系統晶片驗證 (SoC Verification)

105 學年下學期 電機系電子所選修課程 943 U0250

Homework Assignment #5 [Interpolation-Based Model Checker]

(Due: 9:00pm, Thursday, May 25, 2017)

0. Objectives

1. Learning how to implement an interpolation-based (SAT-based) model checker using the algorithms in the lecture note.

2. Verifying the "vending machine" design from HW #1 and some other designs from HW #3.

3. Compare the SAT-based with the BDD-based verification tool in HW #3.

1. Problem Description

In this program assignment, you are going to implement an interpolation-based (SAT-based) model checker to verify the designs in previous homework. The similar platform, V3, as in HW #1 & #3 is provided, where a complete Verilog front-end, synthesizer, and additionally a SAT solver (miniSat) with unsat-core/interpolation generator, plus a bounded model checking (BMC) function are included. You are required to write your program on top of the reference code. The generated executable has the following usage:

satv [**-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.

In additional to the circuit, simulation, and verification related commands supported in HW #1 & #3, this verification tool has (should have) the following functionalities. Please refer to Section 2 for supported commands and their usage:

1. An interpolation-based (SAT-based) model checker. Please refer to the algorithm in the lecture note and the description of some related functions in Section 3.

2. Checking whether an assertion property (i.e. AG(p)) is true(safe) by the interpolation engine. If the property is false, print out the counter example.

3. (Recommended) Try some idea(s) to improve the baseline interpolation-based model checker. You can compare your checker with the reference program and/or the PDR in the latest version of V3.

2. Supported Commands

Other than the commands supported in HW #1 and #3, in this homework, we will support this new command:

```
SATVerify BMC:      Verify the property by BMC

SATVerify ITP:      Verify the property by ITP engine
```

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

2.1 Command "SATVerify BMC"

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

Description: Check the monitor for "netId" or "outputIndex" by BMC engine. It will evoke bounded model checking technique provided in *satMgr.cpp*.

There can be two kinds of proof results:

1. 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)

2. The BMC engine cannot find the bug within reasonable sequential depth ---

```
Undecided at depth = 100
```

You don't need to work on this command as the codes for BMC engine are already included in V3.

2.2 Command "SATVerify ITP"

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

Description: Check the monitor for "netId" or "outputIndex" by SAT engine. It will evoke interpolation-based unbounded model checking algorithm in *satMgr.cpp*.

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)

3. Reference code

The reference code is compressed as "hw5.tgz". Its structure is similar to the one in HW#3. Note that the BDD-related codes are NOT included. To compare with BDD-based verification technique, you should copy the BDD codes from your HW #3, or if you think your HW#3 is not solid enough, you can copy them from other classmate's HW#3.

There is only one TODO in src/itp/satMgr.cpp in this homework. It is in the member function "*SATMgr::itpUbmc()*".

Basically there are three parts in the TODO: (1) Create a net to represent the initial state, (2) Construct timeframe #0 circuit and check if the monitor is violated here. The clauses in this timeframe will be treated as the on-set clauses in the interpolation proof, and (3) Build the interpolation-based model checking main algorithm.

Note that there are some useful functions in *sat.h* and *satMgr.h* that you should carefully study and utilize them in your implementation.

You are also encouraged to write a test program to tell the difference between function *assumeProperty()* and *assertProperty()* and use them properly.

3.1 Helper functions in *sat.h*

- **Function Name**: *addBoundedVerifyData(const V3NetId& net, uint32_t& k)*

  This function builds the circuit for *net* at the k_{th} timeframe recursively and add all built gates' CNF formulae to the SAT solver. We highly recommend you to study the code in function *addBoundedVerifyDataRecursively()* to understand how clauses are added to the SAT solver.

  Example:

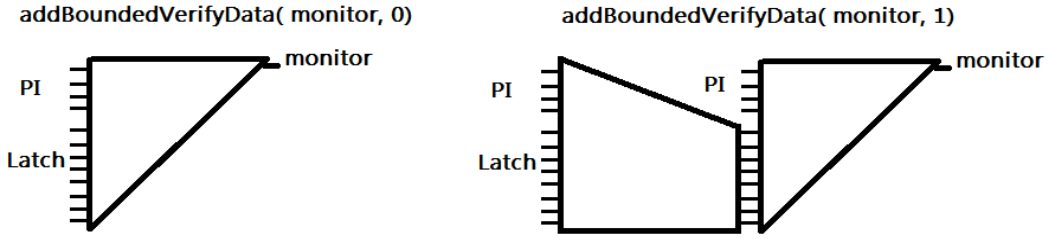

Figure 1

- **Function Name**: *getVerifyData(const V3NetId& net, const uint32_t& k)*

  Get the k_{th} timeframe net's corresponding variable in SAT solver.

3.2 Helper functions in *satMgr.h*

- **Function Name**: *mapVar2Net(const Var& var, const V3NetId& net)*

  Map the variable (in SAT solver) to the net and store the mapping in *SATMgr::_var2Net*. You should use this function to record the common variables between the on-set and off-set of the CNF formula under proof. This map data structure will help the function *getItp()* to generate interpolant using common variables.

- **Function Name**: *getItp()*

  The function builds the interpolant function in the *_ntk* and returns the corresponding net. To use this function, you have to ensure the solver answers an UNSAT and the clauses are marked (by the two functions below) properly.

- **Function Name**:

  markOnsetClause(const ClauseId& cid)

  markOffsetClause(const ClauseId& cid)

  Mark the corresponding clause to on/off set relative to interpolation.

  Note that circuit expansion ordering does matter for marking on/off set. You have to clarify which part of circuit is in the onset and which is in the offset.

4. 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 reference program (ref/satv_ref) on the testcases with the dofiles in the "tests/basic" 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 interpolation-based model-checking algorithm from the lecture notes.

6. **[Basic tests]** Prove the assertions for the smaller testcases in HW#3 (copied here to tests/basic). Make sure your interpolation UBMC can work properly. If any of the properties is false, simulate with the sequential solver of HW#1 to verify the correctness.

7. **[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 interpolation UBMC, reference interpolation UBMC, or even the hybrid UBMC engine from the latest V3 release (see the dofiles in "tests/V3"). Compare the results on correctness and runtime, memory usage, etc.

8. **[Mote tests]** Prove the assertions for the selected testcases from Hardware Model Checking Competition (HWMCC) in the "tests/hwmcc" directory. Note that there are 6 unsat and 4 sat testcases. Please refer to the *unsat.dofile* and *sat.dofile*, respectively. Compare your results with the reference program, or even the hybrid UBMC engine from the latest V3 release.

9. (Optional) Compare with the BDD-based engine in HW#3. Note that we didn't port the BDD-based code in HW#3. You need to copy and revise your HW#3 to make it work in HW#5.

5. What you should turn in

1. Your modified source code and Makefile(s). Please note that we will run plagiarism checking on your source code (with BOTH classmates AND previous students). Any act of plagiarism will lead to severe deduction of your point and we will report this to the departmental office.

2. Your original and/or abstracted "vending.v" (rename it properly) and some dofiles you used to test your program and verify your design. Put them in "tests/" directory. We will use them to test your program.

3. A report file named "<yourID>_hw5_report.pdf" in PDF format. Please describe: (i) Your implementation (make it brief), (ii) Your verification results on basic, hwmcc testcases and vending machine, (iii) Comparison with the ref program (in the "ref" directory) and other model checkers. Please place the report in the root directory (i.e. the same directory as "Makefile") of the homework.

IMPORTANT: Please type "make clean" and remove the unnecessary files (e.g. core dumps, *.o) before submission. Remember to rename the directory to "hw5_<yourID>" before compressing it. Besides, you should compress it under Linux workstation by the command:

```
tar zcvf hw5_<yourID>.tgz hw5_<yourID>
```

6. Grading

We will test your submitted program, your assertions (in your original and abstracted "vending.v"), and your dofiles, and compare its outputs with those of our reference program. The report is important and constitutes a good portion of the score. Please do spend time on the report.

The total score of this homework is 130, including 50 for the report and the correctness of your implementation, and 80 for the performance test. The performance test is to compare your program with other classmates' implementations and the scores are awarded based on the relative rankings.