CSC363H5 Winter 2016 Assignment 1

Name: Akhil Gupta

SN: 1000357071

Question #	Score
1	
2	
3	
4	
Total	

Acknowledgements:

"I declare that I have not used any outside help (excluding the textbook, the notes on the course website, the teaching assistants, and the instructor) in completing this assignment."

Name: Akhil Gupta Date: January 25, 2016

Q1. Give a detailed description of a Turing Machine M which halts on every input and, when given a string $x \in \{0,1\}^*$, accepts if and only if there are strings $y, z \in \{0,1\}^*$ and a non-negative integer n such that $|y| = |z| = n$ and $x = y1^n z$. *Your Turing Machine must not alter any blank tape squares.* That is, your machine is free to write over the contents of the tape containing the input, but it is not allowed to write any symbol to a blank tape square other than $\sqcup$. In your description of M please give

1. A detailed description of the tape alphabet Γ (if you use any extra tape characters), giving the purpose of each extra character you introduce beyond $\Sigma \cup \{\sqcup\}$.
   Answer:
   (a) a: to mark any 0's in the input
   (b) a': to mark any 1's in the first substring (y) in $x = y1^n z$
   (c) a": to mark the 1's in the middle substring (1^n) and mark any 1's in the last substring (z)
   (d) $*$: to mark the entire first substring to pair up symbols from the middle substring to make sure that $|y| = n$
   (e) #: to mark the entire middle substring to pair up symbols from the first substring
   (f) $*'$: to mark the first substring to pair up symbols from the last substring to make sure that $|y| = |z|$
   (g) \$: to mark the last substring to pair up symbols from the first substring
2. A detailed description of the movement of the read-write head on an arbitrary input.
   Answer:
   (a) Check if string is empty, if it is, halt and accept.
   (b) Check if string has a length that is a multiple of 3. In order to check this, we go through the input until we reach a blank symbol by counting the number of symbols mod 3. If it is not a multiple of 3, halt and reject, else go to step 3.
   (c) Move to the beginning of the input and check the first unmarked symbol, if it is 0 then mark it with a, else if it is 1 then mark it with a'
   (d) Move to the end of the input and check the last two unmarked symbols. If it is 0 then mark it with a, else if it is 1 then mark it with a". Do the same thing for the second last symbol i.e. symbol to the left of the last marked symbol. So, we mark one symbol from the beginning of the input and two symbols from the end of the input.
   (e) Repeat steps 3 and 4.
   (f) So after the above steps, if the input was 100111001, we will have $a'aaa"a"a"aaa"$
   (g) Move to the beginning of the input and mark the first unmarked symbol with $*$ (i.e. on the modified input after performing steps 3 and 4).
   (h) Keep moving right until we reach a". Mark a" with #. Then repeat step 6. If we reach a instead of a", halt and reject, else go to step 10.
   (i) So after the above steps, the input was $a'aaa"a"a"aaa"$, we will have $***\#\#\#aa"$
   (j) Move to the beginning of the input and mark the first unmarked symbol with $*'$ (i.e. on the modified input after performing steps 7 and 8).
   (k) Keep moving right to the first unmarked symbol after # and mark it with \$.
   (l) So after the above steps, the input was $***\#\#\#aa"$, we will have $*'*'*'\#\#\#\$\$\$$

(m) Repeat steps 10 and 11 until all symbols are marked. If we mark a symbol with $*'$ and while moving right we reach the blank symbol, halt and reject. Else, halt and accept.

Q2. In this question we consider another variant of Turing Machine that errors in a regular pattern when it writes a symbol to the tape. Define a *Faulty Turing Machine* to be a Turing Machine M which writes an incorrect symbol to the tape every ten steps of the computation. More precisely, the machine M operates like so. Initially, the input $x \in \Sigma^*$ is written on the tape. For each positive integer $i = 1, 2, \dots$ when the machine M is about to perform the $10i^{\text{th}}$ transition and write the symbol γ to the tape, it instead chooses an arbitrary symbol $\gamma' \in \Gamma$ and writes γ' to the tape instead. The read-write head will still move to the left or right, as usual, without error. Acceptance of an input and the language computed by a Faulty Turing Machine are defined as usual.

Show that Faulty Turing Machines can simulate regular Turing Machines. That is, for every Turing Machine M show that there is a Faulty Turing Machine M' such that $\mathcal{L}(M') = \mathcal{L}(M)$ and for every input $x \in \Sigma^*$ the machine M halts on x if and only if M' halts on x. In your simulation, please give

(a) A detailed description of the tape alphabet Γ (if you use any extra tape characters), giving the purpose of each extra character you introduce beyond $\Sigma \cup \{\sqcup\}$.
Answer: Let α be the extra tape characters for M's tape alphabet. $\alpha = \{\alpha_1, \alpha_2, \alpha_3, \cdots, \alpha_n\}$.

(b) A detailed description of the movement of the read-write head on an arbitrary input.
Answer: To show that $\mathcal{L}(M') = \mathcal{L}(M)$, we have to show that the set of strings accepted by M' = the set of strings accepted by M. We know that the Faulty Turing Machine writes an arbitrary symbol γ' instead of γ every 10 steps of its computation. So to simulate regular Turing Machines (M) using Faulty Turing Machines (M'), we need to add extra tape characters to M's tape alphabet (α). When M is simulated on the same input x, every $10i^{\text{th}}$ transition, when M' writes γ', M writes a new character from its modified tape alphabet. So when M sees the arbitrary symbol, it will still move to the left or right, as usual, without error and execute exactly how M' would. Therefore, M halts on x if and only if M' halts on x.

Q3. Now define an *Extra Faulty Turing Machine* to be a Turing Machine M which writes an incorrect symbol to the tape every *two* steps of the computation. More precisely, the machine M operates like so. Initially, the input $x \in \Sigma^*$ is written on the tape. For each positive integer $i = 1, 2, \ldots$ when the machine M is about to perform the $2i^{\text{th}}$ transition and write the symbol γ to the tape, it instead chooses an arbitrary symbol $\gamma' \in \Gamma$ and writes γ' to the tape instead. The read-write head will still move to the left or right, as usual, without error. Acceptance of an input and the language computed by a Extra Faulty Turing Machine are defined as usual.

Let M be any Turing Machine which halts on every input and has input alphabet Σ. Using M, show that there is an Extra Faulty Turing Machine which accepts the language

$$L = \{\sigma_1 b_1 \sigma_2 b_2 \cdots \sigma_{n-1} b_{n-1} \sigma_n | n \in \mathbb{N}, \sigma_1 \sigma_2 \cdots \sigma_n \in \mathcal{L}(M), b_1, b_2, \ldots, b_n \in \Sigma\}.$$

In your simulation, please give

(a) A detailed description of the tape alphabet Γ (if you use any extra tape characters), giving the purpose of each extra character you introduce beyond $\Sigma \cup \{\sqcup\}$.
Answer:
  i. $\gamma' \in \Gamma \notin \Sigma$: the machine writes an arbitrary symbol every 2 steps of the computation. It marks the σ_i's with this
  ii. #: to check and mark the b_i's in the input so that the machine knows the last marked symbol

(b) A detailed description of the movement of the read-write head on an arbitrary input.
Answer: Since we have to show that Extra Faulty Turing Machine accepts L, then there has to be some string $x \in \mathcal{L}(M)$ such that M accepts x. Since the Extra Faulty Turing Machine writes γ' every 2nd step of the computation, we can use that to accept the given language.
  i. Move to the first unmarked symbol at the beginning of the input and copy it to the first blank symbol after the end of the input.
  ii. Move back to the first unmarked symbol of the input and since the Extra Faulty Turing Machine writes an arbitrary symbol from the tape alphabet on $2i^{\text{th}}$ transition, write $\gamma' \in \Gamma \notin \Sigma$
  iii. Check if the first symbol to the right of the last marked symbol (γ') is in M's input alphabet Σ, if not then halt and reject. Else mark it with a # and go to step 1.
  iv. Repeat steps 1 and 2 until all the σ_i's are copied and marked with arbitrary symbols $\gamma' \in \Gamma \notin \Sigma$ and all the b_i's are marked with #.
  v. After performing the above steps. we have the entire $\sigma_1 \sigma_2 \cdots \sigma_n$ at the end of the input. If the last marked symbol is #, halt and reject. Else if it is some $\gamma' \in \Gamma \notin \Sigma$ then continue.
  vi. So now we have to check if $\sigma_1 \sigma_2 \cdots \sigma_n$ is in $\mathcal{L}(M)$. This can be done by performing a subroutine which simulates a turing machine M' on $x = \sigma_1 \sigma_2 \cdots \sigma_n$, and if it accepts, we accept and if it rejects, we reject. Thus concluding whether $x \in \mathcal{L}(M)$ or $x \notin \mathcal{L}(M)$. If its not, then we halt and reject.

Q4. Let $M_0, M_1, M_2, \ldots, M_i, \ldots$ be an enumeration of all Turing Machines in some fixed computable order — that is, there is a computable function $g : \mathbb{N} \to \{0,1\}^*$ such that $g(i) = \langle M_i \rangle$ for all i. Let $f : \mathbb{N} \to \mathbb{N}$ be any one-to-one, computable function. Give a careful proof that the following language *is* semi-decidable:

$$L = \{\langle i \rangle | i \in \mathbb{N} \text{ and for all } j \leq f(i), \langle M_j \rangle \in \mathcal{L}(M_i)\}.$$

You may use the Church-Turing thesis when defining your algorithm, but be sure to use results given in class where necessary when justifying the correctness of your algorithm. Then prove that L is not decidable.

Proof: We have to show that $L \in SD$ and $L \notin D$. Let's start by proving that $L \in SD$.
To show L is semi-decidable, we need to give a Turing Machine M such that $\mathcal{L}(M) = L$. We can use $\overline{DIAG}$ $=\{\langle M \rangle | \langle M \rangle$ does not encode a TM or it does and $\langle M \rangle \in \mathcal{L}(M)\}$. Also, we know that $\overline{DIAG} \in SD$ from Professor Robert's lecture notes. We can achieve this using dovetailing and Universal Turing Machines. We know that computing the function $g(i)$ gives us the encoding for the i^{th} Turing Machine and computing the function f gives us a natural number $f(i)$. So we can use the outputs of these two functions in the main loop of our dovetailing algorithm.

Algorithm 1 Algorithm 1: Dovetailing

```
Input: A sequence of encodings of all integers ⟨i⟩ I = $\text{i}_1, i_2, \ldots, i_i, i \in \mathbb{N}$.
for each $i = 1, 2, 3, \ldots$ do
    for each $j = 1, 2, 3, \ldots, n$ such that $j \leq f(i)$ do
        Simulate the machines $M_1, \ldots, M_i$ on the encoding of machines $M_j$ encoded by $\langle M_j \rangle$ for $i$ steps
        Accept $\langle M_j \rangle$ if all of the above simulations accept (under the conditions)
        Reject $\langle M_j \rangle$ if all of the above simulations reject
    end for
end for
```

The input ⟨i⟩ is accepted by the above algorithm if and only if there are $i, j \in \mathbb{N}$ such that the sequence of machines M_i, $i \in \mathbb{N}$ accepted $\langle M_j \rangle$ after i computation steps, and so the algorithm accepts all inputs in L. If an input ⟨i⟩ is not in L, then either all machines in the sequence M_i reject $\langle M_j \rangle$ (in which case the above algorithm rejects), or none of the machines halt on x (in which case the algorithm does not halt). It follows that the language accepted by the above algorithm is L, and so L is semi-decidable.

Now let's prove that $L \notin D$. Assume L is decidable and let M_0 be the Turing Machine which decides L. Using M_0, we give a Turing Machine that decides A_{TM}, contradicting the fact that A_{TM} is not decidable. That is, there is a TM M such that $\mathcal{L}(M) = L$ and M always halts. We use the Church-Turing thesis. The algorithm for A_{TM} is as follows: On input $(\langle M_i \rangle, \langle M_j \rangle)$, reject if $\langle M_i \rangle$ is not a good encoding of a Turing Machine. If it is, then define the sequence of encodings of Turing Machines $\mathcal{M} = (\langle M_i \rangle)$ and the input string $x = \langle M_j \rangle$ which is also a sequence of encodings of Turing Machines where we can obtain $\langle M_i \rangle$ by computing $g(i)$ and obtain $\langle M_j \rangle$ by computing $f(i), j \leq f(i)$. Simulate M_0 on input $(\langle M_i \rangle, \langle M_j \rangle)$ and accept (or reject) if it accepts (rejects, respectively).

For any input $(\langle M_i \rangle, \langle M_j \rangle)$, if the algorithm for A_{TM} accepts the input, then M_0 accepts the input $(\langle M_i \rangle, \langle M_j \rangle)$. Since M_0 decides L, it follows by definition of L that all Turing Machines (under the conditions) in the sequence $\mathcal{M} = (\langle M_i \rangle)$ accepts $\langle M_j \rangle$. Thus $\langle M_i \rangle$ must accept $\langle M_j \rangle$, and so $(\langle M_i \rangle, \langle M_j \rangle) \in A_{TM}$.

On the other hand, if the algorithm for A_{TM} rejects the input, then M_0 must have rejected the input $(\langle M_i \rangle, \langle M_j \rangle)$, and so similarly we conclude that $\langle M_i \rangle$ does not accept $\langle M_j \rangle$, i.e. $\langle M_j \rangle \notin \mathcal{L}(M_i)$. It follows that $(\langle M_i \rangle, \langle M_j \rangle) \notin A_{TM}$. It is easy to see that the algorithm always halts, and so this algorithm decides A_{TM}. This leads to a contradiction.