CS 381 – HW7 Solutions

1. For each of the following languages, find out if it is recursive or not. If it is, describe a total Turing machine that computes it. If not, explain why the existense of such a machine entails a contradiction.

(a) $\{0^{n_0}1^{n_1}0^{n_2}...1^{n_{2k-1}}0^{n_{2k}}$: for all $0 \leq i \leq 2k, i \in N$ and n_0 is a solution to the equation $n_1x^{n_2} + n_3x^{n_4} + ... + n_{2k-1}x^{n_{2k}} = 0\}$

Solution:

n_0 must be a solution to the equation $n_1x^{n_2} + n_3x^{n_4} + ... + n_{2k-1}x^{n_{2k}} = 0$. Let $n_0 = 0$, a valid solution, so that the problem at hand is to determine whether or not the set of strings of the form $\epsilon 1^{n_1}0^{n_2}...1^{n_{2k-1}}0^{n_{2k}}$ for all $0 \leq i \leq 2k, i \in N$ satisfying $n_10^{n_2} + n_30^{n_4} + ... + n_{2k-1}0^{n_{2k}} = 0$ is recursive. Since all coefficients $n_1, n_3, ..., n_{2k-1}$ must be non-negative, we know that $n_2, n_4, ..., n_{2k} \neq 0$ (because $0^0 = 1$, and then 0 would not be a solution to the equation). It is not too difficult to see that this set of strings is just $(0 + 1)^*0$ (all strings from $\{0, 1\}^*$ that end in 0).

To construct a Total Turing Machine for this set is easy. Scan the input tape to verify the characters belong to the alphabet {0,1}, and proceed to the last character on the tape. If it is a 0 accept – else reject.

Comments:

Numerous typos existed in the problem definition, which left the question open to interpretation. Because of this, we tried to grade fairly by accepting any and all interpreations. The above solution is my interpretation of what was being asked.

1b. $\{\#M \mid \varepsilon \in L(M)\}$

Solution 1b:

This language (let's call it L) is not recursive. We prove this by showing $HP \leq_m L$, where HP is the halting problem, $HP = \{(\#M, x) \mid M$ halts on input $x\}$, which we know to not be recursive.

To do this, we must give a map $\sigma : \{(\#M, x) \mid$ M is a TM and $x \in \Sigma^*\} \rightarrow \{\#M \mid M$is a $TM\}$ such that M halts on input x iff $L(\sigma(\#M, x))$ contains ε.

So, given machine M and string $x \in \Sigma^*$, we construct $M' = \sigma(\#M, x)$ as follows: on input y, M' erases its tape and writes x, then simulates M on input x; M' accepts iff M halts on x.

Suppose M halts on input x. Then on any input y, M' accepts, and in particular M' accepts input ε. Suppose M does not halt on input x. Then on any input y, M' does not halt and therefore does not accept, so $L(M')$ is empty, and in particular M' does not accept ε.

Remark: The proof is also on Page237 of the textbook.

1c. $\{\#M \mid |L(M)| < 100\}$

Solution 1c:

Assume the existence of a total turing machine T that computes L.

We solve the halting problem using T as follows:

On input machine M and word x, we create a turing machine M' that ignores its input and simulates M on x. If M halts on x, then M' accepts its input no matter what it was. Otherwise, M' loops forever on all inputs. Its easy to see that $L(M') = \Sigma^*$ if M halts on x and $L(M) = \phi$ if M fails to halt on x. Now feed M' as input to total turing machine T. If T accepts M' then M' must have fewer than 100 words in its language, and thus must be phi, and thus M must have failed to halt on x. If T rejects M' then M' has more than 100 words in its language, and thus must be Σ^*, and thus M must have halted on x. Because T is assumed to be total, we are guaranteed to halt, and so in finite time we have determined whether or not M halted on x. But this is impossible because the halting problem isn't recursive, and therefore our assumption of total T was wrong ... and the original language must not be recursive.

1d. $\{(\#M_1, \#M_2) \mid L(M_1) = L(M_2)\}$

Solution 1d:

Assume the existance of a total turing machine T that computes L.

We solve the halting problem using T as follows:

On input machine M and word x, we create a turing machine M' that saves its input y and simulates M on x. If M halts on x, then M' accepts only if $y = 0$. Otherwise, M' loops forever on all inputs. Similarly, we make M'' the same way, except of M halts on x, then M'' accepts only if $y = 1$. Its easy to see that $L(M') = \{0\}$ and $L(M'') = \{1\}$ if M halts on x and $L(M') = L(M'') = \phi$ if M fails to halt on x. Now feed M' and M'' as input to total turing machine T. If T accepts then M' and M'' must have the same language, which must be ϕ, and so M must have failed to halt on x. If T rejects then M' and M'' must have difference languages $\{0\}$ and $\{1\}$ respectively, and thus M must have halted on x. Because T is assumed to be total, we are guaranteed to halt, and so in finite time we have determined whether or not M halted on x. But this is impossible because the halting problem isn't recursive, and therefore our assumption of total T was wrong ... and the original language must not be recursive.

2a. Prove that if one changes the definition of Turing Machine to allow an infinite set of states Q, then for every $L \subseteq \{0\}^*$ there exists a total machine that computes it.

Solution 2a:

The idea is to have a state for each element of $\{0\}^*$, and a transition from state to an accept state iff the element is in L, and to a reject state iff the element is not in L.

Formally, given language $L \subseteq \{0\}^*$, define $T_L = (Q, \{0\}, \{0\}, \delta, q_0, t, r)$, where

- $Q = \{q_i \mid i \in \mathbb{N}\} \cup \{t, r\}$
- $\delta(q_i, 0) = (q_{i+1}, 0, R)$ for all $i \in \mathbb{N}$
- $\delta(q_i, blank) = \begin{cases} (t, 0, R) & \text{if } 0^i \in \text{Ł} \\ (r, 0, R) & \text{if } 0^i \notin L \end{cases}$ for all $i \in \mathbb{N}$

2b. Prove that there exists a language $L \subseteq \{0\}^*$ such that for every Turing machine T (under the usual definition) $L(T) \neq L$.

Solution 2b:

Since the number of languages $L \subseteq \{0\}^*$ is uncountable, it suffices to prove that the number of distinct Turing machines with input alphabet $\{0\}$ is countable. To do this, we give finite sets S_n of Turing machines such that for any TM T with input alphabet $\{0\}$, $\bigcup_{n\in\mathbb{N}} S_n$ contains a TM with behavior identical to that of T.

Define $S_n = \{$ Turing machines with input alphabet $\{0\}$, $Q \subseteq \{q_0, ...q_n\}$, and $\Gamma \subseteq \{a_1, ...a_n\}\}$. It is clear that each TM with input alphabet $\{0\}$ is equivalent to one in S_n for some n. Furthermore, since a TM in S_n is determined by its δ function, $\delta : Q \times \Gamma \to Q \times \Gamma \times \{R, L\}$, it follows that $|S_n| \leq (2n^2)^{n^2}$, so S_n is indeed finite for each n. It follows that there are only countably many distinct Turing machines with input alphabet $\{0\}$.

Note:
1.You can also use diagonalization to prove this. The proof is similar to that on Page233 of the textbook.
2.You can use a counter example to prove this, like HP^C, but you should mention how to encode this problem into language in $\{0\}^*$.
3.A Turing machine with only finite number of states CAN compute infinite language. Don't use an infinite language as a counter example to prove this.

3. Prove that the family of recursive languages is closed under the * operation. Namely, if L is recursive then so is $L^* = \{w_1...w_n : n \in N \text{ and for all } i, w_i \in L\}$.

Solution:

Assume that L is a recursive language. Then we know there exists a Total Turing Machine M_L s.t. $L(M_L) = L$. If we can create a Total Turing Machine using M_L, to accept L^* then we will have succeded in proving that recursive languages are closed under asterate.

Let's define a new Turing Machine, call it M' which has M_L encoded in it's state logic. Given any word x, M' will procceed to determine whether or not

$x \in L^*$. If $x = \epsilon$ then M' accepts. Otherwise, M' simulates M_L on x. If M_L accepts then M' also accepts since clearly if $w \in L$, $w \in L^*$. If $|x| = 1$ and M_L rejects, M' rejects (if a single character doesn't exist in L, the * operation will not introduce it in L^*).

Assuming $|x| \geq 2$, let $n = |x|$. M' partitions x into 2 segments, such that $x = x_1x_2$. There is a finite number of ways to partition any string into 2 segments. M' tries each 2-partition in a systematic order. For each, it simulates M_L on x_1 and then again on x_2. If a 2-partition exists such that both the particular x_1 and x_2 are accepted by M_L, then M' accepts (if $x_1, x_2 \in L$ then $x_1x_2 \in L^*$). This process repeats for all possible partitions of x into 3, 4,...n substrings – a finite amount of work. If at no point, M_L can accept each resulting $x_1, x_2, ...x_k$ $k \leq n$, then M' rejects (if $\nexists k$ s.t. $x = x_1x_2...x_k$ where $x_1, x_2..., x_k \in L$, then $x \notin L^*$).

Clearly, $L(M') = L^*$. Furthermore, since M_L is guaranteed to always halt, and there is only a finite amount of work required to determine whethere or not a given $x \in L^*$, M' will always halt. Therefore M' a Total Turing Machine.

Comments:

The key realization to this problem is that for a given x, it takes a finite amount of work to try all possible substrings of x such that their concatenations yields x. People that resorted to using non-determinism should have developed a deterministic approach since non-deterministic TMs are not covered in the course.

The most common error, entailed merely modifying M_L to have it's start state be an accept state, and adding a transition from its accept state to its start state. With a little thought, one can see how the resulting TM is a flawed greedy algorithm to determine whether $x \in L^*$. Assume $a, ab \in L$. $aab \in L^*$ and yet the proposed TM would verify that the first $a \in L$, the second $a \in L$ but reject because $b \notin L$.

4a. For every T, $L(\overline{T}) = \overline{L(T)}$.

Solution 4a:

False. Words that cause T to loop also cause $\overline{T}$ to loop, and are thus are not in $L(\overline{T})$ while they are in $\overline{L(T)}$. Machines that loop exist, and therefore this fact provides a counterexample.

4b. For every T_1 and T_2, if $L(T_1) = L(T_2)$ then $L(\overline{T_1}) = L(\overline{T_2})$.

Solution 4b:

False. We can choose make T_1 to loop on some word $w \notin L(T_1) = L(T_2)$ but have T_2 immediately reject w. Thus $w \notin L(\overline{T_1})$ but $w \in L(\overline{T_2})$.

4c. For every machine T, if $L(T)$ is recursive then so is $L(\overline{T})$.

Solution 4c:

False, by counter-example. Consider the machine T that takes as input a machine M and a word x. T then runs M on x. If M halts on x, T rejects. Its fairly clear that $L(T) = \phi$, and so even though T is not total, $L(T)$ is recursive because some other recursive machine accepts $L(T)$. If we look at $L(\overline{T})$, however, $\overline{T}$ runs M on x and accepts if M halts on x. $L(\overline{T})$ is the language of machine and word pairs such that the machine halts on the word – the halting problem! We know the halting problem isn't recursive, and so this provides us a counter-example.

Note: it is not enough to say that because T may not be total, $\overline{T}$ may not be total and therefore $L(\overline{T}$ isn't recursive. We have to show an example in which T is not total, $\overline{T}$ is not total, and also no total machine T' computes the same language $L(\overline{T})$. The above example is such a language.