Busy Beaver Lab

Part l

1. Example1 states:

q0, saw\_#, saw\_##, qa

Example2 states:

q0, End, FindDelimiter0, Check0, FindLeftmost, FindDelimeter1, Check1, FindNext, qa

1. print("\nState: {}").format(state)

print("Left side: {}").format(left\_hand\_side)

print("Symbol: {}").format(symbol)

print("Right side: {}\n").format(right\_hand\_side)

machine.debug(w)

1. What computation each turing machine performs

1: Accepts only the string “##”, it first checks that the first character of the string is a #, and moves to the right if so, then it check that the second character is a # and moves to the right if so. Finally, it checks that there are not more characters and accepts the string.

2: Accepts only the string “#” and binary strings that are the same before the # compared to after the #. To perform this computation, the machine checks the first character of the left hand side (overrides it with an “X”) and then looks for the same value after the #, then it comes back to the very beginning of the string and marks the next value of the left hand side that is not “X”. The machine repeats the process until it marks the whole left-hand side and then expects to find only “X” on the right-hand side as well, otherwise the string is rejected.

Part lll

1. Generators are advantageous in simulating Turing machines because they are scalable. Since we do not know how many states the Turing machine is going to go through before it stops, we can’t loop on it using a for, but we can using a generator because this one will only keep looping if it receives the proper indication, and you also can’t iterate over it more than once.
2. 4
3. 2 card Busy Beaver:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| A0 | A1 | B0 | B1 | Ones written | Steps |
| 1RB | 1LB | 1LA | 1RH | 4 | 6 |
| 1RB | 1RH | 1LB | 1LA | 3 | 6 |
| 1RB | 0LB | 1LA | 1RH | 3 | 6 |

1. (4n+4)^2(n) -> Number of turing machines of n states. My turing machine is likely new.