(Re)Introduction to Turing Machines

November 11, 2010

Computability and Logic

What is a Turing Machine?

- Named after (not by) Alan M. Turing.
- Perhaps the most important computational model, from a theoretical viewpoint.
- Simple, yet apparently universal.

TM Components

$(Q, \Sigma, \Gamma, \delta, q_0, q_{accept}, q_{reject})$

- Q: finite set of control states
- Σ: input alphabet
- Γ: tape alphabet ($\Sigma \subseteq \Gamma$ and $\sqcup \in \Gamma \setminus \Sigma$)
- δ: transition function
  $$\delta : Q \times \Gamma \to Q \times \Gamma \times \{L, R\}$$
- q_0: initial control state
- q_{accept}, q_{reject}: accepting and rejecting (halting) states

The Setup

- Write the input at start of an infinite blank tape.
- Run the TM at the start of the tape.
  - $\delta : Q \times \Gamma \rightarrow Q \times \Gamma \times \{L, R\}$
- TM halts iff we enter states q_{accept} or q_{reject}.
  - It is possible that the TM might never halt.

TM Configurations

- A configuration like $011q_2 01\textvisiblespace 110$ means
  - The contents of the tape is $01101\textvisiblespace 110$ (padded with blanks $\textvisiblespace$)
  - The TM is in control state q_2
  - The TM head is pointing to the second 0.
- Configurations are always finite. Why?

Transitions in Detail

A machine makes one of the following transitions

$$wa q_i bcz \rightarrow wad q_j cz \quad \text{if } \delta(q_i, b) = (q_j, d, R)$$

$$wa q_i bcz \rightarrow w q_j adcz \quad \text{if } \delta(q_i, b) = (q_j, d, L)$$

TM Languages

- A TM <u>accepts</u> a string if, given the string as input, the TM reaches q_{accept}.

- A language is <u>recognizable</u> (a.k.a. recursively enumerable) if there is a turing machine that accepts exactly the strings in the language.

- A language is <u>decidable</u> (a.k.a. recursive) if it is accepted by a TM that always halts (i.e., if the TM always ends up in q_{accept} or q_{reject}.)

Note on Terminology

- "**recursive**" does *not* mean that the language has some kind of self-referential structure.
- Terminology comes from "recursive functions" of Godel and Kleene.
  - These turn out to be exactly the functions on integers that are implementable with Turing machines.

Acceptance vs. Decision

- One reason the distinction is important is that there are languages that are:
  - accepted by some TM
  - not decided by any TM

TM Demo

- http://ironphoenix.org/tril/tm/

TM Programming Tips

- Divide the work into different phases (subroutines)
- Controller has an arbitrarily large “finite memory”.
- Squares can be “marked” and “unmarked”
  - In arbitrarily many, but finite, ways.
- Take advantage of TM extensions (forthcoming)

Terminology

- A language that is recognized by some TM is called “**recursive**”.
- A language that is accepted by some TM is called “**recursively enumerable**” (R.E.).
- So recursive $\Rightarrow$ recursively-enumerable
  - But the converse doesn't always hold!

More on the Decidable vs. Recognizable Distinction

- If a language is decidable, then its complement is decidable.
  - Why?
- If a language is recognizable, and its complement is recognizable, then the language is decidable.
  - Why?

TMs Computing

- Rather than accepting a language, we can use a TM to **compute a function**:
  - The machine starts with some string x on its tape initially.
  - The machine halts with some string y on its tape finally.
  - The corresponding function f would have

$$f(x) = y$$

TMs Computing Partial Functions

- For some initial strings x the TM might **diverge**.
- More generally, we say the machine computes a **partial function**, and f(x) is undefined for such x.
- An ordinary function is a special case of a partial function, one which is nowhere undefined. We call such a partial function a **total function** to emphasize the distinction.

TM Variations

- The following yield no extra power:
  - Adding the option to stay-in-place rather than moving L/R.
  - Making the tape infinite in both directions (HW)
  - Adding an extra "Erase Tape" move.
  - Multiple tapes with multiple (independent) read/write heads
  - 2-D infinite tape
  - Nondeterminism(!)
- Many attempts to define models of computation; all turn out to be equivalent to Turing Machines.
  - If you can do it in C++, a TM can do it (slowly, encodedly)
- Church's Thesis (a.k.a. Church-Turing Thesis)
  - Any "intuitively computable" procedure can be performed by a TM

Turing Machines as Enumerators

- Several variant definitions. Each specify a language L
  1. A TM that prints out the members of L, one at a time (but not necessarily in any particular order)
  2. A TM that prints out the members of L, one at at time (but...) with possible repeats in some
  3. A TM that, given an integer n, returns the nth element of a sequence like (1) above.
  4. A TM that, given an integer n, returns the nth element of a sequence like (2) above.
- For a fixed language, all these are interconvertable.
- A language is recognizable iff it can be enumerated.

Encoding TMs as Strings

- We know that a TM can be described fully by a finite list of transition rules, of the form:
  - q_0, ␣ $\rightarrow$ q_i, ␣, R
  - q_i, ␣ $\rightarrow$ q_j, ␣, L
  - q_i, x $\rightarrow$ q_k, x, L
  - q_m, a $\rightarrow$ q_n, x, R etc.
- We can devise a way to encode an **arbitrary** set of such rules into a **fixed alphabet**.
- Although the number of states and tape symbols can be arbitrary
  - we can **encode** these by using strings of symbols, say {0, 1}
  - concatenate the symbols to describe rules
  - concatenate rules to describe machines.

Encoding TMs as Strings

- Not **every** string of symbols in the encoding alphabet must correspond to a well-formed TM description.
- But we can determine algorithmically which are and which aren't.
  - In fact, a TM can do so!
  - For those that aren't, assume they describe a default TM that immediately halts and rejects.
- We thus can enumerate of **all TM's**:

  $T_0, T_1, T_2, \ldots$
- We can do the same for encodings of **initial tapes**:

  $x_0, x_1, x_2, \ldots$

Universal Turing

- An encoding of a Turing machine can be viewed as a *program*.
- A Turing machine that interprets such a program to carry out the actions specified is called a **Universal Turing Machine** (UTM).

Universal Turing

- UTMs can be shown to exist by constructing them.
- Think about what would be required.
  - The tape has to hold the tape of the machine being simulated.
  - The tape has to hold the program of the machine being simulated.
  - The program must be laid out in such a way that the necessary markers can be inserted to keep track of the current state, etc.
- All this is possible, if somewhat laborious to construct.

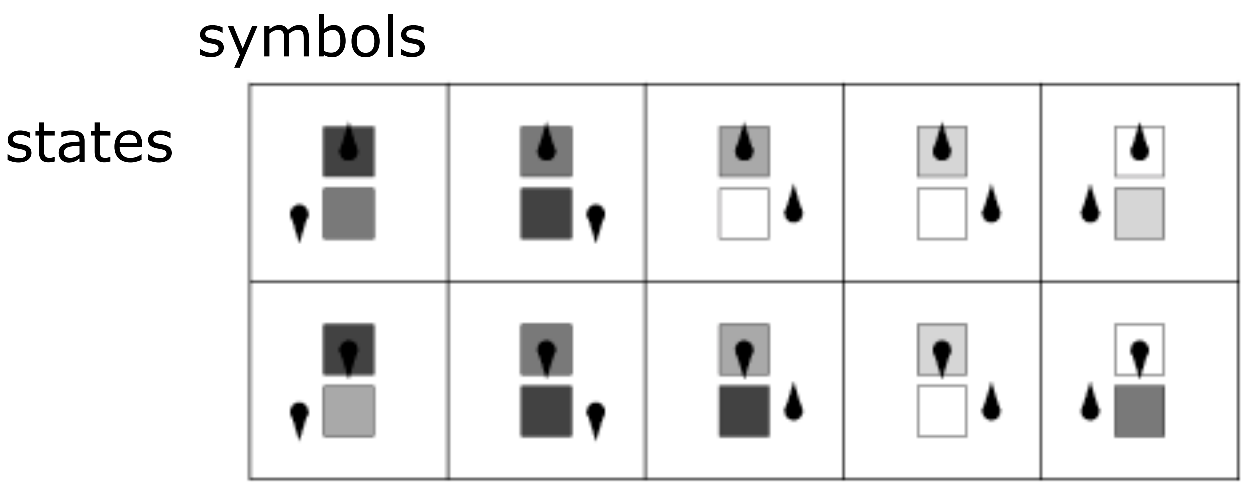
- Whether a machine is universal will depend on the particular encoding used.

Specific UTMs

- The first was constructed by Turing himself.
- Shannon showed any UTM could be converted either to a 2-symbol machine or to a 2-state machine by augmenting states or symbols, respectively.
- Minsky (1960) gave a 7-state 6-symbol machine.
- Watanabe (1961) gave an 8-state 5-symbol machine.
- Minsky (1962) gave a 7-state 4-symbol machine.
- Rogozhin (1996) gave a 4-state 6-symbol machine
- Wolfram and Reed (2002) gave a 2-state 5-symbol machine.
- Smith and Wolfram (2007) gave a 2-state 3-symbol machine.
- No 2-state 2-symbol UTM exists.

A Specific Universal Turing Machine

- 2-state, 5 symbol UTM published by **Wolfram** in 2002



adapted from Wolfram, S. *A New Kind of Science*. Wolfram Media, p. 707, 2002.