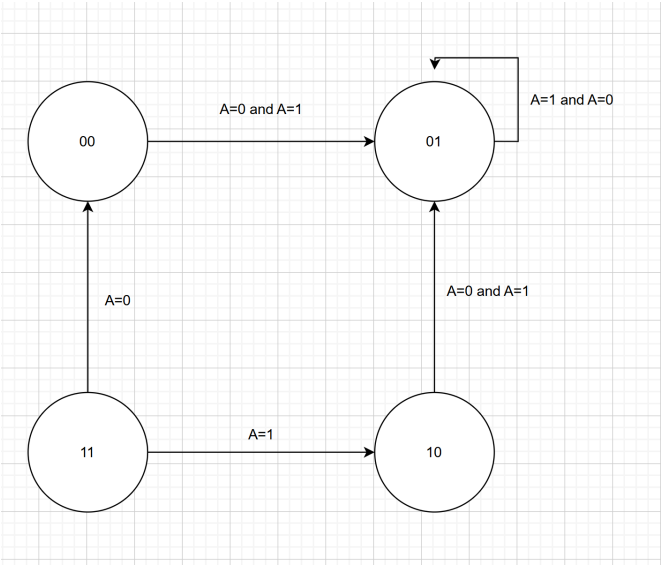
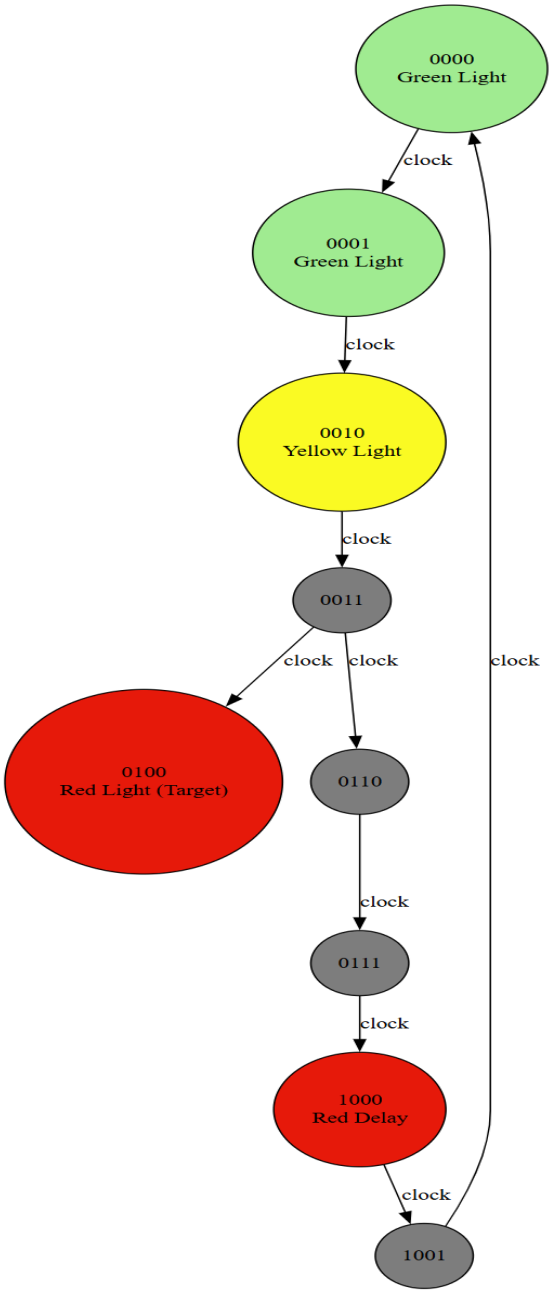
Aidan Chin SPIREID: 3380332 & Donivan Saville SPIREID: 33889341

Part A (Reachability by Graph Traversal): [25 points]
In this part of the lab, you will explore how to check reachability using random execution (with Verilog testbench)
and using explicit reachability by graph traversal (on paper).
A.1) Draw the state transition graph for ex1 and stoplight1. You can use any method that you prefer to determine which state transitions are possible (pen and paper, simulation, etc). Explain your approach for determining which transitions are possible.



This is the state transition graph for the first file ex1 and this was simple enough to be completed by hand and I used draw.io to create the diagram above

Aidan Chin SPIREID: 3380332 & Donivan Saville SPIREID: 33889341



A.2) Use the state transition graphs to determine whether the target states are reachable in ex1 and stoplight1. If so,
give the sequence of states visited along the shortest path to the target state. Shortest path means the path with
the fewest transitions, or equivalently, fewest clock cycles. Describe your approach.

- By looking at the state transition graph for ex1 you can see that the target state of 10 is unreachable when starting from state 00. The only way to reach 11 would be starting from 11
- By looking at the state transition graph for stoplight1 you can see the target state of 0100 is reachable when starting from state 0000. This has to happen under the right conditions however and it will reach state 0100 from state 0011 assuming everything is correct. When running a verilog testbench with random inputs in each cycle it reached the target state after 45 cycles from state 0011.

A.3) Include a table showing how many states of ex1 and stoplight1 are reachable within i transitions, for all values
of i from 1 to 20. “Reachable within i transitions” is cumulative, meaning that any state reachable within j
transitions will also be listed as reachable within j+1 transitions, regardless of whether it is possible to reach
that state on transition j+1 itself.

l (transitions)	Starting State	Next State	Total Reachable States
1	00	01	2
2	01	01 (Loop)	2
3	01	01 (Loop)	2
4	01	01 (Loop)	2
5	01	01 (Loop)	2
6	01	01 (Loop)	2
7	01	01 (Loop)	2
8	01	01 (Loop)	2
9	01	01 (Loop)	2
10	01	01 (Loop)	2
11	01	01 (Loop)	2
12	01	01 (Loop)	2
13	01	01 (Loop)	2

14	01	01 (Loop)	2
15	01	01 (Loop)	2
16	01	01 (Loop)	2
17	01	01 (Loop)	2
18	01	01 (Loop)	2
19	01	01 (Loop)	2
20	01	01 (Loop)	2

I (transitions)	Starting State	Next State	Total Reachable States
1	0000	0001	2
2	0001	0010	3
3	0010	0011	4
4	0011	0110	5
5	0110	0111	6
6	0111	1000	7
7	1000	1001	8
8	1001	0000	9
9	0000	0001	9
10	0001	0010	9
11	0010	0011	9
12	0011	0110	9
13	0110	0111	9
14	0111	1000	9
15	1000	1001	9
16	1001	0000	9
17	0000	0001	9

18	0001	0010	9
19	0010	0011	9
20	0011	0110	9

A.4) For ex1, ex2, stoplight1, and stoplight2, write a Verilog testbench to initialize the state to 0, apply random inputs in each cycle, and notify you if/when it reaches the target state. If the target state is not reached within 100,000 cycles, the result can be reported as “timed out”. If the target state is reached, report how many cycles were simulated before first reaching it. See included file demo.v for an example of a Verilog testbench.

```
 33500: cycle 34, state: 0001, Ped: 0, SigG: 0, SigY: 0, SigR: 1
 34500: cycle 35, state: 0010, Ped: 0, SigG: 0, SigY: 0, SigR: 1
 35500: cycle 36, state: 0011, Ped: 1, SigG: 1, SigY: 0, SigR: 0
 36500: cycle 37, state: 0110, Ped: 0, SigG: 1, SigY: 0, SigR: 0
 37500: cycle 38, state: 0111, Ped: 1, SigG: 1, SigY: 0, SigR: 0
 38500: cycle 39, state: 1000, Ped: 1, SigG: 0, SigY: 1, SigR: 0
 39500: cycle 40, state: 1001, Ped: 1, SigG: 0, SigY: 1, SigR: 0
 40500: cycle 41, state: 0000, Ped: 1, SigG: 0, SigY: 0, SigR: 1
 41500: cycle 42, state: 0001, Ped: 1, SigG: 0, SigY: 0, SigR: 1
 42500: cycle 43, state: 0010, Ped: 0, SigG: 0, SigY: 0, SigR: 1
 43500: cycle 44, state: 0011, Ped: 0, SigG: 1, SigY: 0, SigR: 0
 44500: cycle 45, state: 0100, Ped: 1, SigG: 1, SigY: 0, SigR: 0
Target state (0100) reached at cycle 45
stoplight1_tb.v:38: $finish called at 44500 (1ps)
doniv@DESKTOP-J3AI1K5:~/lab1_files$
```

- Ex1.v and ex2.v both timed out after 100,000 cycles
-

```
1339500: cycle 1340, state: 10010, Ped: 1, SigG: 1, SigY: 0, SigR: 0
1340500: cycle 1341, state: 10011, Ped: 1, SigG: 1, SigY: 0, SigR: 0
1341500: cycle 1342, state: 00000, Ped: 0, SigG: 0, SigY: 0, SigR: 1
1342500: cycle 1343, state: 00001, Ped: 0, SigG: 0, SigY: 0, SigR: 1
1343500: cycle 1344, state: 00010, Ped: 0, SigG: 0, SigY: 0, SigR: 1
1344500: cycle 1345, state: 00011, Ped: 1, SigG: 1, SigY: 0, SigR: 0
1345500: cycle 1346, state: 00110, Ped: 0, SigG: 1, SigY: 0, SigR: 0
1346500: cycle 1347, state: 00111, Ped: 1, SigG: 1, SigY: 0, SigR: 0
1347500: cycle 1348, state: 10000, Ped: 1, SigG: 1, SigY: 0, SigR: 0
1348500: cycle 1349, state: 10001, Ped: 0, SigG: 1, SigY: 0, SigR: 0
1349500: cycle 1350, state: 10010, Ped: 1, SigG: 1, SigY: 0, SigR: 0
1350500: cycle 1351, state: 10011, Ped: 0, SigG: 1, SigY: 0, SigR: 0
1351500: cycle 1352, state: 10100, Ped: 1, SigG: 1, SigY: 0, SigR: 0
1352500: cycle 1353, state: 10101, Ped: 0, SigG: 1, SigY: 0, SigR: 0
1353500: cycle 1354, state: 10110, Ped: 1, SigG: 1, SigY: 0, SigR: 0
1354500: cycle 1355, state: 10111, Ped: 0, SigG: 1, SigY: 0, SigR: 0
1355500: cycle 1356, state: 11000, Ped: 1, SigG: 1, SigY: 0, SigR: 0
1356500: cycle 1357, state: 01000, Ped: 1, SigG: 0, SigY: 1, SigR: 0
Target state (01000) reached at cycle 1357
stoplight2_tb.v:40: $finish called at 1356500 (1ps)
doniv@DESKTOP-J3AI1K5:~/lab1_files$
```

```
`timescale 1ns/1ps

module test_stoplight1;
  reg clock;
  reg Ped;
  wire SigG, SigY, SigR;
  integer cycle_count;

  // Instantiate the stoplight module.
  // Port order: (Ped, clock, SigG, SigY, SigR)
  stoplight uut(Ped, clock, SigG, SigY, SigR);

  initial begin
    // Initialize the internal state to 0 using hierarchical references.
    // Make sure that the internal state registers are exactly named S3,
S2, S1, and S0 in stoplight.v.
    {uut.S3, uut.S2, uut.S1, uut.S0} <= 4'b0;
    clock <= 0;
    Ped   <= $random;
    cycle_count = 0;
    // Optionally dump waveforms:
    // $dumpfile("test_stoplight1.vcd");
    // $dumpvars(0, test_stoplight1);
```

```
  end

  // Generate clock: toggles every 0.5 ns.
  always #0.5 clock = ~clock;

  // At each positive clock edge, update Ped, print status, and check for
the target state.
  always @(posedge clock) begin
    cycle_count = cycle_count + 1;

    $display("%6t: cycle %0d, state: %b%b%b%b, Ped: %b, SigG: %b, SigY: %b,
SigR: %b",
             $realtime, cycle_count, uut.S3, uut.S2, uut.S1, uut.S0, Ped,
SigG, SigY, SigR);

    // Check if the target state (0100) is reached.
    if ({uut.S3, uut.S2, uut.S1, uut.S0} == 4'b0100) begin
      $display("Target state (0100) reached at cycle %0d", cycle_count);
      $finish;
    end

    // Update the Ped input with a new random value for the next cycle.
    Ped <= $random;

    // Timeout: If target state isn't reached in 100,000 cycles, end
simulation.
    if (cycle_count == 100000) begin
      $display("Timed out after 100,000 cycles");
      $finish;
    end
  end

endmodule
```

A.5) Explain whether/how your result for stoplight1 A.4 is consistent with your result from A.2. Be specific about what finding in A.4 would have shown that at least one of the results (A.2, A.4, or both) must be wrong.

Aidan Chin SPIREID: 3380332 & Donivan Saville SPIREID: 33889341

- Based on my findings from A.4 from stoplight1 we find that the target state of 0100 is reachable just like how in A.2 we get that the state is reachable from the state table diagram. However the difference we get with A.4 is how many cycles it takes to reach that target state of 0100 which in this case was 45. This could show that A.2 could be wrong because it doesn't show how it gets 0100 in enough detail.

Part B (SAT-Based Symbolic Reachability): [60 points]

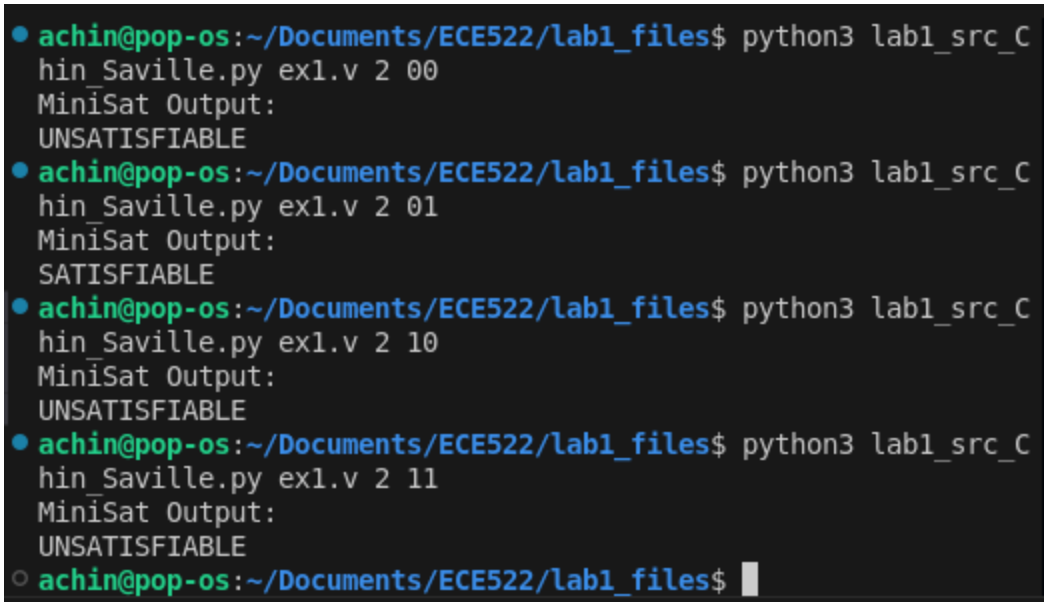
B.1) Use your program to generate the CNF files for ex1 with the transition relation unrolled 2 times. "unrolled 2 times" means there are 2 instances of the transition relation. Be sure to add the initial state of 0, and to enforce
that the next state of each transition relation is equal to the starting state of the subsequent transition relation. Use the SAT solver to check whether state 11 is reachable in 2 transitions (as the 2nd state after the initial). Is your finding consistent with the state transition graph from question A.1? Explain your answer.

```
achin@pop-os:~/Documents/ECE522/lab1_files$ python3 lab1_src_C
hin_Saville.py ex1.v 2 00
MiniSat Output:
UNSATISFIABLE
achin@pop-os:~/Documents/ECE522/lab1_files$ python3 lab1_src_C
hin_Saville.py ex1.v 2 01
MiniSat Output:
SATISFIABLE
achin@pop-os:~/Documents/ECE522/lab1_files$ python3 lab1_src_C
hin_Saville.py ex1.v 2 10
MiniSat Output:
UNSATISFIABLE
achin@pop-os:~/Documents/ECE522/lab1_files$ python3 lab1_src_C
hin_Saville.py ex1.v 2 11
MiniSat Output:
UNSATISFIABLE
achin@pop-os:~/Documents/ECE522/lab1_files$
```

This finding is consistent with the state transition graph because 2 steps from initial state 00, only 1 state is viable being 01

Now that you've checked whether "11" is reachable in 2 cycles, repeat the analysis to check whether the other 3 states (00,01,10) are reachable in 2 cycles, and again explain whether your findings are consistent with question A.1. Include screenshots of results from the SAT solver for each case. Be sure to fix any bugs here before trying the larger examples that follow!

Aidan Chin SPIREID: 3380332 & Donivan Saville SPIREID: 33889341



Again the findings are consistent for the same reason, the only state that can be reached is 01, all others are impossible

B.2) For each design, perform symbolic reachability analysis with the transition relation unrolled 15 times, to check
if the target state can be reached on the 15th transition. In other words, you are checking whether the target state
can be written into the flip-flops on the 15th rising clock edge. Describe your program and any data structures
used. For each design, indicate whether the target state is reachable, and include screenshots of results from the
SAT solver. Measure the SAT solver runtime for each design and provide the runtimes in the report.

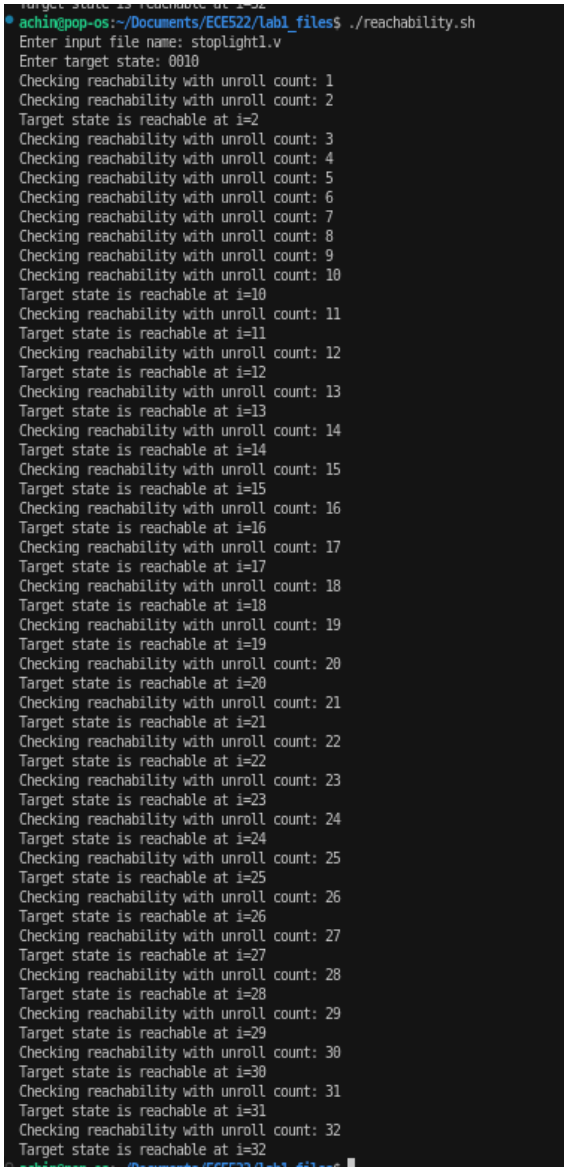
Aidan Chin SPIREID: 3380332 & Donivan Saville SPIREID: 33889341

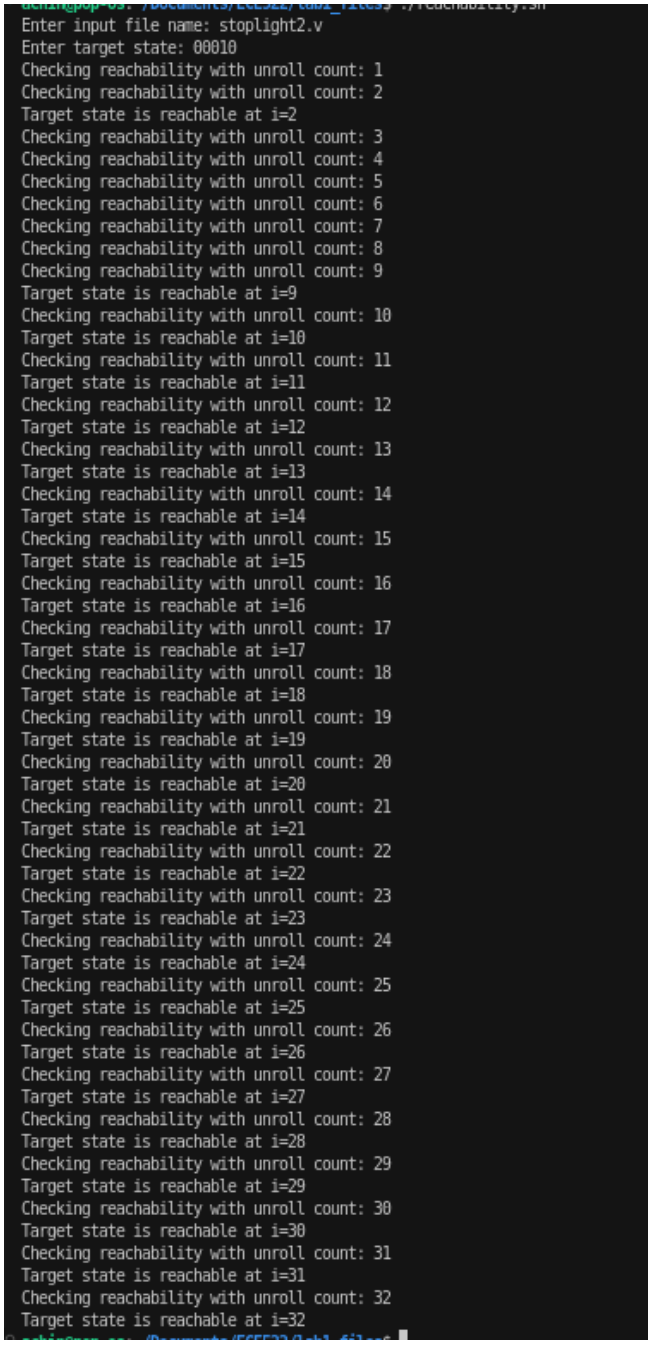
```
achin@pop-os:~/Documents/ECE522/lab1_files$ ^C
achin@pop-os:~/Documents/ECE522/lab1_files$ python3 lab1_src_Chin.py ex1.v 15 11
MiniSat Output:
UNSATISFIABLE
Runtime: 0.001053 seconds
achin@pop-os:~/Documents/ECE522/lab1_files$ python3 lab1_src_Chin.py ex2.v 15 111111111111111
MiniSat Output:
UNSATISFIABLE
Runtime: 0.016345 seconds
achin@pop-os:~/Documents/ECE522/lab1_files$ python3 lab1_src_Chin.py ex3.v 15 10000011000001101101000000000
MiniSat Output:
SATISFIABLE
Runtime: 0.035870 seconds
achin@pop-os:~/Documents/ECE522/lab1_files$ python3 lab1_src_Chin.py ex4.v 15 00000001000000000000000000000000
MiniSat Output:
SATISFIABLE
Runtime: 0.012841 seconds
achin@pop-os:~/Documents/ECE522/lab1_files$ python3 lab1_src_Chin.py ex5.v 15 111011000
MiniSat Output:
UNSATISFIABLE
Runtime: 0.016723 seconds
achin@pop-os:~/Documents/ECE522/lab1_files$ python3 lab1_src_Chin.py stoplight1.v 15 0010
MiniSat Output:
SATISFIABLE
Runtime: 0.007167 seconds
achin@pop-os:~/Documents/ECE522/lab1_files$ python3 lab1_src_Chin.py stoplight2.v 15 00010
MiniSat Output:
SATISFIABLE
Runtime: 0.007001 seconds
achin@pop-os:~/Documents/ECE522/lab1_files$
```

• Upload the source code of your program as a text file when submitting your report. Name the file as
lab1_src_<lastname>.cpp (substitute appropriate extension, use lastname of one team member).
• Upload the dimacs CNF for ex2 with transition relation unrolled 15 times, and appropriate constraints added
for initial and target state. Name the file lab1_ex2_<lastname>.dimacs

its runtime, and the results for each design.
B.3) For the stoplight1, from initial state, check whether the target state is reachable on the ith transition for
i=1,2,...,32. This should be done by invoking your program (and the SAT solver) 32 times while changing the
input argument that specifies the number of times to unroll the transition relation. You can do this using a script.
Report the values of i for which the target state is reachable.

Aidan Chin SPIREID: 3380332 & Donivan Saville SPIREID: 33889341



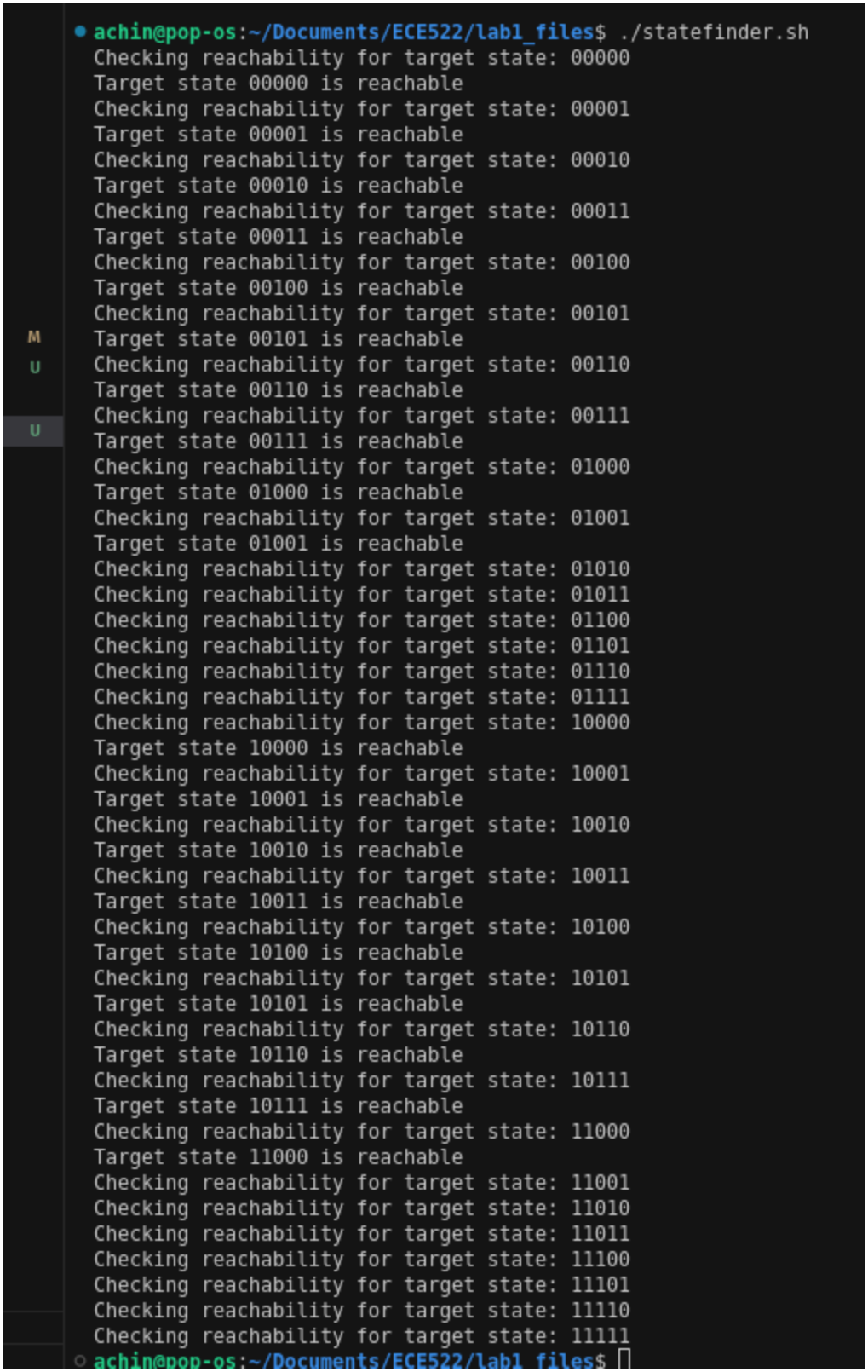


B.4) Repeat the previous question using stoplight2. How does this compare to your finding from random execution
of stoplight2 in A.4?

For stoplight2 the target state becomes reachable after 2 unrolls where as for A.4 the method using the test bench was not reachable until 45 cycles. This can be attributed to the use of random variables instead of taking the fasted method of approach.

B.5) Find all states that stoplight2 could reach on the 17th transition from initial. Explain your approach.

Aidan Chin SPIREID: 3380332 & Donivan Saville SPIREID: 33889341



I just brute forced it with a shell script because my program is pretty efficient

Aidan Chin SPIREID: 3380332 & Donivan Saville SPIREID: 33889341

C.1.

I (transitions)	Starting State	Next State	Total Reachable States
1	000000000	000000100	2
2	000000100	001111000	3
3	001111000	101111000	4
4	101111000	101111000	4
5	101111000	001111100	5
6	001111100	001111000	5
7	001111000	111111100	5
8	111111100	011111000	5
9	011111000	101111100	6
10	101111100	001111000	6
11	001111000	011111000	6
12	011111000	101111100	6
13	101111100	000011010	7
14	000011010	101111000	7
15	101111000	011111000	7
16	011111000	101111000	7
17	101111000	001111100	7
18	001111100	000011010	7
19	000011010	000011010	7
20	000011010	100001110	8

C.2 My approach to completing this was creating a testbench similar to the ones I had created previously for ex1 and ex2 but this time for ex5. Then I ran that testbench for 20 cycles and noted them in the table above to find out how many reachable states were possible in 20 cycles.