CSE 322 - Introduction to Formal Methods in Computer Science
Turing Machines

Dave Bacon
Department of Computer Science & Engineering, University of Washington

Today we finally make it up to....computers! Well at least to the model which best captures what we mean by a computer, the Turing Machine. Yeah!

Okay so what is a Turing machine? It's a big box with an arrow coming out of it and a tape. Or at least that's the picture you'll see drawn in most books. The box representing the computing machine and the tape represents a memory where you can store information. The arrow represents the read-write head of the machine, which can, as you might guess, read and write symbols on the tape. Oh, and by the way, the tape is infinite. Usually we make the tape have one end, say the left and then go off for ever to infinity.

So how does a Turing machine work. Well some input is put onto the tape, with the leftmost symbol being at the very beginning of the tape. Everywhere else the tape is blank (we define a blank symbol "␣".) The Turing machine read-write head starts at the first symbol on the tape. Then it proceeds to compute! How does it compute? Well it has an internal state, like a finite automaton. But now it can make transitions to new states conditional on what state it is in, and what symbol is currently under the read-write head on the tape. Further, when it makes a transition to a new state, it can write a symbol onto the tape at the position where the read-wright head is, and then it *must* move the head either left or right. Finally the machine has two designated states which are accept and reject states. When the machine enters these states it immediately halts and we say that the machine accepts or rejects the input. Note that the machine doesn't have to enter the accept or reject states: it can go on running forever and never halt.

Okay lets give a formal definition of a Turing machine. A Turing machine is going to be a seven tuple, the largest tuple we've encountered! Okay here it is a Turing machine is the 7-tuple $(Q, \Sigma, \Gamma, \delta, q_0, q_a, q_r)$ where

1. Q is a finite set, the set of states.
2. Σ is a finite set, the input alphabet no containing the blank symbol ␣.
3. Γ is the tape alphabet, where $␣ \in \Gamma$ and $\Sigma \subseteq \Gamma$.
4. δ is the transition function. It is a function from $Q \times \Gamma$ to $Q \times \Gamma \times \{L, R\}$, $\delta : Q \times \Gamma \to Q \times \Gamma \times \{L, R\}$.
5. $q_0 \in Q$ is the start state
6. $q_a \in Q$ is the accept state
7. $q_r \in Q$ is the reject state and is not equal to the accept state, $q_a \neq q_r$.

Okay so the most important part of the definition, is, as always, the transition function. The transition function here has a domain of $Q \times \Gamma$. This represents the state of the machine and the symbol under the read write head before the transition. The range of the function is $Q \times \Gamma \times \{L, R\}$. This presents what the state the machine will transition to, what it will right on the tape before moving, and then whether the read-write head will move left or right. If the machine is at the beginning of the tape and it receives instructions to move left, we keep the head at the beginning of the tape.

As a Turing machine computes, the state of the machine can change as well as where the read-write head is as well as what symbols are on the tape. Thus a configuration of a Turing machine is the specification of a state, the location of a read-write head, and the symbols on the tape. We use a cute shorthand for specifying configurations. In particular suppose that we are in the state q and the tape has the string w on it. Further suppose that we are are location i in the tape string $w = w_1 w_2 \dots w_m$, then we can write w as $w = uv$ where $u = w_1 w_2 \dots w_{i-1}$ and $v = w_i w_{i+1} \dots w_m$. To express that we are currently in state q with the read-write head at this location we write this as uqv. This completely specifies a configuration and is a nice shorthand.

Using this notation we can formalize how a Turing machine works. In particular we say configuration C_1 yields configuration C_2 if the Turing machine can legally go from C_1 to C_2 in a single step. Suppose that $a, b, c \in \Gamma$ and $u, v \in \Gamma^*$ and that q_i and q_j are states. From these elements we can construct two configurations, uaq_ibv and uq_jacv. We say that uaq_ibv yields $uacq_jv$ if $\delta(q_i, b) = (q_j, c, R)$. Similarly we say that uaq_ibv yields $uacq_jv$. There are, of course, special cases. For example when we are on the left end, the configuration q_ibv yields q_jcv if the transition is left moving. Further technically we need to add the ␣ symbol to the configuration uaq_i when we are making a right transition.

Some more definitions. The *start configuration* of the machine M on input w is the configuration q_0w. An *accepting configuration* is one where the state of the configuration is q_a. A *rejecting configuration* is one where the state of the configuration is q_r. Accepting and rejecting configurations together are called *halting cofingurations*. A Turing machine M accepts input w if a sequence of configurations $C_1, C_2, \ldots, C_k$ exists, where

1. C_1 is the start configuration of M with input w.
2. Each C_i yields C_{i+1}.
3. C_k is an accepting configuration.

The collection of strings that a machine M accepts is the language of M and is denoted, by, you guessed it $L(M)$.

Okay a few more definitions. We say a language is *Turing-recognizable* if some Turing machine recognizes it (these are sometimes called recursively enumerable languages.) Now Turing machines may end up not accepting by either entering the reject state or by looping forever (not halting.) It is often useful to talk about Turing machines that halt on all inputs. These machines never loop. We call such Turing machines deciders. A decider that recognizes some language also is said to *decide* the language. This leads to the definition of Turing-decidable (sometimes just called decidable). We call a language *Turing-decidable* if some Turing machine decides it. Note that every decidable language is Turing-recognizable. However there are languages that are Turing-recognizable but not Turing-decidable.

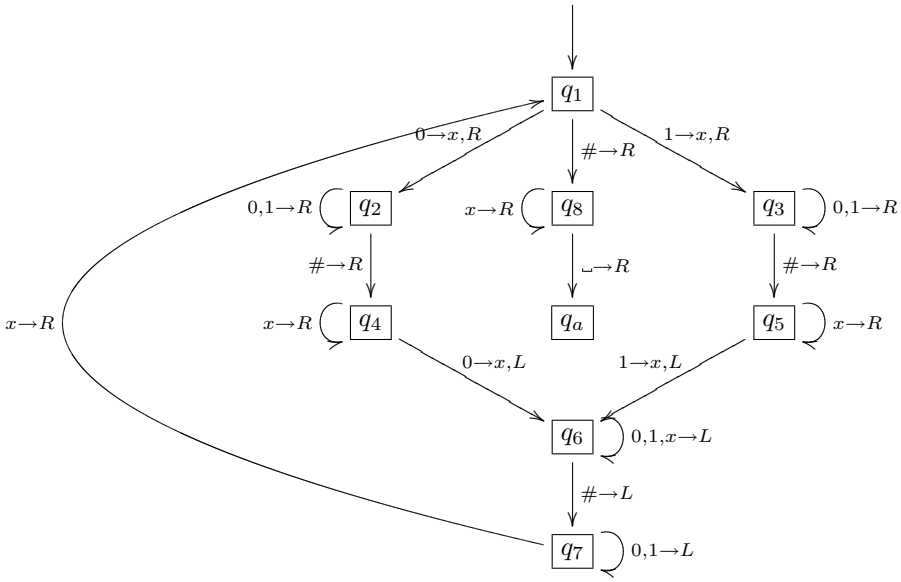
I. EXAMPLE OF A TURING MACHINE

Lets describe a Turing machine which decides the language $L = \{w\#w | s \in \{0,1\}^*\}$. First lets understand how a Turing machine might solve this. Basically what the machine will do to get to an accepting configuration is read the first symbol, then cross it out, and then, remember what that symbol was, move until it reaches the #, and then checks to see if the symbols match, and if they do cross it off and go back to the symbol after the one we crossed out, and then continue in this same manner, except going to the first non-crossed off symbol in the transversals. Finally we need to make sure that we accept if this is all that the input consists of.

Okay here is a formal description of a Turing machine which does this

1. $Q = \{q_1, q_2, \ldots q_8, q_a, q_r\}$
2. $\Sigma = \{0, 1, \#\}$ and $\Gamma = \{0, 1, x, \#, \textvisiblespace\}$.
3. The start state is q_1. The accept state is q_a. The reject state is q_r.

And now we need to specify the transition function. We do this with a state diagram.



Now there are some points about this transition function which we should point out. First of all where is the reject state q_r? Well we have obmitted it. Indeed we have followed a convention where every allowed transition which is not

labeled is a transition to q_r. Further note that all rules should be of the form $s \to a, d$ where $s, a \in \Gamma$ and $d \in \{L, R\}$. But some of these don't have an a, i.e. they are of the form $s \to d$. This is shorthand for the fact that the same symbol which is on the tape is being written onto the tape, i.e. that symbol is not changed. Finally we often write multiple elements which lead to the transition separated via commas.

Let's run through how this machine acts on an input. Lets do 010#010.

$q_1010\#010$
$xq_210\#010$
$x1q_20\#010$
$x10q_2\#010$
$x10\#q_4010$
$x10q_6\#x10$
$x1q_70\#x10$
$xq_710\#x10$
$q_7x10\#x10$
$xq_110\#x10$
$xxq_30\#x10$
$xx0q_3\#x10$
$xx0\#q_5x10$
$xx0\#xq_510$
$xx0\#q_6xx0$
$xx0q_6\#xx0$
$xxq_70\#xx0$
$xq_7x0\#xx0$
$xxq_10\#xx0$
$xxxq_2\#xx0$
$xxx\#q_4xx0$
$xxx\#xq_4x0$
$xxx\#xq_6xx$
$xxx\#q_6xxx$
$xxxq_6\#xxx$
$xxq_7x\#xxx$
$xxxq_1\#xxx$
$xxx\#q_8xxx$
$xxx\#xq_8xx$
$xxx\#xxq_8x$
$xxx\#xxxq_8$
$xxx\#xxx\textvisiblespace q_a$

Wowah. That was a bit long.