SoCV Final PDR Report

Yu-Ching, Huang
Department of Electrical Engineering, National Taiwan University
b05901172@ntu.edu.tw

***Abstract*—v3 is a EDA tool capable of doing simulation and verification. It is able to deal with a variety of circuit-representing data structure, e.g. And Inverter Graph (AIG) and Binary Decision Diagram (BDD). It also implement a property direted reachability verification algorithm proposed by an UC Berkeley team. This report is about some improvement which I implemented on the property directed reachability model checker based on the provided v3 engine. To speed up the solver and resolve some memory leak, there are several methods that is implemented. There are mainly three part of optimization: (1) Ternary Simulation. (2) Counter Example Generation. (3) Memory and Deletion, which will be discussed in detail in this report.**

I. Introduction

Sequential verification is hard, both model checking and equivalence checking. Difficult instances are typically solved using several simplification steps followed by multiple verification engines scheduled sequentially or concurrently. Despite all the available tools, numerous practical instances remain unsolved. Therefore, research in formal verification is always on the lookout for methods that can handle difficult cases.

In 2011, a research team in UC Berkeley proposed a Property Directed Reachability algorithm [1] which is based on a novel method pioneeered by Aaron Bradley [2][3]. Compared to other methods, PDR is considerably efficient.

This project improved v3 which is based on PDR. The rest of the report is organized of follow: section II-A explain my implementation of 3-value simulation, section II-B shows the how counter example generated when a property is proven unsafe, and section II-C introduce the resolving of memory leak in original v3 engine. Section III shows the verification results of vending machine design, and section IV demonstrates the efficiency and correctness of my implementation on HWMCC benchmarks.

Note that the algorithm is based on and inverter graph (AIG) circuit, all the circuits described below are considered as AIG circuits.

II. Improvement

A. Ternary Simulation

Ternary simulation is a way of simulation that includes don't care value besides 0 and 1. In the procedure of PDR, proof obligation is found in each timeframe by SAT, this proof obligation will backtrace to former timeframe until blocked. Since SAT solver is not good at finding many satisfying assignments, it may cost a great time in finding a mintern of proof obligation and refute it one by one during the algorithm of PDR. Therefore, it is desirable to find a larger proof obligation that leads to unsafe state in one time.

We can facilitate ternary simulation to expand the mintern found by SAT solver to a cube. A method is to sequentially flip a bit of input into a don't care and see whether the output is unchanged. This method is of complexity $O(I*n)$, where I is the number of input and n is the number of gate, since we should re-simulate the whole circuit after flipping each input. Moreover, some heuristic to determine the order of flipping input should be considered to improve the efficiency.

On the other hand, I proposed a backward simulation that cost only $O(n)$, which is capable of expanding the mintern to a cube effectively and efficiently. Below describe the algorithm.

First we simulate the circuit to obtain the value of output gate. Then we do backtrace 3-value simulation. Start from output gate, if the value of the gate is 1, i.e. controlling value, then both of its input should be 1. If the output is don't care, then both of its input can be don't care too. Otherwise if the output is 0, then when one of its input is known 0, the other can be don't care; when one of its input is known 1, the other should be 0. Doing the above procedure in breadth first search (BFS) order start from output, then don't care term may be propogated to input thus a cube rather than a mintern of input can be generated.

The procedure is of linear time complexity with respect to gate number since it only traverse each gate one time.

B. Counter Example Generation

Let T denotes the transition function. R_k denotes over-approximated reachability at timeframe k. p denotes the property, $\neg p = 0$ means that the property is safe, otherwise unsafe. s' denotes the next state of state s. $SAT?[f]$ denotes using SAT solver to check function f is SAT or not.

When $SAT?[s = R_k \wedge \neg p] = SAT$, it means that s, a.k.a proof obligation, is either a counter example (cex) or a spurious counter example. We should go back a timeframe to see if the previous approximated reachbility R_{k-1} is strong enough to refute it. That is, $SAT?[R_{k-1} \wedge \neg s \wedge T \wedge s']$. If the counter example is blocked in some timeframe during backward procedure, it is a spurious cex, and we can facilitate it to refine the approximated reachability. If the s is not blocked until R_0, that is, initial state, then it is a cex.

The given v3 only check the safety of a property, it does not provide a trace of cex that arise $\neg p$ when p is proven unsafe. As a result, I implemented the generation of cex to further improve it.

The idea of generating a cex is quite simple: store the input assignments of each proof obligation found in SAT solver during the procedure of recursively blocking cube. When R_0 (initial state) is reached, trace from R_0 to root obligation

at R_k and extract those input assignments. However, it is little bit complicated to implement due to some pitfall in v3 implementation.

In v3, a priority queue Q, is maintained to record those proof obligations that had not been blocked, where proof obligations of lower timeframe have higher priority. If a proof obligation is blocked, we should find another proof obligation from Q, continuing blocking it until all proof obligation are blocked or R_0 is reached. Only when R_k is not strong enough to block it, it would then be push into Q with previous timeframe to check if R_{k-1} can block it.

However, in v3 implementation, to speed up the verification, if a proof obligation s, is block in some timeframe, it would be push again into Q with next timeframe $k+1$. The objective of this operation is to see if we can reach s with one more step. If yes, it is also a cex, and we can prove p unsafe even when we had not computed the over-approximation of reachability of higher timeframe.

This enhancement technique would make tracing cex harder since Q now contains additional obligations.

To deal with it, I implement a graph to record the trace for cex, where it record the parent obligation and input assignment for each obligation so that when R_0 is reached, we can trace from R_0 to the root obligation in R_k, extracting their input assignments. Those input assignments is then the cex trace we are looking for.

C. Memory and Deletion

There are hugely memory leak in the given v3, which lead to memory explosion in many HWMCC benchmarks. I traced the source code under pdr directory and found a lot of "newed" memory that is not deleted. Therefore, I resolve it carefully. Below listed all memory deletion I had added.

- pdrMgr.cpp:148
  When the proof obligation is not found in getBadCube(), the "newed" cube will not be used anymore.
- pdrMgr.cpp:166∼170
  Nested list F can all be deleted when the proving procedure ends.
- pdrMgr.cpp:{200∼203, 256∼260}
  When the cubes are popped from the priority queue, they can be deleted. However, I implemented the graph for tracing cex, so we should delete cubes later after finishing tracing for cex.
- pdrMgr.cpp:223
  z is replaced with t hence can be deleted.
- pdrMgr.cpp:296
  Similar to above condition, s is replaced and never be used.
- pdrMgr.cpp:288
  When continue in loop, it skip the delete operation originally.
- pdrMgr.cpp:353
  When the subsumed block is pop out, we can delete it as well.

III. Verification on Vending Machine

Below is the comparison of my implementation and the given pdrv_ref on the buggy vending machine design. Note that column bmc, umc, and itp are experiment results conducted by those verification command in given v3.

Note that my implementation outperforms the ref program in both runtime and memory usage. And it achieves minimum memory usage through out all verification engines.

In the following table, time and memory are measured by "Total time used" and "Total memory used" in usage command respectively.

	Time (s)	Memory (MB)
Mine	12.6	9.8
pdrv_ref	15.5	25.1
bmc	0.22	30.4
umc	0.13	18.5
itp	0.26	26.6

However, after fixing the bug, none of verification command and PDR can prove the rectified design and my added property in reasonable time.

IV. Experimental Result

Table I demonstrates runtime and memory on HWMCC benchmarks of my implementation and ref program. The measure of time and memory are same to section III. Note that in result column TLE means time limit exceed and ME means memory error.

From the results, my implementation do optimize the given v3 engine. Although there are still a number of cases unsolved because of time limit exceeded, many cases are now solved correctly which originally crashed since memory error. And except from small cases that can be solved in few seconds, both of the runtime and memory usage of my implementation are greatly smaller than ref program.

Moreover, I also implement cex generation, where the correctness are proven by simulation command, i.e. the cex do lead unsafe state. However, I would not show the cex here.

V. Conclusion

v3 is a EDA tool capable of doing simulation and verification. It implements the property direted reachability verification algorithm. This work improve the property directed reachability model checker in the provided v3 engine. There are three mainly improvement: (1) Ternary Simulation. (2) Counter Example Generation. (3) Memory and Deletion. Experimental results show that the implementation do optimize v3 engine considerably. Furthermore, it is able to generate counter example that lead to unsafe state when a property is proven unsafe.

References

[1] N. Een, A. Mishchenko, and R. Brayton. Efficient implementation of property directed reachability. In *2011 Formal Methods in Computer-Aided Design (FMCAD)*, pages 125–134, 2011.

[2] Aaron R. Bradley. Sat-based model checking without unrolling. In Ranjit Jhala and David Schmidt, editors, *Verification, Model Checking, and Abstract Interpretation*, pages 70–87, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg.

	Mine			pdr_ref		
benchmark	result	time (s)	memory (MB)	result	time (s)	memory (MB)
6s209b0.aig	TLE	-	-	ME	-	-
6s210b037.aig	SAT	27.1	34.3	SAT	93.4	11538.4
6s210b105.aig	SAT	50.0	38.6	SAT	121.6	11007.2
6s215rb0.aig	SAT	680.4	472.9	ME	-	-
6s221rb18.aig	UNSAT	0.6	208.1	UNSAT	0.8	199.4
6s275rb253.aig	UNSAT	120.9	211.1	ME	-	-
6s275rb318.aig	UNSAT	226.6	344.2	ME	-	-
6s277rb292.aig	TLE	-	-	ME	-	-
6s277rb342.aig	TLE	-	-	ME	-	-
6s282b01.aig	UNSAT	2.9	21.1	ME	-	-
6s289rb00529.aig	TLE	-	-	ME	-	-
6s289rb05233.aig	TLE	-	-	ME	-	-
6s291rb18.aig	TLE	-	-	ME	-	-
6s291rb77.aig	TLE	-	-	ME	-	-
6s317b14.aig	UNSAT	722.8	150.2	TLE	-	-
6s317b18.aig	UNSAT	49.6	29.6	TLE	-	-
6s318r.aig	SAT	11.8	27.0	SAT	200	10144.5
6s322rb265.aig	UNSAT	1.4	382.2	UNSAT	1.6	369.5
6s325rb072.aig	TLE	-	-	ME	-	-
6s327rb10.aig	TLE	-	-	ME	-	-
6s327rb19.aig	UNSAT	0.1	17.7	ME	-	-
6s330rb06.aig	TLE	-	-	ME	-	-
6s330rb11.aig	UNSAT	85.0	85.7	ME	-	-
6s335rb09.aig	SAT	863.7	533.5	ME	-	-
6s335rb60.aig	SAT	649.9	449.9	ME	-	-
6s344rb054.aig	UNSAT	0.14	51.7	UNSAT	0.21	48.8
6s362rb1.aig	UNSAT	0.45	10.2	UNSAT	0.45	10.2
6s372rb26.aig	TLE	-	-	ME	-	-
6s380b129.aig	TLE	-	-	ME	-	-
6s381rb630.aig	TLE	-	-	ME	-	-
6s384rb024.aig	TLE	-	-	ME	-	-
6s388b07.aig	SAT	0.1	21.7	SAT	0.1	19.6
6s388b09.aig	SAT	0.1	21.7	SAT	0.1	19.6
6s389b02.aig	SAT	0.0	21.7	SAT	0.1	19.6
6s389b11.aig	TLE	-	-	ME	-	-
6s391rb379.aig	UNSAT	0.1	12.0	ME	-	-
6s400rb07819.aig	UNSAT	0.2	73.4	ME	-	-
6s406rb067.aig	TLE	-	-	ME	-	-
6s421rb050.aig	TLE	-	-	ME	-	-
6s421rb083.aig	TLE	-	-	ME	-	-
6s515rb1.aig	UNSAT	2.6	9.3	UNSAT	27.8	1505.9
oski1rub05.aig	UNSAT	0.3	101.4	UNSAT	0.3	81.2
oski1rub05i.aig	UNSAT	0.2	79.9	UNSAT	0.2	76.0
oski1rub06.aig	UNSAT	0.3	101.4	UNSAT	0.3	81.2
oski3ub2i.aig	UNSAT	4.7	30.4	ME	-	-

TABLE I
COMPARISON OF RESULTS OF MY IMPLEMENTATION AND REF PROGRAM

[3] A. R. Bradley and Z. Manna. Checking safety by inductive generalization of counterexamples to induction. In *Formal Methods in Computer Aided Design (FMCAD'07)*, pages 173–180, 2007.