˜
˜

CS 373: Theory of Computation
Madhusudan Parthasarathy

Lecture 15: Universal Turing Machines

11 March 2010

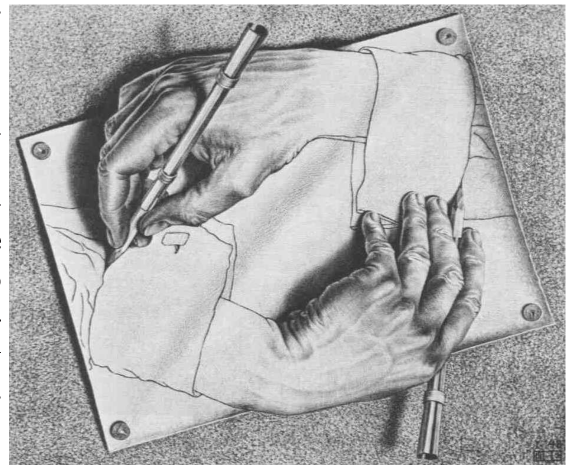
This lecture presents the Universal Turing Machine. The associated lecture has also material covering closure properties of Turing machines, but we have no notes for this as of now. Please refer to the video for this material.

1 Universal Turing machines: TMs simulating TMs

1.1 Motivation

We already seen that a TM can simulate a DFA. We think about TMs as being just regular computer programs. So think about an interpreter. What is it? It is a program that reads in another program (for example, think about Java virtual machine) and runs it.[1]

So, what would be the equivalent of an interpreter in the language of Turing machines? Well, its a TM that reads in a description of a TM M, and an input w for it, and simulates running M on w.



Initially this construct looks very weird - inherently circular in nature. But it is useful for the same reason interpreters are useful: It enable us to manipulate TMs (i.e., programs) directly and modify them without knowing in advance what they are. In particular, we can start talking computationally about ways of manipulating TMs (i.e., programs).

For example, in a perfect world (which we are not living in, naturally), we would like to give a formal specification of a program (say, a TM that decides if a prime number is prime), and have another program that would swallow this description and spits out the program performing this computation (i.e., have a computer that writes our programs for us).

A more realistic example is a compiler which translates (say) Java code into assembly code. It takes code as input and procedures code in a different language as output. We could also build an optimizer that reads Java code and produces new code, also in Java but more efficient. Or a cheating helper program that reads Java code and writes out a new version with different variable names and modified comments.

[1]Things of course are way more complicated in practice, since Java virtual machines nowadays usually compile portions of the code being run frequently to achieve faster performance (i.e., just in time compilation [JIT]), but still, you can safely think about a JVM as an interpreter.

1.2 The universal Turing machine

We would like to build the ***universal Turing machine*** U_{TM} that recognizes the language

$$\mathsf{A}_{\mathsf{TM}} = \left\{ \langle \mathtt{T}, w \rangle \;\middle|\; \mathtt{T} \text{ is a } \mathsf{TM} \text{ and } \mathtt{T} \text{ accepts } w \right\}.$$

We emphasize that U_{TM} is <u>**not**</u> a decider. Namely, its stops only if $\mathtt{T}$ accepts w, but it might run forever if $\mathtt{T}$ does not accept w.

To simplify our discussion, we assume that $\mathtt{T}$ is a single tape machine with some fixed alphabet (say $\Sigma_{\mathtt{T}} = \{\mathtt{0}, \mathtt{1}\}$ and the tape alphabet is $\Gamma_{\mathtt{T}} = \{\mathtt{0}, \mathtt{1}, \llcorner\!\lrcorner\}$. To simplify the discussion, the TM for A_{TM} is going to be a multi-tape machine. Naturally, one can convert this TM into a single tape TM.

So, the input for U_{TM} is an encoding $\langle \mathtt{T}, w \rangle$. As a first step, the U_{TM} would verify that the input is in the right format (such a reasonable encoding for a TM was given as an exercise in the homework). The U_{TM} would copy different components of the input into different tapes:

- ✇$_1$: Transition function δ of $\mathtt{T}$.

  It is going to be a sequence (separated by \$) of transitions. A transition $(q, c) \rightarrow (q', t, \mathtt{L})$ would be encouded as a string of the form:

  $$(\#q, c) - (\#q', t, \mathtt{L})$$

  where $\#q$ is the index which is the index of the state q (in $\mathtt{T}$) and $\#q'$ is the index of q'.

  More specificially, you an think about the states of $\mathtt{T}$ being numbered between 1 and m, and $\#q$ is just the binary representation of the index of the state q.

- ✇$_2$: $\#q_0$ – the initial state of $\mathtt{T}$.

- ✇$_3$: $\#q_{\mathrm{acc}}$ – the accept state of $\mathtt{T}$.

- ✇$_4$: $\#q_{\mathrm{rej}}$ – the reject state of $\mathtt{T}$.

- ✇$_5$: $\$w$ – the input tape to be handled.

Once done copying the input, the U_{TM} would move the head of ✇$_5$ to the beginning of the tape. It then performs the following loop:

(I) Loop:

  (i) Scan ✇$_1$ to find transition matching state on ✇$_2$ and the character under the head of ✇$_5$.

  (ii) Update state on ✇$_2$.

  (iii) Update character and head position on ✇$_5$.

  We repeat this till the state in ✇$_2$ is equal to the state written on either ✇$_3$ (q_{acc}) or ✇$_4$ (q_{rej}).

Naturally, U_{TM} accepts if ✇$_2$ = ✇$_3$ and rejects if ✇$_2$ = ✇$_4$ at any point during the simulation.