In the Name of God

Theory of Computer Science - HW3

Ali Fathi - 99204943 Instructor: Dr. Ebrahimi

---

1 RO Machine

1.1 Proof of Regular Language

Suppose M_{RO} is our read-only Turing machine which recognizes the language $L = L(M_{RO})$. For each word w, we introduce a function T_w (or could be called a vector) as follow (Q is the states of M_{RO}):

$T_w : Q \cup \{\bullet\} \to Q \cup \{A, NA\}$
$T_w(\bullet) =$ the state after first crossing w, starting with q_{start}
$T_w(q) =$ the state after crossing w, starting in q at the right end of w

$T_w(\bullet)$ says that when you start the machine to process w, whether it's header crosses the right end of w or not. If so, $T_w(\bullet) = \bar{q}$ in which $\bar{q}$ is the state of machine after this crossing (when the header is on the next cell of w). If it doesn't cross the the input w, if it accepts the input, $T_w(\bullet)$ would take value A and of it doesn't accept it (i.e. it rejects it or loops on that), $T_w(\bullet)$ would take value NA.
$T_w(q)$ has the same meaning, but when the machine starts at the right end of w and when is in state q. Note that in the way we have described T_w, it isn't important at all that what is written in the right of unchangeable w.
Consider an equivalence relation on Σ^* as follow:

$$w \sim_T v \quad \Leftrightarrow \quad T_w(.) \equiv T_v(.)$$

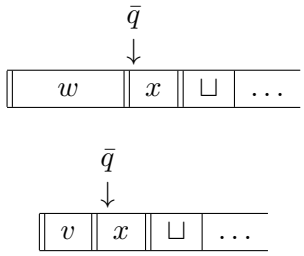
Which says T_w and T_v are same functions on $Q \cup \{\bullet\}$. It's simple to check it is an equivalence relation:

1. $T_w(.) \equiv T_w(.) \Rightarrow w \sim_T w$
2. $w \sim_T v \Rightarrow T_w(.) \equiv T_v(.) \Rightarrow T_v(.) \equiv T_w(.) \Rightarrow v \sim_T w$

3. $w \sim_T v$ and $v \sim_T u \Rightarrow T_w(.) \equiv T_v(.) \equiv T_u(.) \Rightarrow w \sim_T u$

We show that $\sim_T$ is a finer relation than Myhill-Nerode relation $\sim_L$ (i.e. $w \sim_T v$ implies that $w \sim_L v$). To do so, we have to show if $T_w(.) \equiv T_v(.)$, then for all $x \in \Sigma^*$(containing null), vx and wx have the same acceptance destination on M_{RO} (which means both of them are in $L = L(M_{RO})$ or not).

We make induction on the number of turns which header crosses w and v. In the first turn, we have $T_w(\bullet) = T_v(\bullet)$. If both of them are A, it means M_{RO} accepts both wx and vx without any need to visit x. Also if $T_w(\bullet) = T_v(\bullet) = NA$, both of them are not accepted (are rejected or looped) so wx and vx are not in L. Otherwise if $T_w(\bullet) = T_v(\bullet) = \bar{q}$, says header would be in state $\bar{q}$ after first cross (when is above first letter of x):



In this situation, all movements of M_{RO} must be the same in both cases unless header returns to left of x. If header never returns back in one case (and accepts, rejects or loops on right), it must do the same on the other case too because both of them are like the machine which starts with state $\bar{q}$ and input x. So both of wx and vx are in L or not. By the same reason, if header returns back to left side, it should be in a same state like q in both cases and exactly same string must be on the right of tape, such as xy (M_{RO} is allowed to write something after input. So something same like y could be written right of x).

Now we use induction. Suppose after t turns which M_{RO} crosses the w and v during processing wx and vx, header has returned back to the right end of w and v, has the same state q and same thing is written on the right of the tape, like xy. As $T_w(q) = T_v(q)$, in both cases M_{RO} would accept the input (when $T_w(q) = T_v(q) = A$), or doesn't accept it (when $T_w(q) = T_v(q) = NA$), or crosses to the right side by being in state $\bar{q}$ (when $T_w(q) = T_v(q) = \bar{q}$).

Again like what we argued above, in both cases it is like M_{RO} is stating in state $\bar{q}$ running on input xy. So, it would go to q_{accept} in both cases, or doesn't accept the input (rejects or loops), or the header returns back to the left side in the same state q' and the same letters written on the right of the

tape such as xy'.
Our induction is complete here because we proved that if M_{RO} halts in current round t, the result would be the same in both cases wx and vx, and if there is another round $t+1$, in both cases the states and the content of the tape right of x would be the same in the beginning of next round (q' and xy'). Therefore, the number of rounds $t \in \mathbb{N} \cup \{\infty\}$ must be the same running on wx and vx, and also the accepting result.
We proved that $w \sim_T v$ leads to $w \sim_L v$. It means $\sim_T$ is a finer relation of $\sim_L$. Therefore the number of equivalence classes under $\sim_L$ are fewer (or equal) than the number of equivalence classes under $\sim_T$. We have:

$$(\Sigma^* / \sim_T) \leq (range_T)^{dom_T} = |Q \cup \{A, NA\}|^{|Q \cup \{\bullet\}|} = (|Q|+2)^{|Q|+1}$$

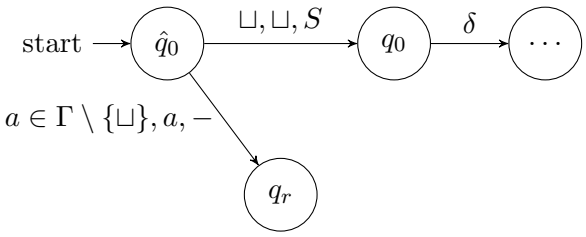
And it's a finite number. Therefore $|\Sigma^* / \sim_L| \leq |\Sigma^* / \sim_T| < \infty$, and according to Myhill-Nerode theorem for regular languages (a language is regular iff it's equivalence classes is finite), L is a regular language. ■

1.2 Algorithm is Undecidable

Consider f to be a Turing-computable algorithm which gets a RO Turing machine description like $\langle M_{RO} \rangle$, and returns an equivalent DFA like $\langle M_{DFA} \rangle = f(\langle M_{RO} \rangle)$. Using f, we can make a decider D for $L_\epsilon = \{\langle M \rangle : M \text{ is a Turing machine and } \epsilon \in L(M)\}$. To do that, first we describe a Turing-computable function g which gets a Turing machine description $\langle M \rangle$ and returns the description of a RO machine $\langle M_{RO} \rangle = g(\langle M \rangle)$ as follow:

$$\begin{aligned} \langle M \rangle &= \langle Q, \Sigma, \Gamma, \delta, q_0, q_a, q_r \rangle \\ \langle M_{RO} \rangle &= g(\langle M \rangle) \\ \langle M_{RO} \rangle &= \langle Q \cup \{\hat{q}_0\}, \Sigma, \Gamma, \hat{\delta}, \hat{q}_0, q_a, q_r \rangle \end{aligned}$$

M_{RO} starts in the state $\hat{q}_0$. In this state, if header is above something not blank, M_{RO} rejects the input (i.e. directly goes to q_r). Else (when it is above blank), it goes to q_0, header stands at it's position (which could be implemented by a redundant movement to right then left in standard machine) and the machine behave exactly like M after:



Obviously M_{RO} does not change any input letter written on tape (if any thing is written, we immediately reject it) so it is really a read-only Turing machine. Also M_{RO} accepts ϵ if and only if M accepts ϵ because M_{RO} imitates exactly the M on input ϵ. Clearly g is a computable function as we have described it algorithmically.
We make the decider D for L_ϵ as follow:
" D on input parsed as $\langle M \rangle$:

- Run G on $\langle M \rangle$ and get $\langle M_{RO} \rangle = g(\langle M \rangle)$
- Run F on $\langle M_{RO} \rangle$ and get $\langle M_{DFA} \rangle = g(\langle M_{RO} \rangle)$
- For $\langle M_{DFA} \rangle = \langle Q, \Sigma, \delta, q_0, \mathcal{F} \rangle$, check whether $q_0 \in \mathcal{F}$ or not. If it was, accept the input $\langle M \rangle$ else reject it. "

According to the way we described g, $\epsilon \in M$ iff $\epsilon \in M$ and according to the equivalence of M_{RO} and M_{DFA}, $\epsilon \in M_{RO}$ iff $\epsilon \in M_{DFA}$. And checking whether $\epsilon \in M_{DFA}$ or not, is as simple as checking whether q_0 is accepting state or not. Therefore D is a decider for L_ϵ and as there shouldn't be such decider, function f must be an uncomputable function. ■

---

2 Reducible to C

For two arbitrary languages A and B on $\Sigma = \{0, 1\}$ (the argument for other alphabets is the same), we introduce C as $C = A0 \cup B1$, in which $A0$ is concatenation of all the strings in A with 0, and $B1$ is concatenation of all the strings in B with 1. If we partition Σ^* as $\Sigma^* = \{\epsilon\} \cup \Sigma^*0 \cup \Sigma^*1$, it would be something like this:

	Σ^*	
C	$A0$	$B1$
$\bar{C}$	$\bar{A}0$	$\bar{B}1$
	ϵ	

As two further partitioning $\Sigma^*0 = A0 \cup \bar{A}0$ and $\Sigma^*1 = B1 \cup \bar{B}1$ has been made.
Defining two mappings $f_1, f_2 : \Sigma^* \to \Sigma^*$:

$$f_1(w) \coloneqq w0$$
$$f_2(w) \coloneqq w1$$

we would obviously have:

$$w \in A \Leftrightarrow f_1(w) = w0 \in A0 \subseteq C$$
$$w \in B \Leftrightarrow f_2(w) = w1 \in B1 \subseteq C$$

and as f_1 and f_2 are just a simple 1-letter concatenation, they are clearly Turing computable. Therefore $A \leq_m C$ and $B \leq_m C$. ■

---

3 Sum is Primitive Recursive

Having primitive recursive function $g : \mathbb{N}^2 \to \mathbb{N}$, we are going to prove the following f is recursive too:

$$f : \mathbb{N} \to \mathbb{N}$$
$$f(x) = \sum_{i=0}^{x} g(x,i) = g(x,0) + g(x,1) + \ldots + g(x,x)$$

First, we prove the following function h is primitive recursive:

$$h : \mathbb{N}^2 \to \mathbb{N}$$
$$h(x,y) = \sum_{i=0}^{y} g(x,i) = g(x,0) + g(x,1) + \ldots + g(x,y)$$

And this is done by using recursion operator on second argument:

$$
\begin{aligned}
&h_1 : \mathbb{N} \to \mathbb{N} \\
&h_2 : \mathbb{N}^3 \to \mathbb{N} \\
&h(x,0) = h_1(x) = g(x,0) = [go(\pi_1^1, z \circ \pi_1^1)](x) \\
&h(x,y+1) = h_2(x,y,h(x,y)) = h(x,y) + g(x,y+1) = \\
&\qquad\qquad = add(g(x,y+1), h(x,y)) \\
&\qquad\qquad = [add \circ (\phi, \pi_3^3)](x,y,h(x,y)) \\
&\phi(x,y,z) \coloneqq g(x,y+1) = [g \circ (\pi_3^1, s \circ \pi_3^2)](x,y,z)
\end{aligned}
$$

In which π_k^i, is $k-adic$ projection on i'th element (i.e. $\pi_k^i(x_1,\ldots,x_k) = x_i$), s is successive function and z is zero function.
We proved that h is a primitive recursive function. Now we construct f using h and composition operator as follow:

$$f(x) = h(x,x) = [h \circ (\pi_1^1, \pi_1^1)](x)$$

And therefore f is a primitive recursive function too. ■

---

4 Primitive Recursive Proof

First we recall some primitive recursive functions:

- $z(x) = 0$: zero function
- $s(x) = x + 1$: successive function
- π_k^i: $k - adic$ projection on i'th element (i.e. $\pi_k^i(x_1,\ldots,x_k) = x_i$)
- $add(x,y) = x + y$: addition function
- $x \dot{-} y = max\{x - y, 0\}$: multiplication function
- $m(x,y) = x \times y$: multiplication function
- $\leq (x,y)$: comparison function
- $= (x,y)$: equality function

And we prove some lemmas:

Lemma. *Constant function is primitive recursive:*

$$k : \mathbb{N}^n \to \mathbb{N}$$
$$k(x_1, \cdots, x_n) = k$$

Proof. We use induction. First we show zero constant is primitive recursive:

$$0(x_1, \cdots, x_n) = [z \circ \pi_n^1](x_1, \cdots, x_n)$$

And if we suppose that k is primitive recursive, also $k+1$ would be primitive recursive:

$$(k+1)(x_1, \cdots, x_n) = [s \circ k](x_1, \cdots, x_n)$$

So the assertion is proved by induction. □

Lemma. *Scaling a specific element is primitive recursive:*

$$k^i : \mathbb{N}^n \to \mathbb{N}$$
$$k^i(x_1, \cdots, x_n) = kx_i$$

Proof. Simply by composition and previous lemma

$$k^i(x_1, \cdots, x_n) = [k \circ \pi_n^i](x_1, \cdots, x_n)$$

□

Lemma. *Delayed function is primitive recursive:*

$$d : \mathbb{N}^3 \to \mathbb{N}$$
$$d(x, y, z) =: \begin{cases} 0 & y < x \\ z & y \geq x \end{cases}$$

Proof. Would be proved by composition:

$$d(x, y, z) =: \leq (x, y) \times z = [m \circ (\leq \circ(\pi_3^1, \pi_3^2), \pi_3^3)](x, y, z)$$

□

Now, we use these lemmas to solve the questions.

a) $f(x, y) = 2x + 3y$
By using $k^i : \mathbb{N}^2 \to \mathbb{N}$ function and composition:

$$f(x, y) = [add \circ (2^1, 3^2)](x, y)$$

b) $f(x,y) = |x - y|$
By using delayed function and composition:

$$\begin{aligned} f(x,y) &= \leq (x,y) \times (y \dot{-} x) + \leq (y,x) \times (x \dot{-} y) \\ &= d(x, y, y \dot{-} x) + d(y, x, x \dot{-} y) \\ &= [add \circ (d_1, d_2)](x,y) \\ d_1(x,y) &\coloneqq [d \circ (\pi_2^1, \pi_2^2, \dot{-}_{inv})](x,y) \\ d_2(x,y) &\coloneqq [d \circ (\pi_2^2, \pi_2^1, \dot{-})](x,y) \\ \dot{-}_{inv}(x,y) &\coloneqq y \dot{-} x = [\dot{-} \circ (\pi_2^2, \pi_2^1)](x,y) \end{aligned}$$

And as we have used only composition on primitive recursive functions, f also would be primitive recursive.

c) $f(x,y) = max(x,y)$
Again by using delayed function and composition:

$$\begin{aligned} f(x,y) &= [\leq (x,y) \times y] + [\leq (y,x) \times x] - [= (x,y) \times x] \\ &= d(x,y,y) + d(y,x,x) - [= (x,y) \times x] \\ &= [add \circ (d_1, d_2)](x,y) \dot{-} [m \circ (=, \pi_2^1)](x,y) \\ d_1(x,y) &\coloneqq [d \circ (\pi_2^1, \pi_2^2, \pi_2^2)](x,y) \\ d_2(x,y) &\coloneqq [d \circ (\pi_2^2, \pi_2^1, \pi_2^1)](x,y) \end{aligned}$$

---

5 Rice Theorem

Recall Rice theorem:

Rice Theorem. *Consider P as a non-trivial semantic property and $L_P = \{\langle M \rangle \,|\, M \in TMs\,,\, M$ possesses the property $P\}$. Then L_P is an undecidable language.*

We prove it by contradiction. Suppose that there is a decider D for L_P. As P is a non-trivial property, there exist machines $\langle M_1 \rangle \in L_P$ and $\langle M_2 \rangle \notin L_P$ .Then we make the Turing machine T as follow:

" T on input w:

- Obtain, via the recursion theorem, own description $\langle T \rangle$

- Simulate D on $\langle T \rangle$
- If D accepted $\langle T \rangle$, Simulate $\langle M_2 \rangle$ on w and go to accept or reject state based on it
- If D rejected $\langle T \rangle$, Simulate $\langle M_1 \rangle$ on w and go to accept or reject state based on it "

If D accepts $\langle T \rangle$ (it means $\langle T \rangle \in L_P$), then we have $L(T) = L(M_2)$ and as P is a semantic property, we should have $T \notin L_P$ which is contradiction. Also if D rejects $\langle T \rangle$ (it means $\langle T \rangle \notin L_P$), then we have $L(T) = L(M_1)$ therefore $T \in L_P$ which is again a contradiction. So it is impossible to have a such decider D. ■
