Introduction to Complexity - Problem Set #1

Clément Canonne (clc2200) - ccanonne@cs.columbia.edu

February 11, 2013

Problem 1

Let $L, L' \subset \Sigma^*$ be any two decidable languages[1], and $M = (Q, \Sigma, \Gamma, \delta, q_{\text{start}}, q_{\text{acc}}, q_{\text{rej}})$, $M' = (Q', \Sigma, \Gamma, \delta', q'_{\text{start}}, q'_{\text{acc}}, q'_{\text{rej}})$ be two TM deciding respectively L and L'.

Remark 1. Since the two languages are arbitrary, it is sufficient to show that $L \cup L'$, LL', $\Sigma^* \setminus L$ and $L \cap L'$ are decidable to prove that the class of decidable languages is closed under the four operations considered; furthermore, to do so, we can indifferently design single or k-tape Turing machines.

(a) Union We describe a 2-tape TM $M^\cup = (Q^\cup, \Sigma, \Gamma^2, \delta^\cup, q^\cup_{\text{start}}, q^\cup_{\text{acc}}, q^\cup_{\text{rej}})$ which decides $L \cup L'$ (where $Q \times Q' \subset Q^\cup$): on input $w \in \Sigma^*$, $M^\cup$ uses its two tapes to simulate *independently* M and M', and accepts whenever one of the two simulations enters an accepting state; it rejects if and only if both simulations enter their rejecting state. In more details, the latter can be achieved by defining transitions of the form

$$\begin{aligned}\delta^\cup((q_{\text{acc}}, q'), a, b) &= (q^\cup_{\text{acc}}, a, b, R)\\ \delta^\cup((q, q'_{\text{acc}}), a, b) &= (q^\cup_{\text{acc}}, a, b, R)\\ \delta^\cup((q_{\text{rej}}, q'_{\text{rej}}), a, b) &= (q^\cup_{\text{rej}}, a, b, R)\end{aligned}$$

for every $(q, q', a, b) \in Q \times Q' \times \Gamma \times \Gamma$. By definition, it is clear that $M^\cup$ accepts w iff either M or M' accepts it, that is if $w \in L \cup L'$; and it will always halt after a finite number of steps, since both M and M' are guaranteed to accept or reject any input after finitely many steps.

[1] Without loss of generality, we can assume that they are defined on the same alphabet Σ – if not, one can always take Σ to be the union of their respective alphabets, and any TM deciding L in the original alphabet can be easily adapted to decide L in the new alphabet, by just rejecting whenever one of the new symbols is encountered. The same remark applies to the tape alphabet Γ used by the two corresponding Turing machines.

(b) Concatenation below is described a 3-tape TM M° deciding the concatenation LL' of the two languages: on input $w \in \Sigma^*$ (hereafter, n will denote the size $|w|$ of the input).

- the first tape allows M° to keep track of the original input as well as the current "breakpoint" $i \in [n]$;
- the second one is used to simulate M on a prefix $w_1 \dots w_i$ of the input;
- the third one is used to simulate M' on a suffix $w_{i+1} \dots w_n$ of the input.

More precisely, for i from 0 to n:

(i) $w_1 \dots w_i \sqcup$ is copied on the second tape, where M is simulated until it halts;
(ii) if it accepts, $w_{i+1} \dots w_n \sqcup$ is copied on the third tape, where M' is simulated until it halts; if it accepts, M° accepts;
(iii) otherwise, i is "incremented" (actually, there is no need for a pointer, just introduce enough (yet finitely many, dependent only on $|\Gamma|$) states for the head in the first tape to keep track of the current position tried).

If the loop ends without ever accepting, M° rejects.

If M° accepts on input w, it means that there exists $i \in [n]$ such that both M accepts on $w_1 \dots w_i$ and M' accepts on $w_{i+1} \dots w_n$; i.e, $w \in LL'$. Similarly, if M° rejects, there is no such i – and $w \notin LL'$. Finally, M° always halts, as each iteration is performed in finitely many steps (since M and M' halt on every input string).

(c) Complementation on can build a TM $\bar{M}$ deciding $\bar{L} = \Sigma^* \setminus L$ from M, by setting $\bar{M} = (Q, \Sigma, \Gamma, \delta, q_{\text{start}}, \overline{q_{\text{acc}}} \stackrel{\text{def}}{=} q_{\text{rej}}, \overline{q_{\text{rej}}} \stackrel{\text{def}}{=} q_{\text{acc}})$ (that is, by swapping the accepting and rejecting states). On any input $w \in \Sigma'$, $\bar{M}$ halts because halting means reaching either q_{rej} or q_{acc}, which M does; and $\bar{M}$ accepts on w iff $\bar{M}$ reaches $\overline{q_{\text{acc}}} = q_{\text{rej}}$ on w, which happens iff to M reaches q_{rej} on w (M and $\bar{M}$ has same transitions and states) and thus rejects, which in turn occurs iff $w \notin L$:

$$\bar{M} \text{ always halts, and } (\bar{M} \text{ accepts on } w) \Leftrightarrow w \in \bar{L}$$

(d) Intersection we can use almost the same construction as in **(a)**, except that the 2-tape TM $M^\cap$ (deciding $L \cap L'$) we define now accepts iff both (simulations of) M, M' accept:

$$\begin{aligned} \delta^\cap((q_{\text{rej}}, q'), a, b) &= (q^\cap_{\text{rej}}, a, b, R) \\ \delta^\cap((q, q'_{\text{rej}}), a, b) &= (q^\cap_{\text{rej}}, a, b, R) \\ \delta^\cap((q_{\text{acc}}, q'_{\text{acc}}), a, b) &= (q^\cap_{\text{acc}}, a, b, R) \end{aligned}$$

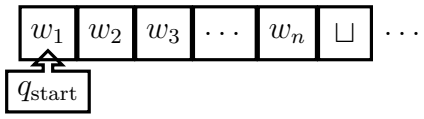
for every $(q, q', a, b) \in Q \times Q' \times \Gamma \times \Gamma$.

Remark 2. One could also use **(a)** and **(c)**, as $L \cap L' = \overline{\bar{L} \cup \bar{L'}}$.

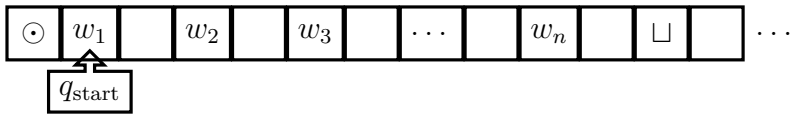
Problem 2: Doubly Infinite Tape Turing Machine

Single-∞ $\preceq$ Double-∞ Suppose you have a (regular) TM $M = (Q, \Sigma, \Gamma, \delta, q_{\text{start}}, q_{\text{acc}}, q_{\text{rej}})$; it can be simulated by a doubly-infinite tape TM $M^2 = (Q^2, \Sigma, \Gamma \cup \{\odot\}, \delta^2, q^2_{\text{start}}, q_{\text{acc}}, q_{\text{rej}})$ (where $\odot \notin \Gamma$) behaving this way: on input $w \in \Sigma^*$, M^2 writes $\odot$ on the rightmost cell before the input, then goes back to the first character of the input and enters state q_{start}; it then mimics M's behavior on the tape restricted to the cells on the right of the the marker $\odot$, using this marker to make sure it never uses the "left" portion of the tape.

Double-∞ $\preceq$ Single-∞ Any doubly-infinite tape TM M^2 with tape alphabet Γ can conversely be simulated by a regular TM M with extended tape alphabet $\Gamma \cup \{\odot\}$ (where $\odot \notin \Gamma$), for instance as follows: on input w



M will shift the tape contents one cell to the right, writing the special marker $\odot$ into the leftmost one, and add an empty cell every other cell containing a letter of the input:



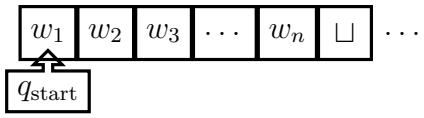
After this first step, M can now replicate the behavior of M^2, using every other cell of its singly infinite tape as the "left direction" of a doubly infinite tape, and the others as the "right direction". The $\odot$ marker allows M to determine if it has reached the left end of its tape, and thus should switch from "right" to "left" direction (or vice-versa). As a summary:

- cell 1: $\odot$
- cell $2i$: cell i of the double-∞ tape
- cell $2i + 1$: cell $-i$ of the double-∞ tape

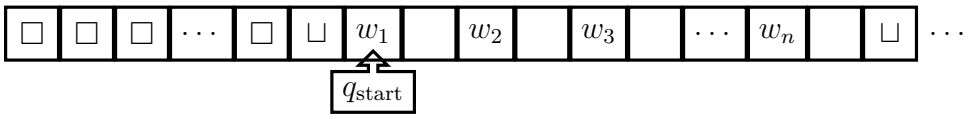
Problem 3: Write-Once Turing Machine (*)

A *write-once Turing machine* (WOTM) is a single-tape TM that can alter each tape cell at most once (including the input portion of the tape). To show that this variant Turing machine model decides the same class of languages as ordinary Turing machines, it is sufficient to explain how to simulate any TM M by a WOTM M' (the other way is obvious).

Fix any single-tape TM $M = (Q, \Sigma, \Gamma, \delta, q_{\text{start}}, q_{\text{acc}}, q_{\text{rej}})$. Consider the following behavior for M': on input $w \in \Sigma^*$ (with n denoting $|w|$), the tape in the initial configuration is of the form



The first thing it does is copy the input to an adjacent location on tape, using for each cell the only writing access allowed (because one needs to mark the input string copied so far); in doing so, M' actually *leaves every other cell blank*, to be used as a marker for the previous one (that is, each w_i is mapped to a couple of contiguous cells $(w_i, \sqcup)$):



(where $\square$ is any new symbol not in Γ). Now, to simulate one step of the execution of M, M' behaves as the original TM does (taking into account the fact that the leftmost part of its tape is "burnt"/no longer to be accessed, and that only every other cell contains relevant symbols from Γ). As long as M does not alter the contents, there is no problem; we turn now to the case where it is supposed to replace the content of a cell (say C), a, by a new value b. M' then proceeds as follows:

- write b in C's neighboring cell;
- move to the beginning of the "current tape", and start copying its contents to a new location (with the same "double cell" scheme), after the very end of the current used space (using the neighboring cells as marker locations for the copying process);
- when C is to be copied, write the new value b instead of a in the new location (as it has been kept in memory in the marker location; note that a special state is needed for this, since we can no longer use C's neighboring cell; but it is fine, as there is only one cell – i.e, C – for which this state is needed)
- continue like this until the entire contents of the tape (with the new value for C) has been copied into the new location, and the old tape contents have been erased and replaced by $\square$.

After this step, M''s tape contains the updated value for C (albeit the whole tape has been "moved" to a new location), and each new cell is "fresh" for the next step.

Problem 4: Read-Only Input Turing Machine (*)

To prove that single-tape TM's that cannot write on the portion of the tape (ROITM) containing the input string can only decide regular languages, it is sufficient to explain:

1. given a deterministic finite automaton $A = (Q, \Sigma, \delta, q_{\text{start}}, F)$ deciding a language $L_A \in \Sigma^*$, how to design a ROITM deciding L_A;
2. conversely, given a ROITM deciding some language L, how to build a DFA deciding L.

DFA $\preceq$ ROITM

Fix a DFA $A = (Q, \Sigma, \delta, q_{\text{start}}, F)$, and denote by $L_A \in \Sigma^*$ the language it decides. Consider the corresponding ROITM $M = (Q_M, \Sigma, \Gamma, \delta_M, q_{\text{start}}, q_{\text{acc}}, q_{\text{rej}})$ defined by $Q_M \stackrel{\text{def}}{=} Q \cup \{q_{\text{acc}}, q_{\text{rej}}\}$, $\Gamma \stackrel{\text{def}}{=} \Sigma \cup \{\sqcup\}$ and

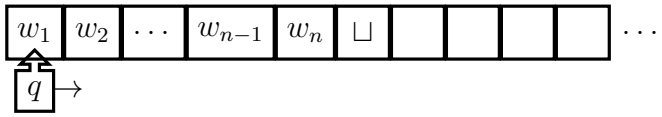
$$\forall (q, a) \in Q_M \times \Gamma, \quad \delta_M(q, a) = \begin{cases} (q_{\text{acc}}, a, R) & q \in F, a = \sqcup \\ (q_{\text{rej}}, a, R) & q \notin F, a = \sqcup \\ (\delta(q, a), a, R) & a \neq \sqcup \end{cases}$$

In other terms, the TM reads each character from the input once, from left to right, and updates its state as A would do until the end of the input is reached; at that point, it either goes to the accepting or rejecting state, depending whether its current state belongs to F. It is straightforward to see that this indeed defines a ROITM which always halts, and decides exactly L_A.

ROITM $\preceq$ DFA

Proof. Let $L \subset \Sigma^*$ be a language decided by a ROITM $M = (Q, \Sigma, \Gamma, \delta, q_{\text{start}}, q_{\text{acc}}, q_{\text{rej}})$. The high-level idea of the proof is to show there exists an algorithm using a *constant amount of memory* which decides L and reads its input in a one-way fashion; then, to conclude by observing that any such algorithm is known to be equivalent to some finite automaton.

We give a constructive proof of existence of this algorithm $\mathcal{A}$, describing how it behaves on any input $w = w_1 w_2 \dots w_n \in \Sigma^*$:



- At the beginning, $\mathcal{A}$ looks at the current letter and δ, and builds a lookup table Φ_1 of size $|Q|^{|Q|}$ which contains all transitions to the right (in more details, the mapping from Q to Q specifying, given a state q with which the cell is entered from the right, either the final state or the new state q' which M enters when moving to the right again).
- During the execution, $\mathcal{A}$ simulates M, with that difference that *it never moves back to the left* – it is effectively one-way. To achieve that, say the head of M is currently on the i^{th} cell C_i, and that by induction we have in memory (only) the lookup table Φ_{i-1} specifying, *for every possible state* q, the new state $q' = \Phi_{i-1}(q)$ in which M would be when coming back[2] to C_i if the current move was a transition to the left ($C_i \to_q C_{i-1} \rightsquigarrow_{q'} C_i$). The first thing $\mathcal{A}$ does when entering C_i is to build (given w_i,

[2]If the accepting or rejecting state is reached by M before the whole input has been read (that is, before reaching the cell C_n containing w_n, $\mathcal{A}$ just enters a special state and keeps moving rightwards until it reaches C_n; in other terms, it just *delays* the accepting or rejecting state.

Φ_{i-1} and δ) the current lookup table Φ_i by going over all possible states q C_i could be entered from the right, and computing the state q' in which it would move back to the right in this case. Note that, although computationally expensive, this process is ensured to end in a finite amount of time, as (a) there are finitely many states to consider; and (b) M cannot leave C_i for C_{i-1} twice in the same state, or it would enter a loop and not terminate.
Once Φ_i has been built and "saved" in memory, $\mathcal{A}$ simulates the transition M would have followed on C_i; if the result is to move to the right (to C_{i+1}, $\mathcal{A}$ does so and erases Φ_{i-1} from its memory – thus ensuring only a constant amount of memory is used, namely roughly at most two lookup tables of size $|Q|^{|Q|}$ at a time. However, if M's head was to move to the left (to C_{i-1}), $\mathcal{A}$ does not perform the move, but instead uses Φ_{i-1} to shortcut it and directly enter the new state M would eventually be into when coming back to C_i.

As this whole process is well defined, $\mathcal{A}$ will eventually reach the end of the input cell (the cell C_{n+1} on which is written the blank symbol $\sqcup$. At that point, there are still two cases to consider:

(a) M (and therefore $\mathcal{A}$) is in a final state, in which case the algorithm accepts or rejects;

(b) or it has not reached such a state yet; now, it can be seen as a two-way, read-and-write algorithm entering a blank tape (as it still cannot access nor write on the input part of the tape, which it left behind). Yet, since its behavior is entirely determined by δ (which is fixed) and $\Phi_n \colon Q \to Q$ (which has been previously computed and is the only thing $\mathcal{A}$ kept in memory at that point), *there are finitely many possible configurations for the "input" of that step, and thus finitely many possible executions.* Since we have the guarantee that eventually M will either accept or reject, the whole behavior of $\mathcal{A}$ after reaching the blank symbol can be hardcoded into a new lookup table of size $2^{|\Phi_n|} = 2^{|Q|^{|Q|}}$, which is still constant memory.

□

Problem 5

All 4 problems are undecidable – to see why, we shall describe reductions from the halting problem (or previously proved undecidable problems) to them.

(a) *Given a TM M, does it halt on the empty string ε?*
Suppose we have a TM D deciding this problem. On input an instance of the halting problem $(\langle M \rangle, w)$, one can generate a TM M'_w in which w is hardcoded, and does the following: it writes W on the tape contents, then simulates M on w. As M'_w halts on ε iff M halts on wW, D can be used as subroutine to solve the halting problem – hence a contradiction.

(b) *Given a TM M, is there an input on which M halts?*
We reduce this problem to the problem of deciding if a TM halts on ε (the previous one, (a)). For every input $\langle M \rangle$, one can build a TM M' which ignores its input (e.g, by erasing the tape contents) and then simulates M on ε. Then, $\exists w$ s.t M' halts on w iff M halts on ε, and any TM deciding (b) could be used to decide (a).

(c) *Given a TM M, is the set $L(M)$ of strings that M accepts (language that M recognizes) finite?*
We again reduce this problem to the problem of deciding (a). For every input $\langle M \rangle$, one can build a TM M' which ignores its input (e.g, by erasing the tape contents) and then simulates M on ε, and accepts if M halts (in particular, M' always either accepts or runs forever). Then,

$$(M \text{ halts on } \varepsilon) \Rightarrow |L(M')| = \infty \qquad \text{and} \qquad (M \text{ does not halt on } \varepsilon) \Rightarrow |L(M')| = 0$$

and any TM deciding whether the set of accepted strings is finite could be used to decide (a).

(d) *Given two TM M, M', is $L(M) = L(M')$?*
Here is a reduction to (a): given the code $\langle M \rangle$ of a TM M, one can generate two TM's M', M'' such that:
- on input $w \in \Sigma^*$, M' accepts if $w = \varepsilon$ and rejects otherwise;
- on input $w \in \Sigma^*$, M'' rejects if $w \neq \varepsilon$; otherwise, it simulates M on $w = \varepsilon$, and accepts when (if) M halts.

It is easy to see that $L(M') = L(M'')$ if, and only if, M halts on ε. Hence being able to decide the former (for general M', M'') implies that the latter is decidable.

Problem 6

$\boxed{\Rightarrow}$ Suppose L is recursive (decidable). Then, there exists a TM M deciding it; one can use it to build N which enumerates L in length-increasing fashion as follows:

- N, on input ε, writes a special symbol (not from the tape alphabet of M) $\odot$ on the tape;
- it then goes over all possible strings $w \in \Sigma^*$ in length-increasing fashion, and, for each w, simulates M on w (always keeping $\odot$ at the end of the used portion of the tape); when the simulation of M halts on w (it halts for any input by definition), N writes $\sqcup w \sqcup$ after $\odot$, so that w is output;
- N removes all characters (if there are) occurring after $\odot$, from right to left, and goes to the next string w to enumerate.

Each w accepted by M (i.e, all $w \in L$) will sooner or later be written as $\sqcup w \sqcup$ ate the end of N's tape (that is, will be outputted); and this will not happen for any $w \notin L$, because of the use of $\odot$. Since the inputs are considered by non-decreasing length, N enumerates L in length-increasing fashion.

$\boxed{\Leftarrow}$ Suppose there is a TM N enumerating $L \subset \Sigma^*$ in length-increasing fashion (as mentioned in the exercise, one can wlog assume L is infinite[3]). One can build a TM M deciding L as follows: on input $w \in \Sigma^*$, M will keep w somewhere in memory, as well as $|w|$, and also will keep track of the number of $\sqcup$ lie after the current position of its head (more precisely, not the actual number, but only if there is *exactly* one or not).

M then just simulates N, and, every time the head is on a cell containing $\sqcup$ and moves left, *and* its "counter" is equal to 1, M will "pause" whatever it is doing (entering a new "comparison mode", independent of the rest of its execution – this can be implemented using finitely more states) and check whether w is equal to the string delimited by these two rightmost $\sqcup$ symbols: if it is the case, M accepts. If it is not *and* the number of characters between the two symbols is strictly greater than $|w|$, M rejects.

By definition of N, if $w \in L$, then the tape contents will at some point end up with $\sqcup w \sqcup$, and this will happen before any pattern $\sqcup v \sqcup$ with $|v| > |w|$ appears at the end of the tape. Hence, M will accept. However, if $w \notin L$, the pattern $\sqcup w \sqcup$ will never appear at the end of the tape; but since L is infinite, it contains a string v with $|v| > |w|$, and the pattern $\sqcup v \sqcup$ will eventually be written at the end of the tape: when this happens, M will reject. Hence, M is a TM that always halts, and accepts only words belonging to L: M decides L. $\square$

[3]If not, the trivial TM that just has $L = \{w^{(1)}, \ldots, w^{(N)}\}$ built-in in its description and checks if its input is one of the $w^{(i)}$'s decides L, and there is nothing left to prove.