Computability and Computational Complexity, A.Y. 2023–2024

Guide to the answers

Monday, January 8, 2024

Exercise 1

Let L be a **finite, non-empty** language on the alphabet $\Sigma = \{0, 1\}$.
For each of the following propositions say whether it is true or false, and briefly motivate your answer.

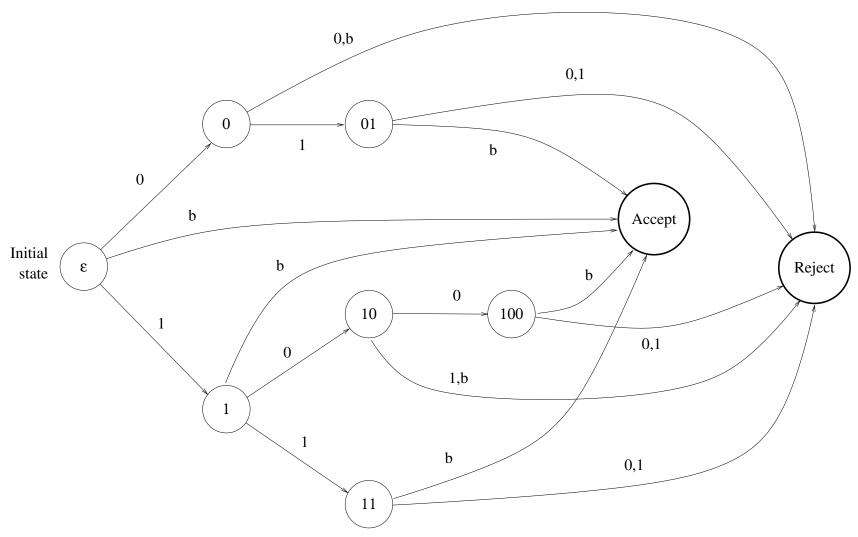
1. L is computable.
2. The property "$\mathcal{M}$ decides L," where $\mathcal{M}$ is a deterministic Turing machine, is recursive.
3. The property "the string representing, in binary notation, the number of steps of $\mathcal{M}(\varepsilon)$ before halting belongs to L," where $\mathcal{M}$ is a deterministic Turing machine, is recursive.
4. The property "the string representing, in binary notation, the number of states of $\mathcal{M}$ belongs to L," where $\mathcal{M}$ is a deterministic Turing machine, is recursive.

Solution 1

1. **True** — Since L is finite, it is clearly decidable: a machine just needs to compare the input against a finite number of strings, accepting as soon as a comparison succeeds, or rejecting after all comparisons have failed (a sequence of `if` statements).
   Better yet, the machine may scan the input left to right, encoding the string in its own state, rejecting as soon as it becomes clear that the encoded string is not in L, accepting at the end.
   For instance, let $L = \{\varepsilon, 1, 01, 11, 100\}$. Then the following pseudocode clearly decides whether $x \in L$:

```
on input $x$:
  if $x = \varepsilon$ then accept;
  if $x =$ "1" then accept;
  if $x =$ "01" then accept;
  if $x =$ "11" then accept;
  if $x =$ "100" then accept;
  reject;
```

   As an alternative, the following TM decides L by scanning the input and encoding it in the state name as a mnemonic help; as long as the state name is a prefix of some string in L then the machine proceeds; when it finds the first blank, if the state name encodes a string in L then it accepts; in all other cases it rejects:



Notice that the arrows are only marked by the input symbol: the machine always moves right and keeps the same symbols on the tape (it is basically a deterministic finite-state automaton).

2. **False** — "$\mathcal{M}$ decides L" is clearly semantic and not trivial, therefore Rice's Theorem applies: the property is undecidable.

3. **True** — The property refers to the behavior of a machine $\mathcal{M}$ on the empty tape, therefore it does not refer to the language recognised by it: it is not a semantic property, thus Rice's Theorem does not apply. Let m be the largest number encoded in binary form by strings in L. We just need to simulate machine $\mathcal{M}$ for at most m steps. If it halts in n steps (with $n < lem$), we just check whether the binary encoding of n is in L (which we can do, because L is computable), otherwise $\mathcal{M}$ clearly doesn't halt within a number of steps encoded in L (maybe it doesn't halt at all). Therefore the property is recursive.
   For instance, if $L = \{\varepsilon, 1, 01, 11, 100\}$ then a machine has the property iff it halts after 1, 3, or 4 steps on the empty tape. Otherwise it doesn't. We just need to simulate $\mathcal{M}$ for at most 4 steps.

4. **True** — Just count the number of states, encode it into x and decide whether $x \in L$.
   If $L = \{\varepsilon, 1, 01, 11, 100\}$, then a machine has the property iff it has 1, 3, or 4 states.

Observations

- The sample pseudocode and TM above are just shown for clarification, but there was no need to provide an algorithm for point 1.
  Just the observation that comparing a string against a finite number of alternatives is obviously feasible would achieve full marks.

- Answering "false" to both points 1 and 2 is a significant logical fallacy: if L were uncomputable, then property "$\mathcal{M}$ decides L" would be trivial, because no machine has it, and therefore it would be trivially decidable by the machine that always rejects.

- For point 3, we don't need to know if $\mathcal{M}(\varepsilon)$ halts: we just need to simulate it for a maximum number of steps (4 in the example). Therefore, HALT_ε is not Turing-reducible to the property.

Exercise 2

Let L be a **finite, non-empty** language on the alphabet $\Sigma = \{0, 1\}$.
For each of the following propositions say if it is true, false or (to the best of our knowledge) unknown, and briefly motivate your answer.

1. $L \in \mathbf{P}$.
2. $L \in \mathbf{L}$.
3. L is polynomial-time reducible to 3SAT.
4. 3SAT is polynomial-time reducible to L.

Solution 2

1. **True** — Verifying whether the input x belongs to a finite series of alternatives is clearly polynomial with respect to the number of involved strings and to their length. For instance, the pseudocode of the example requires one comparison for every string in L in the worst case; the TM of the example runs for as many steps as the longest string in L (plus one). Therefore, $L \in \text{DTIME}(1) \subset \mathbf{P}$.
2. **True** — Comparing a string with a hardcoded one can be done by just scanning the input tape, without ever writing anything (or, if we want to write an acceptance bit, writing in constant space). Therefore, $L \in \text{DSPACE}(1) \subset \mathbf{L}$.
   Both the pseudocode and the TM of the example above are constant time.
3. **True** — 3SAT is **NP**-complete, therefore any language in **NP**, including L, can be reduced to it in polynomial time.
   The reduction is very simple: given input x, if $x \in L$ then produce a satisfiable 3-CNF formula, e.g. $(x_1 \vee x_2 \vee x_3)$, otherwise produce an unsatisfiable one, e.g. $(x_1 \vee x_1 \vee x_1) \wedge (\neg x_1 \vee \neg x_1 \vee \neg x_1)$.
4. **Unknown** (probably false) — If 3SAT is polynomial-time reducible to L, it means that L itself is **NP**-complete. Since we know that $L \in \mathbf{P}$, this is true if and only if $\mathbf{P} = \mathbf{NP}$. Hence the requested reduction is, to the best of our knowledge, unlikely.

Observations

- To answer the questions we just needed to know that L is finite.
- While L is both constant time and constant space, there was no need to observe that to get full marks on the point.
- One could start by answering point 2 and then just observe $\mathbf{L} \subseteq \mathbf{P}$.
- For point 3, no need to provide a reduction: just stating why it exists is enough.

$\boxed{\textbf{Exercise 3}}$

Define the probabilistic time complexity class **RP** and prove the following inclusions:

$$\mathbf{P} \subseteq \mathbf{RP}, \qquad \mathbf{RP} \subseteq \mathbf{NP}, \qquad \mathbf{RP} \subseteq \mathbf{BPP}.$$

$\boxed{\textbf{Solution 3}}$

See the lecture notes and the exercises. In short:

1. $L \in$ **RP** if there is $0 < \varepsilon < 1$ and a NDTM $\mathcal{N}$ that decides L in polynomial time, with the further restriction that if $x \in L$ the ratio of accepting computations in $\mathcal{N}(x)$ is bounded from below by ε.

2. **P** is the special case in the above definition where $\mathcal{N}$ is deterministic, therefore if $x \in L$ the ratio of accepting computations is 1 (which is greater than ε by definition).

3. The above definition of **RP** is precisely **NP** with further restrictions.

4. **BPP** puts different restrictions on acceptance ratio ($0 < \varepsilon < \frac{1}{2}$):

   - if $x \in L$, the ratio of accepting computations is bounded from below by $\frac{1}{2} + \varepsilon$, which is apparently stronger than the bound required by **RP**; however, we just need to iterate the computation of $\mathcal{N}(x)$ for a constant number of times to boost the ratio of accepting computations above any value (less than 1);
   - if $x \notin L$, the ratio of accepting computations is bounded from above by $\frac{1}{2} - \varepsilon$; this is satisfied by any language in **RP**, since in that case the ratio is zero.