CSCI 510 **Models of Computation**

HW3: You Oughta Know

Solution Key

Include your *full name*, *CS login*, and the problem number(s) on each piece of paper you hand in, and please staple your pages together before handing in.

While collaboration is encouraged in this class, please remember not to take away notes from collaboration sessions other than your scheduled lab section.

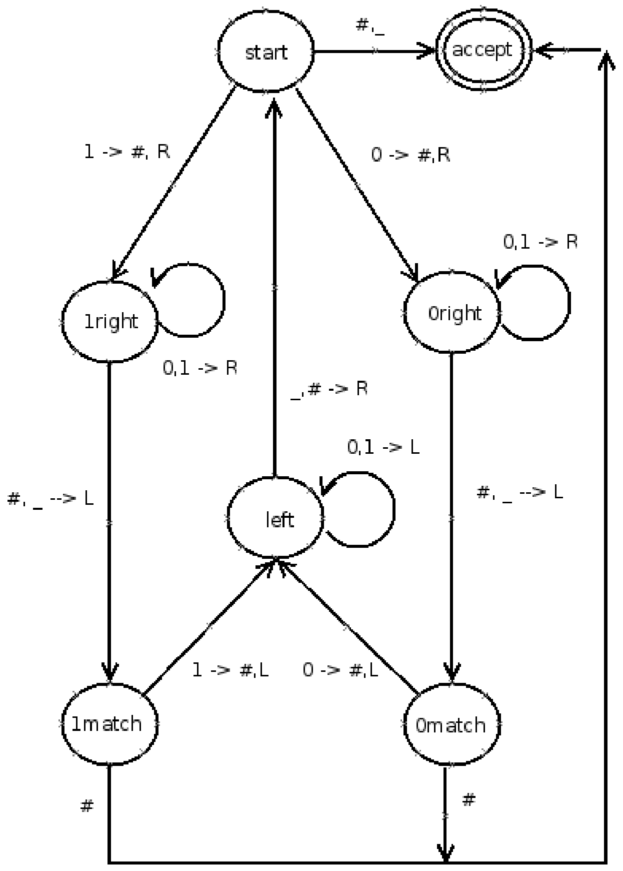
Problem 1

A palindrome is a string W with the property that $W = W^R$, where W^R is the *reverse* of W. For example, 010 and 0110 are both palindromes.

Describe in detail a Turing machine that recognizes the language of palindromes where the input alphabet is $\{0, 1\}$. The description of your Turing Machine should be in the form of a diagram of the states and transitions. Be sure to specify the tape alphabet.

Solution

In order to recognize the language of palindromes, we can use a TM that has one special tape character in addition to its standard alphabet. We call this tape character #. The TM should do the following:



This TM takes $\theta(n^2)$ time. This is not the most efficient Turing machine to test for palindromes — the number of steps can be approximately halved by reading the next character while you're down at the other end of the input making sure that the current character matches with the rightmost unmarked character.

The following questions are lab problems. Please remember to prepare a solution for your assigned problem before going to your lab section.

Lab Problem 1

Consider the Boolean function

$$f(x_1, x_2, \dots, x_n) = x_1 \wedge x_2 \wedge \cdots \wedge x_n.$$

The number of gates required to construct a circuit that computes this function for a fixed n is dependent on the number of inputs allowed to each AND gate in the circuit.

a. Show that the number of 2-input AND gates required to construct a circuit that computes f is $n-1$ and the depth of such a circuit is at least $\lceil \log_2 n \rceil$.

b. Find a similar formula for the size and depth of a circuit constructed using r-input AND gates for an arbitrary (but fixed) r.

Solution

a. With 2-input AND gates, we can only reduce the number of wires by one with each gate, so there must be $n-1$ gates to get the number of wires down to one. For the depth, we can only reduce the number of wires by a factor of 2 (by pairing the remaining wires and ANDing those) at each level. Therefore, we need at least $\log_2 n$ levels to complete the circuit. Moreover, by using exactly this construction, we can construct the circuit with a depth of $\lceil \log_2 n \rceil$.

b. Number of gates will be $(n-1)/(r-1)$, depth will be $\lceil \log_r n \rceil$. This is because, with each gate, we combine r wires into 1, so we reducing the number of wires by $r-1$. Since we only have one output, the total number of wires needs to be reducted to 1, so we need to combine wires such that we have eliminated $n-1$ wires. To do this we need $(n-1)/(r-1)$ gates.

   For the depth, at each level, we can reduce the number of wires by a factor of r by sending every wire through a gate, but we cannot do better than that (since otherwise we would send a wire through multiple gates, which we cannot do on a single level). Therefore, we need a depth of at least $\log_r n$, and we can do it in exactly that depth.

Lab Problem 2

A standard Turing machine has a single tape that has a left end but extends infinitely to the right. In this problem you will examine two variants of the standard Turing machine and prove that they are all equivalent.

a. Show that a Turing machine with a single doubly-infinite tape (that is, a tape that extends infinitely in both directions) can be simulated by a standard TM and vice versa. This shows that the two are equivalent models of computation.

b. Now consider a Turing machine with multiple tapes, each of which has a left end but extends infinitely to the right. Each tape has its own head, and the heads of different tapes can be in different places at the same time. Prove that, for an arbitrary but fixed n, a standard TM can simulate an n-tape TM and vice versa.

Solution

a. Let T_{DE} be a TM that has a tape that extends without limit to the left and right from the initial head position and let T_{SE} be a TM that has a tape that is bounded on the left and extends without limit to the right.

   Given a T_{SE} we can simulate it with a T_{DE} that simulates T_{SE} because the latter machine does not attempt to move left from the leftmost position. To ensure that T_{SE} does not move left from this position, T_{DE} can write a special symbol to the left of the initial head position and never allow a move beyond it.

b. To simulate T_{DE} with an instance of T_{SE}, let the latter have a two-track tape. Only one track is used at a time. Let the upper track contain the initial contents of the single track tape of T_{DE} and let the initial head position be in the leftmost cell of the first track. When the head of T_{DE} is on or to the right of the initial head position, the head of T_{SE} reads and writes to the upper track. Otherwise it reads and writes to the lower track.

Lab Problem 3

In class you saw two types of bit shifting:

- Logical shifting, where the bits are shifted and then the open slots are filled with 0s. So if we logically shift `10101110` left by two bits, we get `10111000`.

- Cyclic shifting, where the bits that are shifted off one end wrap around. So, if we cyclically shift `10101110` left by two bits, we get `10111010`.

a. Reduce logical shifting to cyclic shifting. That is, assuming you have a machine M that can perform cyclic bit shifting, construct a machine, using M as a subroutine, that performs logical bit shifting.

b. Reduce cyclic shifting to logical shifting.

Note: You may not explicitly inspect or change individual bits for either of these reductions.

Problem

a. We will assume that we are shifting left. If we are shifting right, the same reduction works, but the extra bits are appended to the beginning instead of the end. Assume we have a machine M that, on input $\langle w, k \rangle$ will cyclically shift w by k bits. So, on input $\langle w, k \rangle$ with $|w| = n$, first append n 0s to the end of w (call this $u = w \circ 0^n$). Then get $u' = M(u, k)$. Finally, output the n most significant bits of u'.

   This is correct because it M will shift 0s into the place of the lower order bits of w when it shifts u, so by taking only the n most significant bits of u', get the shifted w with 0s for the lowest order k bits, which is precisely a logical shift.

b. Assume we have a machine M that, on input $\langle w, k \rangle$ logically shifts w by k bits. On input $\langle w, k \rangle$, we want to cyclically shift w by k bits. if $n = |w|$, we will assume $k < n$, since otherwise we could let $k = k \bmod n$ and have the same problem. In this case, we construct $u = w \circ w$ (in other words, concatenate w with itself, so u is two copies of w together). Now let $u' = M(u, k)$. Finally, we take the n most significant bits of u' and output that.

   This is correct because the second half of u is another copy of w, so when M shifts u, it will shift higher order bits of w into spots $n-k$ to n instead of 0s, which is precisely a cyclic shift.