Homework 3: Turing Machine, Variants, Decidability, and Recognizability

CS 212 Nature of Computation
Habib University
HW3 L3 3

Fall 2023

1. A Turing machine is said to *compute* a function f if, started with an input x on its tape, it halts with $f(x)$ on its tape. Consider a binary operator $\triangle$ and a function f defined as follows.

$$0\triangle 0 = 1, 0\triangle 1 = 1, 1\triangle 0 = 0, 1\triangle 1 = 1$$
$$f : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}^n, n \in \mathbb{Z} - \mathbb{Z}^-$$
$$f(a,b) = \{c_1 c_2 \dots c_n \mid c_i = a_i \triangle b_i, i = 1, 2, \dots, n\}$$

Consider the Turing machine, M, that computes f given a #-separated pair of binary strings as input. The Turing machine should print nothing if the function is undefined.

(a) [5 points] Give a high-level description of M.

Solution: We can build a Turing Machine M that computes f such that it traverses over the tape and inputs a and b. If the lengths don't match, it goes into the reject state. Similarly, if the machine reads anything other than a 0 or 1 (excluding the blank symbol and #), it goes into the reject state as these are conditions for which the function would be undefined. Else the machine computes c_i by applying the $\triangle$ operator on a_i and b_i and writes it on the tape.

We can provide a high level description of M. Machine M starts with the head at a_1. We use "skip" to denote one or more moves of the head that do not replace any tape symbols.

1. Skip right until the very first blank symbol (␣) is encountered (indicating that b has ended), and write a #. This # indicates end of the input string (if anything other than a 0, 1, # is encountered, we move to the reject state)
2. Skip left until the first blank symbol is encountered and move right. Hence, we are starting from a_1.
3. Remember this symbol, call it a_i, and move right.
4. If the current symbol is a # (input a is exhausted)
   (a) Skip right to find the first symbol that is a 0 or a 1.
   (b) Remember the current symbol, call it b_i, and replace it with ␣.

(c) Move right to the # (end of input b). If instead of a # we read a 0 or a 1, we move to the reject state since a has ended even though b has not. So a and b were not of equal lengths.
(d) Skip right to find the first ␣ (next output space).
(e) Write $a_i \triangle b_i$ on the space.
(f) Skip left until a # is encountered (end of input b), and replace with a ␣. Skip left until a # (end of input a) and replace with a ␣.
(g) Skip right until a 0 or a 1 is read, thus starting read from the beginning of c.
(h) Accept

5. If the current symbol is a 0 or a 1 (input remains to be processed)
   (a) Skip right to find the first #. Then skip right to find the first symbol that is a 0 or a 1. If instead of a 0 or a 1, we encounter a #, then we move to the reject state since b has ended even though a has not. So b and a were not of equal lengths.
   (b) Remember the current symbol, call it b_i, and replace with ␣.
   (c) Skip right to find #. Skip right to find the first ␣. (next output space).
   (d) Write $a_i \triangle b_i$ on the space.
   (e) Skip left to # (end of input). Skip left to # (end of string a). Skip left to the first ␣ (just processed bit of a) and move right to start reading from a_i.
6. Repeat from Step 3

(b) [5 points] Explore the website, `https://turingmachine.io`, in order to write a formal description of M on it. Download the description from the website and submit the downloaded YAML file along with the eventual PDF.

2. [10 points] An *Euclidean-Space* Turing machine has the usual finite-state control but a tape that extends in a three-dimensional grid of cells, infinite in all directions. Both the head and the first input symbol are initially placed at a cell designated as the origin. Each consecutive input symbol is in one of the six neighboring cells and does not overwrite a previous symbol. The head can move in one transition to any of the six neighboring cells. All other workings of the Turing machine are as usual.

   Provide a formal description of the *Euclidean-Space* Turing Machine and prove that it is equivalent to an ordinary Turing machine. Recall that two models are equivalent if each can simulate the other.

Solution:

An *Euclidean-Space* Turing Machine has a 3D grid of cells, infinite in all directions, so it can move in all 3 dimensions. We consider the 3D grid to represent the x, y, z coordinate systems for ease, with right and left movements being on the x-axis, up and down movements being on the z-axis, and forward and backward movements being on the y-axis. Let $ESTM$ be an Euclidean-Space Turing Machine. Formally, we can define $ESTM$ as a 7-Tuple: $ESTM = (Q, \Sigma, \Gamma, \delta, q_o, q_{\text{accept}}, q_{\text{reject}})$ where:

1. Q is a finite set of states.
2. Σ is a finite set of input symbols.
3. Γ is a finite set of tape symbols, where $\Sigma \subseteq \Gamma$ and Γ has additional tape symbol(s)
4. $\delta : Q \times \Gamma \to Q \times \Gamma \times \{\text{L, R, U, D, F, B}\}$ is the transition function.
   The machine is in state $q \in Q$, and scans tape symbol $\gamma \in \Gamma$. It changes to state $q' \in Q$, writes or reads symbol from the tape, and moves in any of the six directions left(L), right(R), up(U), down(D), forward(F), backward(B).
5. q_o is the start state.
6. q_{accept} is the accepting state.
7. q_{reject} is the rejecting state.

We can show that the *Euclidean-Space* Turing Machine is equivalent to a normal Turing Machine by showing that a normal Turing Machine can simulate the *Euclidean-Space* Turing Machine, and vice versa. Let M be an ordinary Turing Machine and $ESTM$ be an *Euclidean-Space* Turing Machine.

- ***ESTM* can simulate *M***

  This is trivial as the $ESTM$ can simulate M by simply ignoring its ability to move in any additional directions, utilising only a single row (take x-axis for example), having all the same transitions and states as M. Therefore, any language recognized by M can be recognized by $ESTM$. Thus, $ESTM$ simulates M.

- ***M* can simulate *ESTM*** For M to simulate an $ESTM$, we can introduce mapping; map each cell in the 3D grid of the $ESTM$ to a unique cell in the 1D tape of M. This can be done by assigning a unique number to each cell in M's tape, and for each move in the $ESTM$, we move in M based on some function or formula.

We can do this by assigning a natural number $n \in \mathbb{N}$ to each cell in M's tape, starting from 0 which corresponds to the starting cell in $ESTM$. The corresponding cells to the right are labelled 1, 2, 3 ... and so on, and on the left are labelled -1, -2, -3 ... and so on. Next we need to map each movement on the 3D tape to the 1D tape. We can define a pairing function for this purpose, such as the Cantor Pairing Function (a bijective function), defined as $\pi(x, y) = \frac{1}{2}(x + y)(x + y + 1) + y$. This function uniquely assigns one natural number to each pair of natural numbers. We can extend this same idea to 3D by using the Cantor Pairing Function twice: $\pi_3(x, y, z) = \pi(\pi(x, y), z)$. This way, M would simulate the movements by translating them into movements to the corresponding position on the tape according to the mapping function we have defined. For example, if the head in the $ESTM$ moves up (+1 unit in the z direction), M would move its head to the cell on the tape corresponding to the new $(x, y, z + 1)$ coordinates.

The reads and writes remain the same for M as they were in $ESTM$. This way M can simulate the $ESTM$ as each cell of M is mapped onto a corresponding cell of $ESTM$.

Since M and $ESTM$ can both simulate each other, we have proved they are equivalent and an $ESTM$ is no more powerful than an ordinary Turing Machine.

■

3. [10 points] Let $A = \{L \mid L \text{ is decidable but not context-free}\}$. Prove that every element of A contains an unrecognizable subset.

Solution: Let $L \in A$. Then L is decidable but not context-free. Since every finite language is context-free, then we can infer that L is infinite.

We consider the power set of L; $\mathcal{P}(L)$ since the power set is essentially a set containing all possible subsets of L. According to Cantor's Theorem, the cardinality of the power set of any set is strictly greater than the cardinality of the set itself; $\mid \mathcal{P}(L) \mid > \mid L \mid$. Since L is infinite, then $\mathcal{P}(L)$ must also be infinite, however, as a direct consequence of the Cantor's Theorem, we can infer that $\mathcal{P}(L)$ is uncountably infinite unlike L which is countably infinite since the cardinality of the power set is always strictly greater than the cardinality of the set itself. `COROLLARY 4.18` of the book states that there are some languages that are not Turing recognizable - essentially showing that the set of all Turing Machines is countable.

Since there exists uncountably many subsets in $\mathcal{P}(L)$, but countably many Turing Machines, then there must exist some subsets in $\mathcal{P}(L)$ for which no corresponding Turing Machine exists, since a bijection cannot be made from the set of Turing Machines, to $\mathcal{P}(L)$.

Therefore, for every element $L \in A$, L must have an unrecognizable subset.

■