CS 245: TUT 105 - Tutorial 11

Last time: FOL Soundness & Completeness

This time: Turing Machines

Meme(s) of the week:

Borrowed from Reddit.com/r/ProgrammerHumor

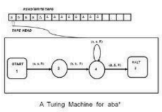
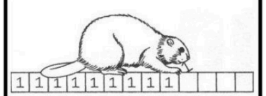
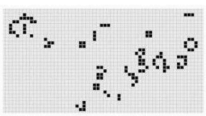
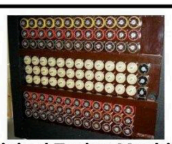
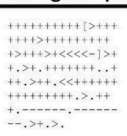
	Function purist A Turing machine is something that manipulates symbols on a strip to solve problems	Function neutral A Turing machine is something that manipulates a strip or any data structure	Function rebel A Turing machine is something with a strip
Form purist A Turing machine is mathematical model	 Turing machine model is Turing complete	 Busy Beaver game is Turing complete	 Conway's Game of Life is Turing complete
Form neutral A Turing machine could be an implementation	 Original Turing Machine is Turing complete	 C language is Turing complete	 PowerPoint is Turing complete
Form rebel A Turing machine could be anything	 BrainF**k is Turing complete	 MySQL is Turing complete	 Snakes & Ladders is Turing complete

Figure 1: NAND gate in AoE II's editor, designed to show its internal workings–simpler implementations are possible. Every bit is represented by two rails (grass for 0, a bridge for 1). Only one rail is active at a time, with a goat acting as the signal carrier. When the gate fires, the bit-goats are removed and a new bit-goat is placed in its respective output rail. To avoid race conditions, 'gate ready' rails (ice) are set, with a signal-goat in the rail closest to the gate indicating that it can start the calculation.

Review:

Motivation:
We would like to be able to model a "theory of computation."
For that, we need a "model of computation," i.e. an abstract representation for a device capable of processing some input, doing some computation, and outputting the result.

One simple but powerful example of such a model is the Turing Machine (TM), which, informally, consists of an infinite sequence of tape subdivided into squares, and a tape "head" which advances over a single square in discrete time steps. In a given step, it can:

- Read the symbol on the tape currently under the head.
- Change its internal state, based on this symbol.
- Write a symbol to the tape at this square.
- Move one square left or right on the next time step, or stay in the same place.

We now state the mathematical definition of Turing Machine.

Definition. A *Turing machine* is an abstract machine described by the triple

$$M = (Q, \Sigma, \delta)$$

where

- Q is a finite set of *internal states*;
- Σ is the *tape alphabet*, a finite set of *symbols* which includes the *blank* symbol $\square$; and
- $\delta: Q \times \Sigma \to (Q \cup \{\mathbf{Halt}\}) \times \Sigma \times \{L, R, S\}$ is the *transition function*.

strings over finite alphabet, σ

Def.: A function $f: \sigma^* \to \sigma^*$ is called **computable** if $\exists$ a TM M over a tape alphabet $\Sigma \supset \sigma$ s.t. $\forall s \in \sigma^*$, M halts when run on s and outputs $f(s)$.

Example: The task of determining if a number is even or odd can also be represented as a function. For any natural number $n \in \mathbb{N}$, let us write $\langle n \rangle \in \{0,1\}^*$ to denote its binary encoding. Define

$$\textsf{IsEven}\colon \{0,1\}^* \to \{0,1\}$$

to be the function where for each $n \in \mathbb{N}$,

$$\textsf{IsEven}(\langle n \rangle) = \begin{cases} 1 & \text{if } n \text{ is even} \\ 0 & \text{otherwise.} \end{cases}$$

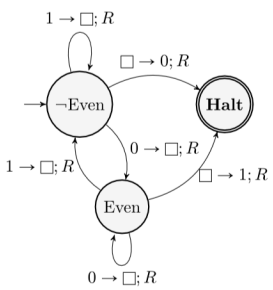
For example, the input string

⋯		1	0	1	1	0		⋯

corresponds to the number 22 so a Turing machine that computes **IsEven** halts with the tape containing a single 1:

⋯				1				⋯

A Turing machine that computes this function is this one:



Alternate notation: Instead of labeling transition edges in a TM diagram like in the photo, you can simply label them with:

$\alpha; \beta; \gamma$, where

$\alpha \in \Sigma$, $\beta \in \Sigma$, $\gamma \in \{L, R, S\}$

It is similar to the **Erase** function in that it erases all the symbols on the tape as it goes, but it also uses two states to remember if the last digit read on the tape was a 0 or not. With this information, once it reaches the end of the input string, it can write 0 or 1 and then halt, as required.

It is believed (although this is not the kind of statement that can be proven, mathematically), that Turing machines can model any algorithmic problem…

In other words, any function that can be computed by an algorithm can be computed by a Turing Machine. This is the Church-Turing Thesis.

Church-Turing Thesis: Every function that can be computed by algorithms in any reasonable model of computation can also be computed by a Turing machine.

~ Fin. ~

Practice Problems:

1. Running a TM by hand...

Let T be the Turing machine with set of states $S = \{s_0, s_1, s_2, s_3\}$, input alphabet $I = \{0, 1, B\}$, start state s_0, and transition function defined by the five-tuples

- $(s_0, 0, s_1, 0, R)$
- $(s_0, 1, s_1, 0, L)$
- $(s_0, B, s_1, 1, R)$
- $(s_1, 0, s_2, 1, R)$
- $(s_1, 1, s_1, 1, R)$
- $(s_1, B, s_2, 0, R)$
- $(s_2, B, s_3, 0, R)$

Note: In this problem, we also define a start state for our TM. One can also define some subset of the states as accepting; these are denoted by double circles in our TM graphs. If a state has no transitions out of it, though, it is automatically accepting.

There are many similar notational variations in the description s of TMs. You will see and get used to a few of them.

For each of the initial tapes below, determine the final tape when T halts, assuming that T starts in initial position (in start state s_0, with read/write head positioned over the leftmost non-blank symbol on tape).

a)

...	B	B	0	1	0	1	B	B	...

b)

...	B	B	1	1	1	B	B	B	...

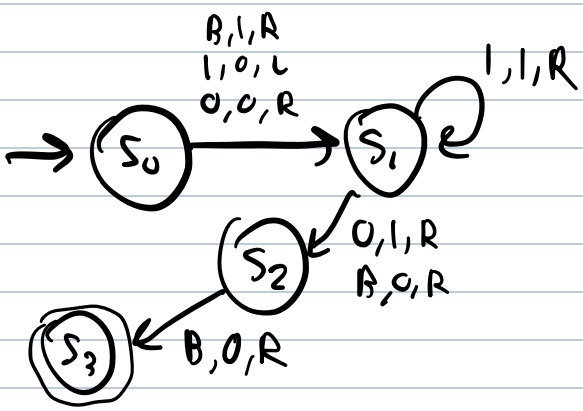
c)

...	B	B	0	0	B	0	0	B	...

d)

...	B	B	B	B	B	B	B	B	...

Sol.: First, draw the TM...



* See next page!

We will indicate the configuration of the Turing machine using a notation such as $0[s_2]1B1$, as described in the solution to Exercise 1. (This means that the machine is in state s_2, the tape is blank except for a portion that reads $01B1$, and the tape head points to the left-most 1.) We indicate the successive configurations with arrows.

a) Initially the configuration is $[s_0]0101$. Using the first five-tuple, the machine next enters configuration $0[s_1]101$. Thereafter it proceeds as follows: $0[s_1]101 \rightarrow 01[s_1]01 \rightarrow 011[s_2]1$. Since there is no five-tuple for this combination (in state s_2 reading a 1), the machine halts. Thus (the nonblank portion of) the final tape reads 0111.

b) $[s_0]111 \rightarrow [s_1]B011 \rightarrow 0[s_2]011 \rightarrow$ halt; final tape 0011

c) $[s_0]00B00 \rightarrow 0[s_1]0B00 \rightarrow 01[s_2]B00 \rightarrow 010[s_3]00 \rightarrow$ halt; final tape 01000

d) $[s_0]B \rightarrow 1[s_1]B \rightarrow 10[s_2]B \rightarrow 100[s_3]B \rightarrow$ halt; final tape 100

Side note: Shorthand for TM State:

The "configuration notation" for TMs represents a description of the TM — (both tape and tape head) — at a given time step. In it, a string such as

$0011s_i11B10$ alternatively written $0011[s_i]11B10$

indicates that the TM is in state s_i, that the tape is blank except the portion that reads 001111B10, and that the TM read/write head reads the 1 immediately following s_i (the 3rd occurrence of 1 on the tape, in magenta).

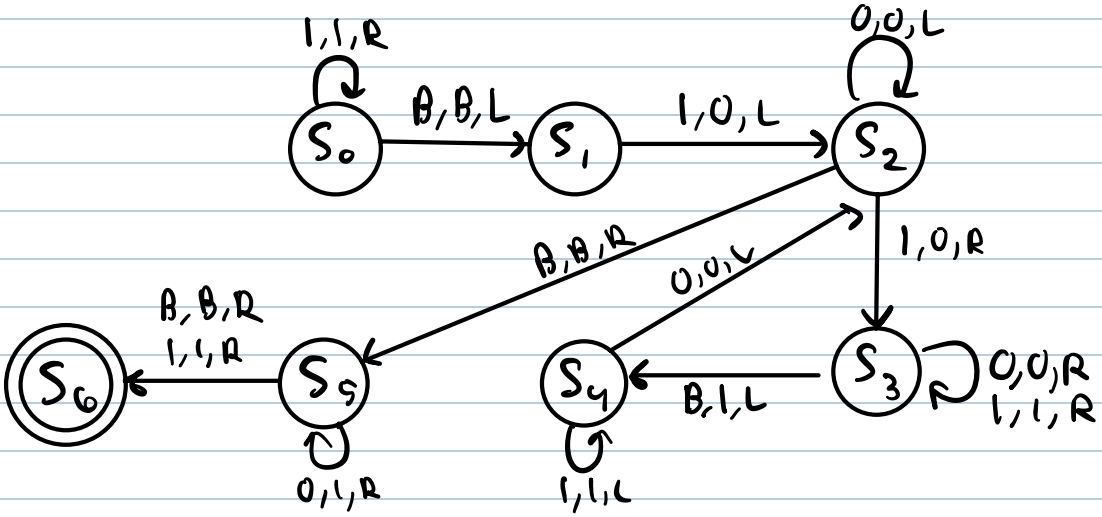
2. Building a TM, I:

Construct a TM that computes the function f(n) = 2n for all natural numbers n.

Solution:

We start with a string of $n+1$ 1's, and we want to end up with a string of $2n+1$ 1's. Our idea will be to replace the last 1 with a 0, then for each 1 to the left of the 0, write a new 1 to the right of the 0. To keep track of which 1's we have processed so far, we will change each left-side 1 with a 0 as we process it. At the end, we will change all the 0's back to 1's. Basically our states will mean the following ("first" means "first encountered"): s_0, scan right for last 1; s_1, change the last 1 to 0; s_2, scan left to first 1; s_3, scan right for end of tape (having replaced the 1 where we started with a 0) and add a 1 at the end; s_4, scan left to first 0; s_5, replace the remaining 0's with 1's; s_6, halt.

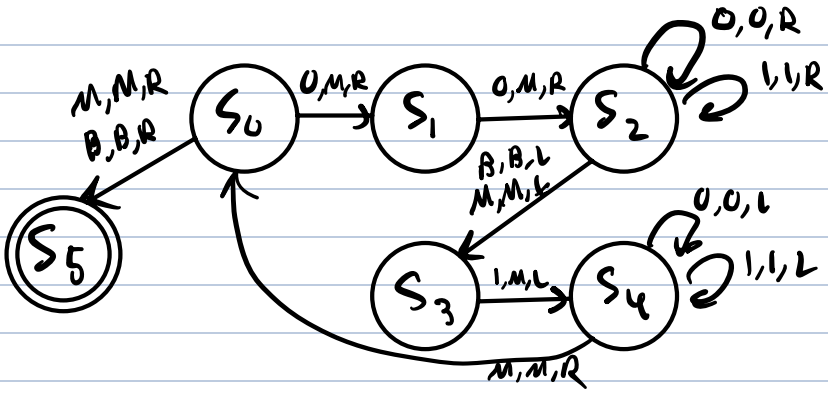
The needed five-tuples are as follows: $(s_0, 1, s_0, 1, R)$, (s_0, B, s_1, B, L), $(s_1, 1, s_2, 0, L)$, $(s_2, 0, s_2, 0, L)$, $(s_2, 1, s_3, 0, R)$, (s_2, B, s_5, B, R), $(s_3, 0, s_3, 0, R)$, $(s_3, 1, s_3, 1, R)$, $(s_3, B, s_4, 1, L)$, $(s_4, 1, s_4, 1, L)$, $(s_4, 0, s_2, 0, L)$, $(s_5, 0, s_5, 1, R)$, $(s_5, 1, s_6, 1, R)$, (s_5, B, s_6, B, R).



3. Building a TM, II.

Construct a Turing machine that recognizes the set $\{0^{2n}1^n \mid n \geq 0\}$

Solution:



~ Fin. ~