $\mathcal{P}_4 = \{\mathcal{M} : \mathcal{M}$ halts on all input strings of length $\leq 10\}$ — **not computable**.
  We know that the halting problem is not computable even for inputs of length 0.
  More formally, we can prove that $\mathcal{P}_4$ is not computable in many ways:

  - By redefining the acceptance notion so that the act of halting implies acceptance (i.e., by saying that whenever a TM halts then the input is to be considered accepted). By doing this, $\mathcal{P}_4$ becomes semantic and non- trivial.
  - By reducing the HALT_ε problem to it: given an arbitrary TM $\mathcal{M}$, to test $\mathcal{M} \in \text{HALT}_\varepsilon$, create $\mathcal{M}'$ that erases the input and then runs $\mathcal{M}$. Then, $\mathcal{M}$ halts on the empty input if and only if $\mathcal{M}'$ halts on every input (and in particular for all inputs of size up to 10):
    $$\mathcal{M} \in \text{HALT}_\varepsilon \Leftrightarrow \mathcal{M}' \in \mathcal{P}_4.$$
  - By identifying a TM that has property $\mathcal{P}_4$ (e.g., any machine that always halts) and mimicking the proof of Rice's Theorem.

- $\mathcal{P}_5 = \{\mathcal{M} : \mathcal{M}$ halts in no more than 10 steps on all input strings$\}$ — **computable**.
  The property is clearly not semantic (a machine halting in no more than 10 steps can easily be transformed into an equivalent machine without the property by adding 10 dummy states either on startup or before the halting state). So, Rice's theorem cannot be invoked. A simple method to verify the property is to simulate the execution of the $\mathcal{M}$ on every possible input and reject whenever a simulation lasts longer than 10 (simulated) steps. Although the number of input strings is infinite, the only relevant portion of the input is given by the tape cells reachable within 10 steps; therefore, the number of input strings to be tested is finite (at most 20 symbols, considering that TMs are bidirectional).

Remarks

- A short reason as to *why* each property is (or isn't) semantic or trivial would be appreciated.