SOCV_Final Report

PDR Improvement

Author: Yo-Chi Lee (李友岐)
Email: b03208022@ntu.edu.tw
Department: Electrical Engineering
School: National Taiwan University
Location:Taipei, Taiwan

***Abstract*—This report is about how I improve the property directed reachability model checker based on the provided code and direction. To speed up the design, there are several methods that need to be implemented. Besides, certain heuristics are also helpful.**

I. Introduction

Model checking for sequential circuit is difficult. The complicated cases are usually solved by several verification engines. Although computer scientists have developed many efficient model checkers, there are still thousands of practical instances remain unsolved. Therefore, numerous scientists and engineers still try their best to improve model checker. Few years ago, Aaron Bradley invented a new method to model checking, which is a breakthrough in EDA area. More recently, a research group of UC Berkeley implemented a model checker based on Bradley's earlier work, and they named it property directed reachability (PDR) [1]. Compared to other methods, PDR is considerably efficient. This final project is based on UC Berkeley's work.

II. Overview

In this project, TA provided us some directions to speed up the program. By following these directions, I successfully improve the performance of model checker. There are totally three methods I implemented in the final project, which are SAT generalization, UNSAT generalization and modifying the data structure of class cube. Next three sections are the details of my work.

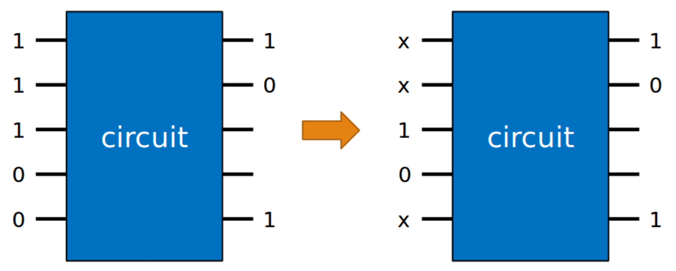

Fig 1: Illustration of SAT gneralization in solveRelative()

III. SAT generalization

SAT generalization can be split into two parts, the first part is in getBadCube() and the other is in solveRelative(). In the first part, we need to check whether monitor's value remains same after each latch changes the value to don't care. If it does, we can change that latch's value to don't care, and further decrease the size of cube. In the second part, we need to check whether the next state remains same after changing each latch's value to don't care. If it does, we can also change that latch's value to don't care just like Fig 1, and further decrease the size of cube. The main problem is that the checking process is too long when the number of latches becomes extremely large. To fix this, we need to filter the next state first. If a next state value is don't care, then we can just ignore it. With this method, the process time of SAT generalization decreases significantly.

IV. UNSAT generalization

Similar to SAT generalization, we can also make some latches become don't care to decrease the size of cube. However, we need to check whether the new cube intersects with the initial state. If it does, then we can't change latch values like that. In the origin design, if the new cube intersects with the initial state, then it will return origin cube. However, the origin cube is too large, this method is not efficient . Actually, if we want to avoid the initial state, we only need to change one latch's value to 1. Since initial state represents the condition that all latch have value 0. As a result, if the new cube reaches initial state, then we change latch's value. If a latch's value is originally 1, then we change this latch's value from don't care to 1. Thus, the new cube won't reach initial state. However, if no latch's value is originally 1, then we can only return the origin cube.

monitor	result	time
Vending, 0	safe	1.0499
Vending ,1	safe	0.0329
Vending ,2	safe	0.0092
Vending ,3	safe	0.0088

Fig 2: Verification results of the vending machine

V. Modifying data structure of class cube

In the origin design, class cube has a fixed size, which is the number of latches. However, sometimes it only requires a much smaller space. As a result, it will lead to a serious memory leak. In order to revise it, we can use hash table rather than array. In C++, map has same effect with hash. Thus, I revise the data structure of class cube, using map instead of dynamic array. When a latch's value becomes don't care, we can erase it from cube→map. This method saves lots of memory and highly increase the efficiency of the program.

VI. Verification results of the vending machine design

With PDR model checker, I can easily verify the basic test cases and vending machine design. The results are all correct, and all the monitors I wrote in design are safe, just like Fig 2. It only takes a little time to verify vending machine, which is much better than BDD.

VII. Experimental results of the hwmcc benchmark

Although I implemented such methods, there are still some cases in hwmcc can't be solved by my program. Unfortunately, I didn't figure out the solution to those cases. Most of the hwmcc cases are successfully verified by my program , but few of them result in core dump or output the incorrect results. The performance of V3 is way better than my program not matter speed or correctness.

VIII. feedbacks and suggestions

In this semester, I learned a lot of verification relative knowledges. Although the homework is a little heavy, I still consider this class to be one of the best course I've ever taken. However, I think this class contains too much content, some of them are selectable. Maybe less content would be better .

IX. REFENECE

[1] Efficient Implementation of Property Directed Reachability. Niklas Een, Alan Mishchenko, Robert Brayton,2011.