The Computational Beauty of Nature – Spring 2020

Lab 5: Turing Machines

In this lab you will experiment with Turing machines. Write down your answers to each exercise as you go.

1. Go to our class web page under *Links* and click on *Turing Machine simulator.* A file named **mystery-TMs.txt** containing six mystery Turing machines is available under the *Labs* section of the web page. Open this file in a new browser window. Cut and paste the rules for TM1 into the simulator, then run the machine on each of the following inputs and record the output produced in each case. If you like, you can try other strings of **0**'s and **1**'s as well. Describe in just a few words the operation that this Turing machine performs on its input.

TM1 Input	TM1 Output	TM1 Function:
1. **011001**	1.	
2. **111**	2.	
3. **00**	3.	
4. **0101**	4.	
5. **1101**	5.	
6. **1**	6.	

2. Mystery machine TM2 takes a single string of **1**'s (containing no **0**'s) as input. Try the example inputs below, and a few more of your own to fill out the table. If we interpret each string as representing a number written in unary notation (e.g. **11** = 2, **111** = 3, **1111** = 4, etc.), what arithmetic function does this machine compute?

TM2 Input	TM2 Output	TM2 Function:
1. **111**	1.	
2. **1**	2.	
3. **11111**	3.	
4.	4.	
5.	5.	
6.	6.	

3. Machines TM3 and TM4 each take two strings as input, separated by a single **#** symbol, where each string is a block of **1**'s containing no **0**'s. What functions do these machines compute?

TM3 Input	TM3 Output	TM3 Function:
1. **11#111**	1.	
2. **1111#1**	2.	
3. **1#1**	3.	
4.	4.	
5.	5.	
6.	6.	

TM4 Input	TM4 Output	TM4 Function:
1. **11111#11**	1.	
2. **111#11**	2.	
3. **111#111**	3.	
4.	4.	
5.	5.	
6.	6.	

4. TM5 takes a single block of **1**'s (containing no **0**'s) as input and outputs "yes" or "no". What classification function is this machine performing on its input?

TM5 Input	TM5 Output	TM5 Function:
1. **11111**	1.	
2. **1111**	2.	
3. **111**	3.	
4.	4.	
5.	5.	
6.	6.	

5. What happens if TM5's input string contains **0**'s? Can you fix the machine so that it simply ignores any **0**'s it encounters during its computation? Hint: add two new rules, one for the symbol **0** in state *s1*, and one for the symbol **0** in state *s2*.

6. Modify the TM5 machine so that instead of printing "yes" or "no", it prints "odd" or "even", respectively. Hint: in order to print the word "even", you'll need to use a couple of extra states.

7. TM6 takes an arbitrary string of **0**'s and **1**'s as input, and outputs "yes" or "no". What classification function is this machine performing on its input?

TM6 Input	TM6 Output	TM6 Function:
1. **10011001**	1.	
2. **11011001**	2.	
3. **1111**	3.	
4.	4.	
5.	5.	
6.	6.	

8. Now open the file **other-TMs.txt** (available under *Labs* on the class web page), which contains several other Turing machines. Modify the Inverter machine so that the read/write head returns to the beginning of the string right before halting. Hint: add a new state *s2* that moves the head leftward until it encounters a blank.

9. Create a new Turing machine that, when started on a blank tape, simply prints out the word "ciao" and halts.

10. Modify your machine so that it prints "ciao" over and over, in an endless loop, with a space between each "ciao".

11. Try out the unary Multiplication machine in **other-TMs.txt**. This machine takes as input two strings of **1**'s (containing no **0**'s), separated by a single **#** symbol, and computes their product. For example, try the input **111#1111**. To increase the speed of the machine you can click the "Run at full speed" checkbox. Also try the Divisor machine, which takes two numbers in unary notation separated by a **#** and determines whether the first number divides evenly (with no remainder left over) into the second number. For example, try **111#111111**. Finally, try out the Primality tester, which takes a single number in unary notation and determines whether it is prime (**p**) or composite (**c**).

12. Create a Turing machine that moves to the right, scanning over **0**'s and **1**'s until it finds an **x**. It should then erase everything on the tape between that **x** and the next **x** to the right. It should halt when it gets to the second **x**. The **x**'s themselves should not be erased. You may assume that the only symbols on the tape will be **0**'s, **1**'s, **x**'s, and blanks. You can do this with a fairly simple machine that uses only two states. (Note that if the machine is started on a tape that does not contain two **x**'s to the right of the machine's starting position, then the machine will never halt.)

13. Create a Turing machine that takes as input a single string containing any combination of **0**'s, **1**'s, and **x**'s. The machine should scan the string until it finds two *consecutive* **x**'s, and then immediately halt. For example, if the input string is **001x00xx1**, the machine should halt immediately after scanning the rightmost **x** in the string.

14. Now modify your machine so that it scans until it finds *three* **x**'s in a row. When it finds a group of three consecutive **x**'s, it should halt. For an added touch, have the machine move back to the first of the three **x**'s and halt there. Make sure your machine works for any combination of **0**'s, **1**'s, and **x**'s, such as **0011x01xx1011x111xx00xxx00**.

15. Try out the machine Binary Add1 in **other-TMs.txt**. This machine takes a string of **0**'s and **1**'s as input, representing a number in binary (base 2) notation, and adds 1 to the number, performing any carries as needed. For example, **11** (the binary representation of 3) is transformed to **100** (the binary representation of 4). Try a few other inputs as well. We can easily transform this machine into a binary "counter" by modifying the last rule so that, instead of halting after adding 1, it returns to the starting state *s1*. Modify the rule in this way and then observe the behavior of the machine on the input **0**.

16. The machine Decimal Add1 is just like Binary Add1, except that it adds 1 to a number written in ordinary decimal (base 10) notation. For example, **3** gets transformed to **4**, and **99** gets transformed to **100**. Try out a few other inputs as well. We can transform this machine into a decimal "counter" in the same way as before, by having the machine return to state *s1* instead of halting. Modify the machine in this way and then run it on the input **0**.

17. It is easy to create a Turing machine that, when started on a blank input tape, will print an infinite number of **1**'s (in fact, this can be done with just a single rule). A more interesting question is: for a machine that *does* eventually halt, what is the maximum number of **1**'s it can possibly print on its tape? This depends on how many states the machine has. But machines with just a few states can print a surprisingly large number of **1**'s, and run for a very long time, before eventually halting. Such a machine is called a *busy beaver*. The file **other-TMs.txt** contains an example of a 5-state busy beaver machine. Although this machine is relatively simple (only 5 states), it generates very complex behavior. Try it out, starting on blank tape, to see just how many steps it takes to halt.

18. Construct a Turing machine to do the following. Assume that the machine is started on a tape that contains nothing but a string of **$**'s. The machine is started on the left end of this string. The purpose of the machine is to multiply the length of the string by 3. For example, if given a string of seven **$**'s, it should halt with twenty-one **$**'s on the tape. If it is started on a string that contains just one **$**, it should halt with three **$**'s on the tape. Here is one possible way (but not the only way!) that the machine might accomplish this task: Change one of the **$**'s to an **x**, then go to the end of the string and write two more **x**'s. Go back and process the next **$** in the same way. Continue until all the **$**'s have been processed. Then change all the **x**'s to **$**'s.