系統晶片驗證 (SoC Verification)

109 學年下學期 電機系電子所選修課程 943 U0250

Final Project [Property Directed Reachability]

(Due: 9:00pm, Thursday, July 01, 2021)

0. Objectives

1. Learning how to improve a baseline "property directed reachability (PDR)" model checker using the algorithms in the lecture note.

2. Compare your results with the built-in model checker (V3) using testcases provided in this project.

1. Problem Description

In this final project, we provide a baseline PDR model checker in the V3 framework. Although it can successfully prove or disprove some simple properties, it suffers from runtime and/or memory explosion due to its overly simplicity in implementation. In this project, you are asked to improve this PDR algorithm by our hints or your own idea(s).

The similar platform, V3, as in HW #1 & #3 & #5 is provided, where a complete Verilog front-end, synthesizer, and a complicated verification framework are included. The baseline PDR is particularly implemented for this project (with some codes deliberately removed), and you are required to write your program on top of the reference code. The generated executable has the following usage:

pdrv [**-File** <*dofile*>]

where the **bold words** indicate the command name or required entries, square brackets "[ ]" indicate optional arguments, and angle brackets "< >" indicate required arguments. Do not type the square or angle brackets.

2. Supported Commands

In additional to the circuit parsing, simulation, and verification related commands supported in HW #1, #3 and #5, in this final project, we will support this new command:

```
SATVerify PDR:     Verify the property by PDR
```

Its command interface is included in the reference code. You don't need to work on it. Please refer to the documents/tutorials of V3 and HW #1 for the lexicographic notations and the circuit-related commands.

2.1 Command "SATVerify PDR"

Usage: **SATVerify PDR < -Netid <netId> | -Output <outputIndex> >**

Description: Check the monitor for "netId" or "outputIndex" by PDR. It will evoke model checking with property directed reachability technique.

There can be two kinds of proof results:

1. Property is verified. Printing ---

   ```
   Monitor "varName" is safe.
   ```

2. A bug is found. Printing with a counter-example ---

   ```
   Monitor "varName" is violated.
   Counter Example:
   0: 00000
   1: 00001
   2: 11010
   ```

(note: the reverse order of circuit inputs)

Please note that the baseline PDR implementation does not print out the counterexample. This is one of your TODOs in this project (ref: *pdrMgr.cpp*).

3. Reference code

The reference code is compressed as "final.tgz". Its structure is similar to the one in HW#5. The provided PDR algorithm is implemented in directory src/pdr. Note that this PDR adopts the algorithm proposed in the paper "*Efficient Implementation of Property Directed Reachability, FMCAD11" (fmcad11_pdr.pdf)*, which is almost the same as PDR taught in the lecture note. You are encouraged to study the lecture note and the paper carefully before burying yourself in the source code.

4. Hints

In this section, we provide some promising directions to speed up your PDR implementation.

- **SAT Generalization:** generalize the cube by ternary simulation. Please refer to TODO in the function "`V3SvrPDRSat::ternarySimulation()`".

- **Generalization Ordering:** design a heuristic to decide the ordering (of variables to be removed) for UNSAT generalization

- **Data Structure:** modify the data structure of Cube, TCube

- **Other improvements from V3:** you are also encouraged to study the codes in the original V3 model checker. However, it is to certain degree complicated and implemented in different data structures. DO NOT just copy the code over. Doing so will be treated as plagiarism. Comprehend and rewrite it in this baseline PDR.

5. Benchmark

We provide 45 testcases adopted from HWMCC competition (in "hwmcc" directory). All cases are single output circuits in aig format. We also provide the reference result/runtime on our machine so that you can choose easier cases to improve your PDR in the very beginning (see "ref/result.xlsx"). Note that the results in the column of "v3" are run by the command "VERify PDR", which is the built-in PDR model checker.

6. What you should do?

You are encouraged to follow the steps below for this homework assignment:

1. Read the specification carefully and make sure you understand the requirements.

2. Run the testcases with the dofiles in the "tests" directory. Understand the command usage.

3. Study the SAT package (miniSat). In principle, you don't need to dive in its code to understand how a SAT solver is implemented. Instead, you should get familiar with its interface functions for clause (proof instance) construction and proof.

4. Run the default induction-based UBMC engine to understand how the timeframe expansion model is constructed and how the SAT engine is called.

5. Implement the counterexample trace generation function(s).

6. Run and trace the PDR code for the trivial testcases that the baseline engine can prove.

7. Implement the improvements for PDR algorithm.

8. **[SAT-Based Verification]** Write/Define **at least 3 monitors** for the BUGGY “vending machine” design in HW #1 (you can reuse the monitors from HW#1 or #3). Prove them with your improved PDR or even the hybrid UBMC engine from the latest V3 release. Compare the results on correctness and runtime, memory usage, etc. If any of the properties is false, simulate with the sequential solver of HW#1 to verify the correctness.

9. **[Keep improving]** If your PDR still aborts on any testcase in the hwmcc directory, try to improve your algorithm until it is competitive to or even better than the build-in V3 PDR.

10. **[Verify the correctness]** Please be sure to verify the correctness of your implementation. If your PDR solver generates any incorrect proof result (i.e. safe to unsafe, or vice versa) for the given or hidden testcases, you will be punished with points deduction in this final project.