CS–172 Computability & Complexity, Spring 2021

Homework 7 Solutions

Note: These solutions are not necessarily model answers. Rather, they are designed to be tutorial in nature, and sometimes contain more explanation than is required for full points. Also, bear in mind that there may be more than one correct solution. The maximum total number of points available is 37.

1. (a) **Recognizable but not decidable**. We show first that this language L_{TWICE} is recognizable: to recognize it, we just define a TM that, on input $\langle M, q\rangle$, simulates M on all possible inputs, accepting if M ever reaches state q twice in a single simulation. To simulate M in such a fashion, we use a dovetailing approach in the same way a recursive enumerator might: simulate it for one step on the lexically first string, then for two steps on each of the first two strings, and so on. *5pts*

We now show that L_{TWICE} is undecidable, by giving a mapping reduction from A_{TM}. The reducing function f on input $\langle M, w\rangle$ outputs a machine/state description $\langle M', q'\rangle$ as follows. On any input, M' first erases its input and replaces it with the fixed string w. Then it behaves like M, except that all transitions to the accept state of M instead go to a new state q' (which is not part of the state set of M). When in the state q', M' remains in q' and moves right regardless of the symbol it is reading. Clearly this construction can easily be carried out by a TM, so f is Turing computable.

Now we can check that the description $\langle M', q'\rangle$ output by f has the following properties:

- M accepts $w \Rightarrow M'$ enters state q' twice on some input. To check this, note that if M accepts w then M' will transition to state q' and remain in it forever (thus visiting it more than twice!).
- M' enters state q' twice on some input $\Rightarrow M$ accepts w. To check this, note that, by design, the only way M' can enter state q' at all is if M accepts w.

Hence f is a mapping reduction from A_{TM} to L_{TWICE}, and therefore L_{TWICE} is undecidable.

[*A serious error in this part (and also in part (c)) was to simply claim something like "there is no way to know whether a given machine halts on a given input or not", or that some particular method (such as dovetailing) does not work: you really have to do a reduction from a known undecidable language in order to rigorously prove that a new language is undecidable. Another serious problem was doing the reduction the wrong way round (i.e., from the new problem to the old one). A more minor error was not correctly or explicitly handling the new state in the reduction.*]

(b) **Decidable**. If M does *not* enter any state twice on a given input, then clearly its computation can run for at most $q-1$ steps, where q is the number of states of M. Thus we can decide this question for a given input w by running M on w for (at most) q steps and keeping track of which states M has entered. But we can extend this idea to *all* inputs by noting that we need only consider inputs of length at most q (since this is the maximum number of input symbols that can be scanned within q steps). So we can decide the problem by simulating M for at most q steps on every input of length at most q. *4pts*

[*Some students only argued that the TM would stop in a finite amount of time on a given input, but did not argue that the TM only needs to consider finitely many inputs.*]

(c) **Not recognizable**. To show this, we give a mapping reduction from $\overline{A_{\text{TM}}}$ to this language, which we call $L_{\text{ACCEPTS=}}$. Since we know that $\overline{A_{\text{TM}}}$ is not recognizable, this will prove the same for $L_{\text{ACCEPTS=}}$. Our reduction will make use of a TM $M_{=}$ which decides the language of strings over $\{0, 1\}$ with the same number of 0s and 1s, which we denote $L_{=}$; of course we know how to *5pts* build such a machine (and indeed we gave it explicitly in class a few weeks ago). The reducing function f, on input $\langle M, w \rangle$, outputs $\langle M' \rangle$, where M' behaves as follows :

```
$M'$ = "On input $x$ :
    1. Run $M_=$ on $x$.
    2. If $M_=$ accepts, accept.
    3. If $M_=$ rejects, run $M$ on $w$.
    4.     If $M$ accepts, accept.
    5.     If $M$ rejects, reject."
```

M' as described above accepts Σ^* if M accepts w, and accepts $L_=$ otherwise. In other words, $\langle M' \rangle$ belongs to $L_{\text{ACCEPTS}=}$ if and only if $\langle M, w \rangle$ belongs to $\overline{A_{\text{TM}}}$. Moreover, f is clearly Turing computable, so we are done with the reduction.

NOTE: In this case, it turns out that $\overline{L_{\text{ACCEPTS}=}}$ is also not recognizable. To see this, you should be able to come up with a mapping reduction from A_{TM} to $L_{\text{ACCEPTS}=}$ very similar to that above. (Note that this is automatically also a reduction from $\overline{A_{\text{TM}}}$ to $\overline{L_{\text{ACCEPTS}=}}$.)

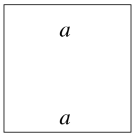
[*See comments above on part (a) for serious errors. In addition, in this part, some students incorrectly argued that the complement of the language is recognizable using the dovetailing technique. These students were confused about the fact that the language is* not *all strings with an equal number of 0s and 1s, but the set of all* Turing machines that accept *this language. The dovetailing technique is not valid since the description of a Turing a machine $\langle M \rangle$ cannot be accepted in finitely many steps as in the dovetailing argument.*]

(d) **Decidable**. We give a simple algorithm for deciding PCP_1. Since the alphabet is unary, we need only worry about the length of the string. Each domino i is of the form $[\frac{1^{u_i}}{1^{d_i}}]$, and we need to select a sequence of dominos such that $\sum u_i = \sum d_i$. Now note that if for every domino i, $u_i > d_i$, then there can be no match. Similarly, if $u_i < d_i$ for all i, there can be no match. Moreover, if there is some domino i such that $u_i = d_i$, then that domino itself gives us a match. Finally, note that if there are dominos i and j such that $u_i > d_i$ and $u_j < d_j$, then $(d_j - u_j)$ copies of domino i followed by $(u_i - d_i)$ copies of domino j forms an exact match. Putting all the above observations together, we see that an instance $\langle P \rangle$ of PCP_1 has a match if and only if there are dominos i and j such that $u_i \geq d_i$ and $u_j \leq d_j$. Since we can check this property easily (and we can also check if $\langle P \rangle$ is a valid instance of PCP with a unary alphabet), we can decide PCP_1. *4pts*

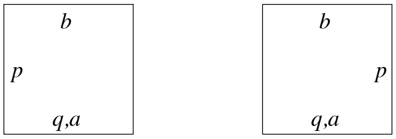
2. We wish to reduce from $\overline{\text{HALT}_{\text{TM}}}$ to PTP. Thus given input $\langle M, w \rangle$ to $\overline{\text{HALT}_{\text{TM}}}$ we wish to construct a set of tiles such that M does not halt on w if and only if we can tile the quadrant. Following the hint, the tile set will be chosen so that each row of the quadrant describes a configuration of M, with each tile representing a tape square. The first row will be forced to be the start configuration, and each succeeding row will represent the next configuration. The idea is that if M does not halt on w, then we can tile the entire quadrant, but if M halts, we have to stop after the halting configuration. This is the main idea; now all we have to do is implement it. *10pts*

We construct our tiles so that in a successful tiling, for every row except the first, the characters along the bottom of a row describe the previous configuration while the characters along the top describe the current configuration. Recall that in one step a Turing machine reads its current state and tape symbol and then writes out a new tape symbol, changes to a new state, and moves either left or right. The tape square being read by the tape head will be denoted (q_i, a) indicating that the state is q_i and the tape symbol is a, while a tape square not being read will contain only the tape symbol that it contains.

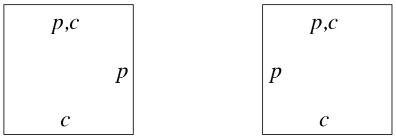
Thus for every $a \in \Gamma$ (including the blank symbol), we will have the *copying* tile below that simply copies a from the previous configuration to the current one.



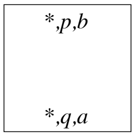
To handle transitions, for every transition $(q, a) \rightarrow (p, b, L)$ we have the *transition* tile below left, and for every transition $(q, a) \rightarrow (p, b, R)$ we have the transition tile below right.



To complement these, we also have the *receiving* tiles shown below which represent the new current tape squares. For each state p and character c, we have one tile as shown below left and another as shown below right.

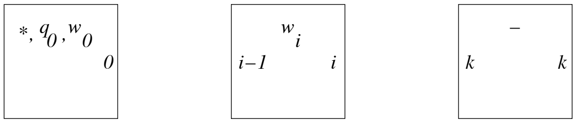


We need to be careful with the left end of the tape. We cannot allow the Turing machine to move left when at the left end of the tape, and we cannot allow the right receiving tile to appear at the left end. To enforce this, we create a new set of left-end tiles that can only be placed along the left edge of the quadrant. For every tile described so far except the left transition tile and the right receiving tile we make a new version that has an extra character $*$ on both top and bottom that indicates that it is meant only for the left edge. Note that if we make the corner tile a left-end tile, then every tile along the left edge of the quadrant will also be a left-end tile, as we need. To handle left transitions at the left end, for every transition $(q, a) \rightarrow (p, b, L)$ we make the tile shown below:



From these descriptions, you should be able to justify to yourself that given a row with a valid Turing machine configuration, the next row is forced to be the configuration of the Turing machine at the next step. Further, since there are no transitions out of the accepting and rejecting states, once the Turing machine halts we cannot add any more rows to the tiling. All that is left now is to design a set of tiles for the first row that forces the first row to encode the start configuration.

We know that the start configuration has the form $q_0 w$ where q_0 is the start state. Writing w as a string of symbols $w_0 \dots w_k$, we create a corner tile as shown below left and tiles for w_i, $1 \leq i \leq k$ below center, as well as a blank tile shown below right. We can see that this guarantees that the first row be the start configuration.



[*The most common error in this problem was forgetting to handle the special tiles along the left-hand edge.*]

3. (a) The set of Turing machines is countable, and hence the set of computable functions is also countable. On the other hand, we show that the set of functions from Σ^* to Σ^* is uncountable using a diagonalization argument. Assume for purposes of contradiction that there are only a countable number of functions; enumerate these as $f_0, f_1, f_2, \ldots$. Now consider the set of all strings Σ^* in lexicographic order $s_0, s_1, s_2, \ldots$. We can write out a table in which the i, j entry is $f_i(s_j)$. Now consider the function g such that $g(s_i)$ is simply $f_i(s_i)$ with an extra 1 at the end. Then $g \neq f_i$ for all i, since g differs from f_i on s_i. Thus we have a contradiction, so the set of functions is not countable, and thus there exist functions that are not computable. *3pts*

(b) Consider a computable function f. We show that the range of f is recognizable by designing a TM M that enumerates the range of f. (Recall from the previous HW that a language is recognizable by a Turing machine if and only if it is enumerable by a Turing machine.) This is easily done: for each string $w \in \Sigma^*$, in lexicographic order, M computes $f(w)$ (which is possible since f is computable: note that each of these computations must halt) and outputs it. *3pts*

(c) Take your favorite language L that is recognizable but not decidable (e.g., HALT$_{\text{TM}}$). Since L is recognizable, there is a Turing machine M that enumerates it. Define the function f as follows: on input w, with $|w| = n$, output the nth string enumerated by M. (Call the first string enumerated by M the 0th string.) This function f is computable: we can build a Turing machine for it that simply counts the length of the input string w and then waits for M to enumerate that many strings. Moreover, the range of f is clearly L so we are done. *3pts*

[*Some students designed a function f that is not computable (e.g., a function that requires actually solving the Halting Problem).*]