ECE 597MB/622 - Spring 2020
Modeling and Verification of Embedded Systems
Lab #1 Reachability Analysis
TA contact info: Peter Stanwicks (pstanwicks@umass.edu)

Objectives of this Lab:

1) To understand symbolic reachability analysis
2) To gain some familiarity with Boolean satisfiability problem
3) Practice programming skills

Sources:

1) **Minisat (SAT solver):** http://minisat.se
2) **Picosat (SAT solver):** http://fmv.jku.at/picosat/
3) **DIMACS CNF introduction:** http://people.sc.fsu.edu/~jburkardt/data/cnf/cnf.html
4) **Icarus Verilog (simulator):** http://iverilog.icarus.com
5) **Further reading about the correspondence of gates and CNF clauses:**
   T. Larrabee, "Test pattern generation using Boolean satisfiability," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 11, no. 1, pp. 4–15, 1992

General lab report instructions:

1) Lab reports should be typed, with name and ID at the beginning of the Lab report.
2) All screenshots and text should be clearly readable.
3) You can work alone or in groups of two.

Introduction:

Sequential systems have memory elements with values that evolve according to inputs and combinational logic. The behavior of sequential systems can be characterized in terms of states and transitions. Reachability analysis deals with determining which states can be reached from a given initial state or initial states. In this lab, the initial state is 0 for each benchmark. Reachability analysis of large embedded systems is a complex task attracting significant research efforts.

The examples used in this lab are provided as sequential Verilog design modules with combinational logic and state updates on each positive clock edge. These designs are posted on Moodle with the lab assignment. For simplicity, the only combinational gate types used in the designs are 2-input AND gates and NOT gates (inverters). Figure 1 shows a graphical depiction of the smallest benchmark (ex1.v) with 4 gates and 2 flip flops. The correspondence between gate types and CNF clauses are also shown below, and more details can be found in the Larrabee paper listed above. For more details, and especially for information about unrolling the transition relation formulas in symbolic reachability, please see reachability slides from class on Moodle.

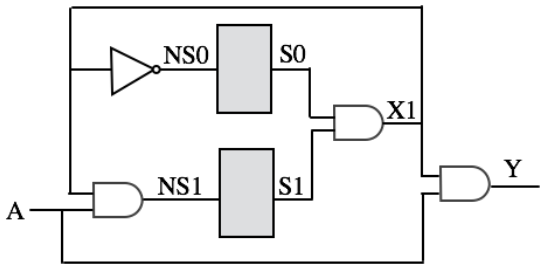
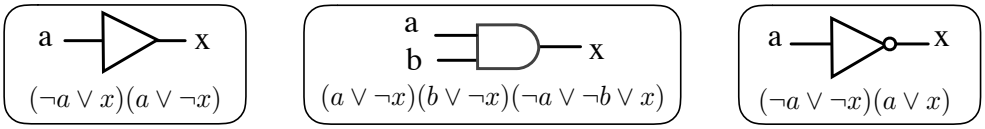


Fig. 1 Drawing of ex1.v

CNF formulas for the gates needed in this lab:



	target state bits (i.e. 31 indicates NS31/S31)																													
design	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ex1																															1	1
ex2																		1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
ex3				0	0	0	0	0	0	0	0	0	1	0	1	1	0	1	1	0	0	0	0	0	1	1	0	0	0	0	0	1
ex4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0
stoplight1																													0	1	0	0
stoplight2																												0	1	0	0	0

Part A [25 points]:

In this part of the lab, you will explore how to check reachability using random execution (with Verilog testbench) and graph traversal (on paper).

(1) Draw legibly the state transition graph for ex1.v and stoplight1.v. Include all states, even if unreachable. You can use any method that you prefer for figuring out which state transitions are possible (pen and paper, simulation, etc), but you must include all transitions.

(2) Using these two state transition graphs, indicate whether the target states are reachable. If so, what is the minimum number of execution steps (clock cycles) needed to get from the initial state to the target state?

(3) If you are given an arbitrary transition relation that maps states to sets of states, describe in pseudo-code the most efficient algorithm you can think of to check whether a target state is reachable from initial state. Your algorithm must terminate upon (1) reaching the target, or (2) finding that the target can never be reached.

(4) For the benchmarks given (ex1, ex2, ex3, ex4, stoplight1, 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 1M cycles, the execution can terminate and the result can be reported as “timed out”. If the target state is reached, report how many cycles of random simulation were performed before reaching it. Include your testbench code with the report.

(5) Are your results from part A.4 consistent with your results from A.2? Describe this. If the results are consistent, give a precise explanation of what results would show them to be incompatible with each other.

Part B (Symbolic Reachability) [75 points]:

This part of the lab solves the same problems as above, but now uses SAT-based symbolic reachability. Essentially, you are writing your own simple verification tool for Verilog that uses SAT solving as a backend. You must write a program that will check reachability by parsing Verilog modules, unrolling the transition relation and converting it to dimacs formatted CNF, and calling a SAT solver. The SAT solver that your program calls (e.g. picosat, minisat, or any other that you choose to install on your computer) will check whether the formula is satisfiable, and the result will reveal whether the target state is reachable in some number of transitions from the initial state. You can write your program in C, C++, Java, Perl or Python (check with TA if you want to use another language). Your program must take two inputs from command line: (1) the name of the Verilog file to read; and (2) the number of times to unroll the transition relation in the symbolic search. Note that the target states listed in the table above are reproduced as a comment at the bottom of each Verilog file so that your program can read it from there.

(1) Use your program to generate the CNF files for ex1 with the transition relation unrolled twice. Be sure to add the initial state of 0 and be sure to enforce that the next state of each transition relation is equal to the starting state of the next transition relation. Use the SAT solver to check whether state 11 is reachable as the 2^{nd} state after the initial one. Is your finding consistent with the state transition graph from question A.1? Explain your answer. Now that you've checked whether "11" is reachable in 2 cycles, modify the example and repeat, 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. Be sure to fix any bugs here before trying the larger examples that follow!

(2) For each of the benchmarks, perform symbolic reachability analysis, with 10 unrollings. Note that you are checking whether the target state is reachable in exactly the 10^{th} state after the initial one. In other words, you are checking whether the next state in the 10^{th} unrolling can be the target state. Describe how you've implemented your program and any data structures used. For each design, indicate whether or not the target state is reachable as the 10^{th} state. How long does the SAT solver take to perform its search for a satisfying assignment?

(3) Compare your results using SAT-based reachability to the results from random simulation. Explain any differences in results and what causes them. Are there any limitations in SAT-based reachability due to only checking whether a state is reachable in the last unrolling and not intermediate ones? How might you improve the symbolic search to avoid this limitation?

(4) Submit the source code of your program as a separate file with your report and include instructions for how to run it for testing.

(5) For the benchmark stoplight1.v, check whether the target state is reachable as the i^{th} state after the initial state 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 unrollings. You can do this using a script. Report the values of i for which the target state is reachable. How does this compare with your results from A.2?

(6) Repeat previous question using benchmark stoplight2.v. How does this compare to your finding from random execution of stoplight2.v in A.4?

(7) Given the number of state bits in the stoplight designs, is it correct to conclude that any states not reachable within 32 unrollings will remain unreachable with more unrollings? Justify your answer.