Computability and Computational Complexity, A.Y. 2019–2020

Solution outlines to the written exam

Mauro Brunato

Wednesday, January 15, 2020

Exercise 1

Consider the following language in $\{0,1\}^*$:

$$K = \{0^n 1^n : n \in \mathbb{N}\} = \{\varepsilon, 01, 0011, 000111, 00001111, 0000011111, \dots\}$$

i.e., all strings composed by a sequence of zeroes followed by the *same* number of ones.

1.1) Write a single-tape Turing Machine with alphabet $\Sigma = \{\text{␣}, 0, 1\}$ that recognizes K.

1.2) Prove or disprove the decidability of each of the following properties of TMs:

$$\begin{aligned}
\mathcal{P}_1 &= \{\mathcal{M} : \mathcal{M} \text{ decides } K\}, \\
\mathcal{P}_2 &= \{\mathcal{M} : \mathcal{M} \text{ decides } K \text{ in less than 100 steps}\}, \\
\mathcal{P}_3 &= \{\mathcal{M} : \mathcal{M} \text{ decides } K \cap \Sigma^{100} \text{ (i.e., strings in } K \text{ not longer that } 100 \text{ symbols)}\}.
\end{aligned}$$

For 1.1 use any notation you like, and encode acceptance and rejection as you prefer (0/1 on tape, two different halting states, etc.).

Solution 1

1.1) The simplest, although, not the most efficient, machine just keeps erasing the leftmost 0 and the rightmost 1 until the input is empty or some unexpected symbol appears (e.g., leftmost 1, righmost 0, blank when a 1 should be erased).

We assume that the input is a contiguous string of 0's and 1's, surrounded by blanks, and that the machine starts on the leftmost input symbol. Here is the transition table:

	␣	0	1
erase-leftmost-0	␣/→/accept	␣/→/go-right	1/→/reject
go-right	␣/←/erase-rightmost-1	0/→/go-right	1/→/go-right
erase-rightmost-1	␣/←/reject	0/←/reject	␣/←/go-left
go-left	␣/→/erase-leftmost-0	0/←/go-left	1/←/go-left

An encoding suitable for the TM simulator seen in class[1] is:

```
erase-leftmost-0 0 _ r go-right    ; found and erased a 0
erase-leftmost-0 1 1 r halt-reject ; unexpected 1
erase-leftmost-0 _ _ r halt-accept ; the input has been consumed

go-right 0 0 r go-right           ; keep skipping the input
go-right 1 1 r go-right
```

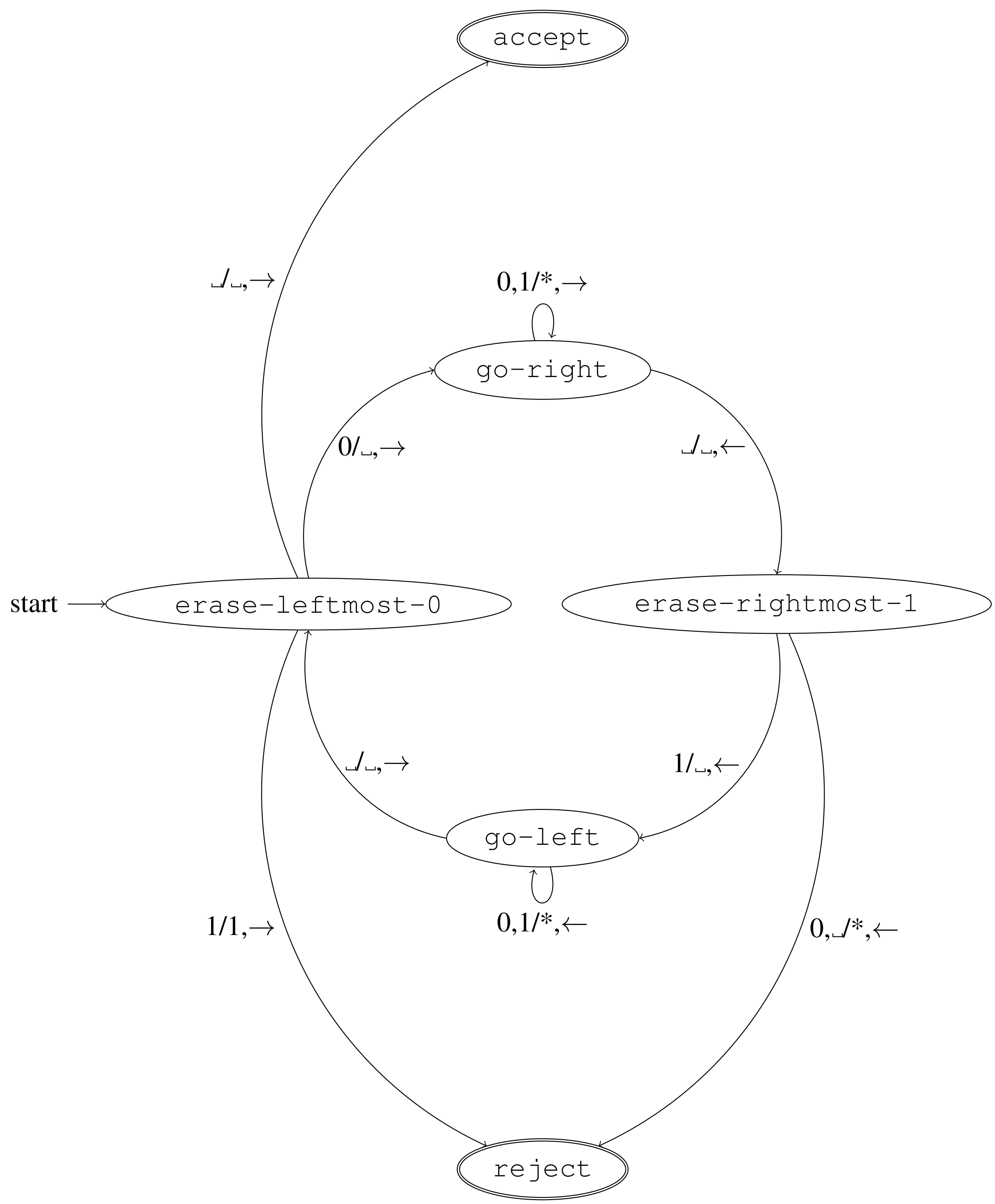
[1] `http://morphett.info/turing/turing.html`

```
go-right _ _ l erase-rightmost-1 ; found the end of the input

erase-rightmost-1 1 _ l go-left     ; found and erased a 1
erase-rightmost-1 0 0 l halt-reject ; unexpected 0
erase-rightmost-1 _ _ l halt-reject ; unexpected blank

go-left 0 0 l go-left          ; keep skipping the input
go-left 1 1 l go-left
go-left _ _ r erase-leftmost-0 ; found the beginning of the input
```

The same machine as an automaton form:



1.2)

- Property $\mathcal{P}_1$ is clearly semantic ($\mathcal{M} \in \mathcal{P}_1 \Leftrightarrow L(\mathcal{M}) = K$) and is not trivial (there is at least one machine that decides K and at least one that doesn't); therefore, by Rice's Theorem, it is undecidable.

- A TM limited to 100 steps cannot decide K. Consider, e.g., the string $s_1 = 0^{1000}1^{1000} \in K$. A TM limited to 100 steps wouldn't be able to read the whole input, therefore it wouldn't be able to tell s_1 from $s_2 = 0^{1000}1^{1001} \notin K$. Therefore, $\mathcal{P}_2 = \emptyset$, hence it is trivially computable by a TM that always rejects.

- Again, $\mathcal{P}_3$ is semantic and non-trivial, thus uncomputable.

Observations

- Many other TMs are possible for 1.1.

- Observe that, since 1.1 requires the TM to just *recognize* K, rejection could be replaced by a non-halting computation.

- As usual, there is a significant distinction between the computability of K and the computability of the property "This machine decides K".

- Property $\mathcal{P}_2$ doesn't just require the TM to halt after 100 steps, but also to decide K. Therefore, simulating the TM for 100 steps isn't enough: we also need to consider which inputs it should be simulated on.

- The fact that the language defining $\mathcal{P}_3$ is finite doesn't matter: Rice's theorem is still valid, because we wouldn't be able to always assert whether a TM would halt or not.

Exercise 2

Prove that K, the language defined in Exercise 1, belongs to the complexity class $\mathbf{L}$.

While 1.1 required a single-tape machine, class $\mathbf{L}$ *has a different assumption. Here, however, you are not asked to write down the TM: just a few lines of pseudocode will do.*

Solution 2

Since $\mathbf{L} = \text{DSPACE}(\log n)$, we need to describe an algorithm whose additional space is logarithmic wrt the input's size.
For instance, consider a 2-tape TM implementation of the following algorithm:

```
function K ( x)
    c ← 0
    while next input symbol in x is 0
        c ← c + 1
    while next input symbol in x is 1
        if c = 0
            reject and halt
        c ← n − 1
    if next input symbol in x is ␣ and c = 0
        accept and halt
    else
        reject and halt
```

The machine allocates a counter c in its working tape, initialized with 0, and scans the input: as long as it finds zeroes, it increases c; then as long as it finds ones, it decreases it. It fails when finding a zero following a one, or when c falls below zero (underflow), or if c is nonzero at the end of the input.
The function is clearly implementable on a two-tape TM, and just the counter c needs to be stored in the working tape. Observe that in the worst case c is increased once per input symbol, therefore its value is never larger that $|\boldsymbol{x}|$, so it requires at most $\lceil \log_2 |\boldsymbol{x}| \rceil = O(\log |\boldsymbol{x}|)$ binary digits.

Observations

- Remember: we talk about log*space*, not time.
- Using more than one counter and an index to scan the input is fine, as long as we use a constant number of $O(\log |\boldsymbol{x}|)$-bit variables.

Exercise 3

Let $L_1, L_2 \in \mathbf{NP}$. Does $L_1 \cup L_2 \in \mathbf{NP}$? Does $L_1 \cap L_2 \in \mathbf{NP}$? Why?

*Be as formal as you can, e.g.: "Since $L_1 \in$ **NP**, then there is a TM $\mathcal{M}_1$ such that…"*

Solution 3

Since $L_1 \in \mathbf{NP}$, then there is a NDTM $\mathcal{N}_1$ that decides L_1 in polynomial time. Same for L_2.
Given input x, to decide whether $x \in L_1 \cup L_2$ we just need a NDTM that accepts x whenever $\mathcal{N}_1$ *or* $\mathcal{N}_2$ accepts it:

- Store x for future use.
- Run $\mathcal{N}_1$ on input x. If $\mathcal{N}_1$ accepts, then accept and halt.
- Restore input x.
- Run $\mathcal{N}_2$ on input x.

This machine runs in time that is, in the worst case, the sum of the times of $\mathcal{N}_1(x)$ and $\mathcal{N}_2(x)$ plus the time to copy and restore x, therefore it is polynomial in $|x|$.
Likewise, to decide whether $x \in L_1 \cap L_2$ we need a NDTM that accepts x whenever $\mathcal{N}_1$ *and* $\mathcal{N}_2$ accepts it:

- Store x for future use.
- Run $\mathcal{N}_1$ on input x. If $\mathcal{N}_1$ rejects, then reject and halt.
- Restore input x.
- Run $\mathcal{N}_2$ on input x.

The worst-case runtime is the same of the previous machine.

Observations

- The fact that $L_1 \cap L_2$ is (in some sense) "smaller" than both L_1 and L_2 doesn't mean that it is "easier", nor that $L_1 \cup L_2$ is "harder".
- Also remember that the exercise doesn't cite completeness.

Exercise 4

Consider the following classical **NP**-complete languages:

$$\text{CLIQUE} = \{(G,k) : \text{Undirected graph } G \text{ has a completely connected subgraph of size } k\},$$
$$\text{INDSET} = \{(G,k) : \text{Undirected graph } G \text{ has a completely disconnected subgraph of size } k\}.$$

4.1) Describe a polynomial-time reduction from one language to the other.

4.2) Show that CLIQUE $\cap$ INDSET $\neq \emptyset$.

For 4.1, choose the direction you like. In 4.2, don't be afraid of simple answers: to show that a set is not empty, you just need to find an element in it.

Solution 4

4.1) See the notes: $G = (V, E)$ has a clique of size k if and only if $\bar{G} = (V, \bar{E})$ (same vertex set, complementary edge set) has an independent set of the same size.

4.2) We need to show that there is a graph G and an integer k such that G has both a clique of size k and an independent set of size k. Just take any nonempty graph G and $k = 1$:

$$(G, 1) \in \text{CLIQUE} \cap \text{INDSET}.$$

Observations

- Any example is OK, provided that the same graph contains both a k-clique and a k-indset for the *same* value of k.
- CLIQUE and INDSET are *languages*, in this case sets of instances in the form (G, k): to show that their intersection is not null, we must show that the *same* instance belongs both to CLIQUE and INDSET.