Directions

In each of the following problems, you are asked to construct a Turing machine. These will (optimistically) be graded by a human, so for the benefit of a human reader, please supply plentiful comments, commenting almost every line, as I did in the classroom examples, so that one decipher your code. Realistically speaking, as with any computer program, your Turing machine code will contain bugs, so you might want to check these programs against JFlap or the emulator from your textbook. I am assuming you will want to type these programs rather than write them on the assignment sheet, so I didn’t leave space for the solutions. Please just staple your solutions behind the assignment sheet, or if you don’t do that, clearly label the top page Turing Machine Homework, and staple all your pages together. These problems are increasing in difficulty and each one provides techniques you need for the subsequent ones, so work them in order, and don’t give up till you are done. Problems 1, 2, and 3 are worth 2 points each and problem 4 is worth 4 points, total 10.

1. Concatenation. Write a TM with input alphabet $\Sigma = \{a, b\}$, that concatenates its inputs. That is, when started with a string of the form $\alpha \# \beta$, it moves all the characters of $\beta$ one square to the left, erasing the $\#$ and leaving $\alpha \beta$ on the tape. For example, given the input $aabab \# abab$ it produces $aabababab$.

   Hint: You can accomplish this by swapping the $\#$ with the character to the right of it, repeating until you hit a blank.

2. Copy. This machine takes an input string over $\{a, b\}$, puts a $\#$ after it, and then makes a copy of the input to the right of $\#$. For example, if the input is $abaab$, when the machine halts it has $abaab\#abaab$ on its tape.

   Hint: You will have to leave a marker as in one of the classroom examples to keep your place. You will also have to “remember” what character you are “carrying” from the front to the back. You would use a variable for that in Java, but in Turing machine language, you use a state. That is, you will have two different states for moving right, with identical instructions until you get to the place where you have to “dump” the remembered character. One of them dumps an $a$, and the other one dumps a $b$. When you have to remember things, which would be the values of variables in Java, it will help to give your states names that mention that variable. For instance, I could have a state $3a$ and a state $3b$, one of which is “remembering” $a$ and the other one “remembers” $b$. 


If, in a more complex program, I had two “variables” to code as states, I would use \(3aa, 3ab, 3bb, \) and \(3ba\) as states.

3. Reverse. Write a TM that takes a string over the alphabet \(\{a, b\}\) and reverses it. For example, input \(aabba\) becomes \(abbaa\). Hint: Swap the first and last character, then the second and next-to-last, and so on, using a technique like you used in problem 2.

4. Addition in binary. This Turing machine works with two inputs made of 0’s and 1’s, separated by a +. The inputs represent the bits of binary numbers, in order from lowest to highest. (Thus, the reverse of how you would write them on paper. For example 10 will be 0101, not 1010.) When the machine terminates, the sum of the two numbers will be on the tape in the same form. For example, started with input 0101+01, representing the problem we would usually write as 10 + 2, it produces 0011, the reversed binary representation of 12. For simplicity you may assume that the second input is not longer than the first input.

\[\text{Hint: You need to implement the following algorithm:}\]

```c
done = false;
for(j=0; !done ; j++)
{ if(y[j] == blank)
   other = 1;
else
   other = y[j];
if(x[j] == '+')
{ done = true;
  first = 0;
}
else
  first = x[j];
if(carry == 0)
{ if(first == 1 && other == 1)
   { carry = 1;
     next = 0;
   }
else if(first== 0 && other] == 0)
  next = 0;
else
  next = 1;
}
else // carry = 1
{ if(first == 1 && other = 1)
   { next = 1;
     carry = 1;
   }

2```
else if(first == 1 || other == 1)
    { next = 0;
      carry = 1;
    }
else
    { carry = 0;
      next = 1;
    }

x[j] = next;

Because of the assumption that the second argument is not longer than the first, the answer can just be written over the first input. (Otherwise you would first have to move the second input far enough to the right to be sure there was room.) You will need to use a marker to mark the position (j in the pseudocode); and then you will have to “remember” what was there before you placed the marker. That, of course, is first in the pseudocode. Then you move right, changing states when you pass the + sign, to the first nonblank character. Then, remember that character as other, and replace it by a blank, and move left again, to replace the marker by the correct value of next. Move right (that corresponds to j ++), remember the character you see as the new value of first, replace it with a marker, and repeat. Thus, there are three variables to remember, each of which can have two values. You might name your states things like 7-010, which you think of as state 7, remembering first = 0, other = 1, carry = 0. When you finish with the part of the code implementing the algorithm above, be sure to erase the + sign with a blank, not with a 0.