ECE 597MB/697MB - Spring 2016
ST – Modeling and Verification of Embedded Systems
Lab #1 Reachability Analysis

Objectives of this Lab:

1) To understand explicit and 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) **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 in groups of two. Describe the contributions of each group member in the lab report.

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 set of states such as the initial state or initial states. Reachability analysis of large embedded systems is a complex task attracting significant research efforts.

The four examples used in this lab are be provided in ".bench" format. For simplicity, the only gate types used in the bench files are 2-input AND gates, NOT gates (inverters), and flip flops (memory elements). Figure 1 shows a graphical depiction of the smallest benchmark (ex1.bench) 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 formulas in symbolic reachability, please see reachability slides from class on Moodle.

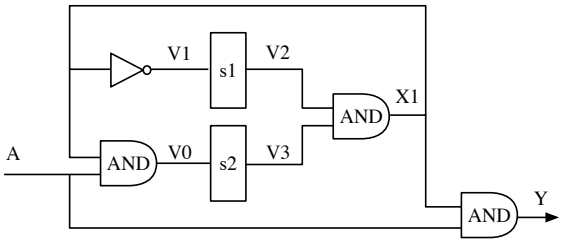


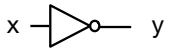
Fig. 1 Drawing of ex1.bench

CNF formulas for AND gates and NOT gates:

a) AND gate

$(a+\neg z)(b+\neg z)(\neg a+\neg b+z)$

b) NOT gate



$(\neg x+\neg y)(x+y)$

Questions:

Part A (Explicit Reachability) [50 points]

Write a program that will read in a bench file and compute all reachable states by repeatedly applying the transition relation (i.e. evaluating combinational logic) from reached states until finding a fixed point (no new reachable states) or exceeding 600 seconds of computation without finding a fixed point. When a fixed point is found, use the set of reached states to check whether a given state is reachable. For each xxx.bench file, a corresponding xxx.state file is provided to indicate what state you are trying to reach. The order of the values in the xxx.state file correspond to the order of appearance of DFFs in the xxx.bench file. You can write your program in C, C++, Java, Perl or Python (check with Hao if you want to use another language), and the program should take xxx.bench and xxx.state as inputs when calling the program. The output of your program for each example should indicate either "reachable," "unreachable," or "timed out."

(1) Draw the state transition graph for ex1.bench by hand.

(2) Describe using pseudo-code your program for explicit reachability. What search algorithm do you use, and what data structure do you use to store the set of reached states? Analyze the complexity of your code using big-O notation.

(3) Apply your program to the four benchmarks given (ex1,ex2,ex3,ex4) and give the result indicating whether the specified state for each is reachable, unreachable, or timed out. If your program returns "reachable" or "unreachable", report the total number of reachable states. If your program times out, report the number of states reached within 600 seconds. We will test your program for correctness using additional small benchmarks (that will not cause a time out).

(4) Include your code with the report, and be sure to provide instruction on how to run the code for testing.

Part B (Symbolic Reachability) [50 points + 10 extra credit]

This part of the lab solves the same problems as above, but now uses SAT-based symbolic reachability instead of explicit reachability. The input of your program is again the xxx.bench file and the xxx.state file, but now an additional input should specify the number of times to unroll the transition relation formula. Your program should produce as output a dimacs CNF file, which is the input format for the SAT solver (i.e. minisat or picosat).

(1) Generate the CNF file for ex1, with the transition relation unrolled twice. Be sure to add the initial condition of "00" to the first transition relation formula, and be sure to enforce that the outputs of the first unrolled formula are equal to the inputs of the second unrolling. Now, use the SAT solver four times to check whether

each possible state (00,01,10,11) is reachable as the 2nd state after the initial one. Note that you are ignoring the xxx.state file here, and will directly modify the CNF formula each time to indicate what state you are trying to reach (if you wish, you can create your own xxx.state files for this instead of modifying by hand). Is your finding for reachable states consistent with the state transition graph from question A.1? If so, explain this. Be sure to fix any bugs here before trying the larger examples that follow! For the states that are reachable as the 2nd state, the satisfying assignment provided by the SAT solver provides values that show what state occurs between the initial state and the second state. Report that intermediate state as well.

(2) For each of the benchmarks, repeat your explicit reachability analysis using symbolic reachability, with 10 unrollings. Note that we are checking whether the designated state is reachable in exactly the 10th state after the initial one. Describe your implementation. For each result, indicate whether or not the state in the xxx.state file is reachable in the 10th state. How long does the SAT solver take to perform its search for a satisfying assignment? We will test this your code with additional xxx.state files, so please include instructions. For the SAT solver minisat, an example of how to run it is shown below. Feel free to try any other solver that you like, as all should accept the input format.

Command : `./minisat file1.cnf file2.txt`

file1.cnf is the input file for minisat and **file2.txt** is the solution generated by minisat.

(3) Compare your runtimes to those from explicit reachability. Are there any limitations related 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) Attach your code in as an appendix in the report, and include instruction for how to run your program for testing.

(5) Extra Credit: For the benchmark ex4, check whether the ex4.state is reachable as the ith state after the initial state for i=1,2,…,20. Rerun this experiment using 2 different SAT solvers. Plot the runtime of each SAT solver on the y-axis against i on the x-axis. Indicate for which values of i the state is reachable.