國立臺灣大學電機資訊學院電子工程學研究所

碩士論文

Graduate Institute of Electronics Engineering
College of Electrical Engineering and Computer Science
National Taiwan University

Master Thesis

利用時態分解技術增進性質導向可達性技術

Improving Property Directed Reachability with Temporal Decomposition

楊明仁

Ming-Jen Yang

指導教授：黃鐘揚 博士

Advisor: Chung-Yang (Ric) Huang, Ph.D.

中華民國一百零五年七月

July, 2016

doi:10.6342/NTU201601674

誌謝

在清水斷崖外海的獨木舟漂泊之際，才慢慢意識到自己總算是真的畢業了，不再那種有出去玩出去或吃大餐，至終總會想起論文還沒寫完，投影片還沒做完的虛假狂歡感覺。

能夠完成這篇論文，我想我有許許多多要感謝的人事物(以下根據注音順序排序)：

感謝 Ric 這位良師益友，引領我踏入 EDA 的大門，並且一路上不斷地給我教學和指導，同時給予我們最棒的自由研究風氣和最大程度的信任。

感謝范寬學長，在研究上給了我相當大的鼓勵和啟發，如果不是你，我不會能夠那麼順利的找到研究的方向。

感謝倒垃圾的阿姨每天辛苦的幫我們整理垃圾，若沒有你，我在實驗室製造的垃圾應該已經塞滿電二了。

感謝電機溫馨的 B99 眾，曾冠豪沈昇勳郭令儀王國婷吳彥諶曾泓諭柯婷瑋劉人碩陳乃群周昱廷洪銘駿林彥如陳思鐸樊聖亭蔡曜宏等等等，各種溫馨的宵夜和吃飯嘴砲還有重訓團，感謝那些 seminar 的簽到，讓我不用每個禮拜要卡一個下午在 112，感謝有這一群人一起奮戰面對論文大魔王。

感謝女友 Emma，在適當的時候追我，感謝你小我五歲但是卻絲毫沒有何一點不成熟的地方，感謝這五個月來的陪伴、視訊、那些加油和鼓勵，還有在陽明山獻給我的初吻，你是我每天起床之後想要寫論文的最大動力。

doi:10.6342/NTU201601674

感謝臉書奇奇怪怪莫名其妙修圖修很大或是光線角度很詭異的的正妹和粉絲專頁，讓我每天都有正當、合法、又不傷害身心的管道抒發壓力。

感謝凱舜和 Cindy，願意信任我給我很棒的 offer，讓我在寫論文的時候沒有後顧之憂，甚至還讓我出國玩三個月再 on broad，以後會鞠躬盡瘁的…。

感謝江弘勛學長，提供我很多研究的參考資料和文獻，如果沒有你，我的論文應該會晚好幾個禮拜才能誕生。

感謝前女友，在適當的時候離開我，我因為你而探索了很多人生的方向和意義，也讓我碩二的時候能夠更專注在學業上面。

感謝系男排，接納我六年，同時也願意讓我這個退化的很嚴重的大雷包上場，享受和你們一起在場上拼鬥的時光，那些一起拿到的獎盃，回想起來會是我大學時代最美好的回憶之一。

感謝系女排，雖然每個學妹加學姊都在嗆我，但你們合起來總有一種神奇的魔力，在我失戀的時候幫我療傷，在我畢業壓力很大的時候能夠讓我放鬆，也要感謝你們每一個活動所送的小禮物和充滿認真祝福話語的小卡片，陪我度過了很多個低潮的時刻。

感謝小熊學長，常常回實驗室關心我們的近況，給我們許許多多做研究的建議和流程上的幫助。

感謝智仁和大胖學長，感謝你們在 Cadence Intern 時的照顧和後來許許多多的勉勵，後來還常常特別跑來台北跟我們吃飯和參與我們的口試。

感謝這個社會不明就裡的繳稅，也感謝沒有人特別追究資源分配的合理性，讓我們這群最會考試的人拿到了大把大把的資源，有充足的經費，有好的系館，校園環境，(不一定)好吃又便宜的餐廳，可以出國比賽，買伺服器，領薪水做研

doi:10.6342/NTU201601674

究，但同時也讓我這樣無權無勢沒有家庭背景的人有翻身、自由的決定自己想過的人生的機會，我會在畢業之後謹記這樣的社會責任，盡力為這個社會帶來正面的影響。

感謝廁所阿姨把廁所打掃的乾乾淨淨，讓我每天上廁所的時候都能確保身心的順暢。

感謝我 23 歲的身體，任勞任怨地被我操勞了快要六年，熬夜家常便飯，用眼過度，三餐不正常是常態，我在接下來的人生中一定會好好對待你的。

感謝吳政穎學長，為實驗室的學弟妹寫出了一個龐大而完善整潔實用的驗證架構 V3，同時也常常不辭辛勞地給我們問問題，讓我得以在這神一般的 code 下乘涼。

感謝王立為，一起征戰了許多大大小小的比賽和練習，尤其是最後一學期受到了你非常多的幫忙和照顧。

感謝緣分，讓我們相聚，也感謝緣分，讓我們必須別離，才更懂得珍惜相聚的時光。

doi:10.6342/NTU201601674

中文摘要

因為對於安全或是不安全的案例都有突出的效能，性質導向可達性技術從二零一一年發表以來一直是模式驗證問題中最有效率的演算法。然而，因為模式驗證問題本質上的高複雜度，還是有許多的案例沒有被驗證。因此，改善性質導向可達性技術的研究是一個非常重要的主題，近來也產生了許多相關的研究。在這篇論文中，我們提出了三種基於將時態分解技術的概念融合於性質導向可達性技術的創新方法。我們將我們所提出的方法實作於“驗證三”的驗證架構中，並且使用世界”硬體模式驗證競賽”的案例與原本的性質導向可達性技術做比較。實驗結果顯示我們的方法可以驗證許多原本未被驗證的案例，對不安全的案例有一點四倍的運作時間提升，代表著這些方法可以與原本的性質導向可達性技術在相當多的基準點上互補。

doi:10.6342/NTU201601674

ABSTRACT

Property Directed Reachability has always been the most efficient algorithm for the safety property checking problem for its well performance on both safe and unsafe case since its publication in 2011. However, there are still a large number of cases remaining unsolved due to the intrinsic high complexity of model checking problem. Therefore, improving PDR is an important subject and attracts lots of research recently. In this paper, we present three novel approaches that integrate a sequential optimization technique, called temporal decomposition into PDR. We implemented our work in V3 and compared to original PDR in V3 on HWMCC benchmarks. The experimental result shows that the proposed methods solved many originally unsolved cases, and improved unsafe cases by 1.4x in runtimes. It means that these methods can complement original PDR on a large set of benchmarks.

doi:10.6342/NTU201601674

CONTENTS

doi:10.6342/NTU201601674

doi:10.6342/NTU201601674

LIST OF FIGURES

doi:10.6342/NTU201601674

LIST OF TABLES

doi:10.6342/NTU201601674

Chapter 1 Introduction

Model Checking [1] is an automatic procedure that determines if a given finite state machine satisfies a specification. It is widely used in both hardware and software verification. Many real cases applications, such as sequential equivalent checking and, can be viewed as a model checking problem.

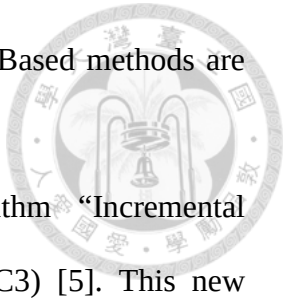
However, as long as the rapid development of system on chip (SoC), the NP complexity and the high space requirement constrain the applicability of current model checker in large circuits or complicate designs. Therefore, lots of researcher always searching for methods to improve model checking technique to handle difficult cases.

Methods of model checking originally focused on the use of Binary Decision Diagram (BDD) [2]. These algorithms compute the image of current states iteratively until a fix point is reached, or a counter-example is founded. But BDD often suffers from the size explosion during image computation as the number of state variable increases.

In the beginning of the twenty-one century, several SAT-Based algorithms based on induction [3] was proposed. These algorithms unroll a circuit across multiple time frames and solve the property by the powerful SAT solver. However, some cases with long counter-example trace become intractable because they require solving a very large unrolling of the circuit.

In 2003, McMillan proposed a new SAT-Based method based on Craig's Interpolation Theorem [4]. It uses interpolants to compute the over approximation of the set of the reachable states. If the over approximation converges to a fix-point, then the property is proved to be safe. Because the interplolants can be extracted from UNSAT

doi:10.6342/NTU201601674



core in linear time, and represent set of states easily. Interpolation-Based methods are one of the most efficient model checking algorithms.

In 2011, Aaron Bradley proposed another novel algorithm "Incremental Construction of Inductive Clause for Indubitable Correctness" (IC3) [5]. This new SAT-Based method doesn't require unrolling the circuit. Moreover, instead of using interpolation to represent the reachable states, it uses cubes and clause in each timeframe to constrain the possible state variable, and block only the state that would lead to a real counter-example recursively. Research result shows that IC3 outperforms Interpolation-Based methods in many cases.

Niklas Een later proposed an improved version of implementation of IC3, called "Property Directed Reachability" (PDR) [6]. One of its improvement is that it simplified some UNSAT generalization procedure in IC3, and uses the ternary value simulation (also call SAT generalization) to reduce the size of proof-obligations.

Today, almost every verification framework like ABC [7] and V3 [8] implemented PDR algorithm as one of the main portfolio model checking engine. In this work, we extend PDR to be able to unroll circuit first several timeframes to represent the initial state, rather than finding it by blocking cubes with proof-obligations.

1.1 Motivation

The Hardware Model Checking Competition has always been the most representative competitive event for model checkers. It collects many industrial benchmarks and calls for realistic solutions. In our previous work, we built a multi-engine portfolio verification framework and won 2nd place in HWMCC [9] for years. Nevertheless, the are still many instances remaining unsolved There are several

doi:10.6342/NTU201601674

works that also modified original PDR, try to get better result like [10] [11] [12] [13] [14]. But to the best of our knowledge, there are no works that combine temporal decomposition with PDR.

Hence, we try to explore the PDR algorithm in this direction and experiment the performance for different configuration of PDR by unrolling the circuit.

1.2 Contributions of the Thesis

In this thesis, we explore PDR in detail and try to improve it performance through observe its solving process. We implemented the transient signal detection algorithm in V3, collected statistics of the existence and frequency that transient signals occur in HWMCC benchmark suits. And we proposed several alternative heuristics to improve PDR: the forward reachability constraint on PDR, the time-shifted PDR and the multi-step PDR.

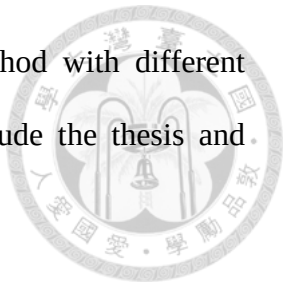
The experimental results show that the time-shifted PDR does improve the performance of PDR, solves more cases than original PDR with 0.7x runtime in overall cases, and all other method solves some unique cases that wasn't able to be solved by original PDR, providing a great variability to original PDR.

1.3 Organization of the Thesis

The introduction outlined the related work, motivation and the contribution of this thesis. The rest of this thesis is organized as follows: We give the fundamental background knowledge of this work including definition of model checking problem, algorithm of PDR in Chapter 2. Our main algorithm – time-shifted PDR and multi-step PDR, and the explanation of its correctness are presented in Chapter 3. In Chapter 4, we

doi:10.6342/NTU201601674

show the experimental results of this thesis, comparison our method with different decomposition depth with original PDR in V3. Finally, we conclude the thesis and suggest the possible future works in Chapter 5.



doi:10.6342/NTU201601674

Chapter 2 Preliminaries

In this chapter, we give the basic background knowledge related to those terminologies that mentioned in our work.

2.1 Propositional Satisfiability

Let $\mathbf{B} = \{\mathbf{True}, \mathbf{False}\}$ be a binary domain. The assignment to a Boolean variable (also referred to a propositional atom) is defined by $\mathbf{B}$. A *literal* is either a Boolean variable or its negation. A *clause* is a disjunction of literals whereas a *cube* is a conjunction of literals. A satisfiability instance, or a Conjunctive Normal Form (CNF) formula is a conjunction of clauses. A satisfiability instance is *satisfiable* (SAT) if and only if there exists an assignment (also referred to a model) to all variables such that every clause in the instance is true. Otherwise it is *unsatisfiable* (UNSAT).

The model checking problem consists of many propositional formulae. In proposition logic, a proposition formula is formed with proposition atoms with the use of logical connectives like conjunction, disjunction, negation and implies. A propositional formula can be converted into a SAT instance in linear time and space while the satisfiability is preserved.

doi:10.6342/NTU201601674

2.2 Finite State Transition System

A digital circuit can be represented as a Finite-state transition system to verify its correctness. Here we follow the conventional notations, a *Finite-state Transition System* M is a tuple $\langle I,S,Init,Tr\rangle$, where I is the set of primary input variables, S is the set of states variable, $Init(S)$ is a propositional logic formula over S describing the set of initial states, and $Tr(I,S,S')$ is a propositional logic formula over primary input variables I, current-state variables S and the next-state variables S', denoting the transition relation. A state s is an assignment to all state variables over S. A run of M is a sequence of (I,S) pairs: $(i^0,s^0), (i^1,s^1), (i^2,s^2), \ldots$, such that $Init(s^0)$ as well as $Tr(i^k, s^k, s^{k+1})$ for every adjacent state pair is satisfied.

2.3 Model Checking Problem

Before introducing model checking problem, we give a brief description on temporal logic formula. Temporal logic formula is commonly represented in Computation Tree Logic (CTL), Linear Temporal Logic (LTL) formulas, or CTL* formulas which is a superset of CTL and LTL, for it freely combines path quantifiers and temporal operators.

In temporal logic there are two path quantifiers: **A** stands for all computation paths while **E** refer to the existence of some computation paths. Besides, there are many temporal operators describe property along the path in computation tree. In this thesis we introduce two temporal operators: **G** stands for all states globally and **F** for some states in the future eventually. For instance, in temporal logic f = **AG**(φ) refers to the safety checking problem where f is true if and only if for all paths starting from I_0 in

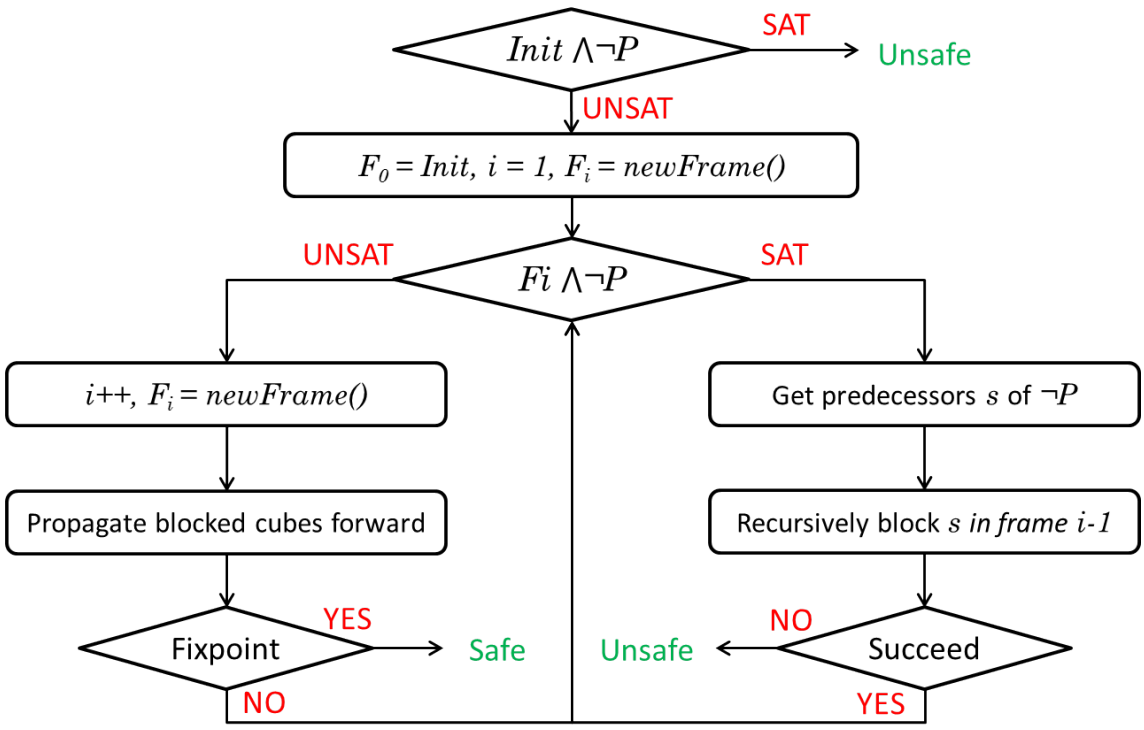
doi:10.6342/NTU201601674

finite-state system *M*, all states on those path satisfy φ. In other words, a safety property is a property stating that "something bad should never happens".

Nowadays, verification of checking safety property is a critical mission. We want to make sure that an asserted property should always hold throughout the system and assure a bug will never occur. As the design grows larger, it is a great challenge to explore the counter-example of bug or prove that the system is always safe. In this thesis, we focus on safety property checking problems from Hardware Model Checking Competition (HWMCC) benchmark. In the next section, we'll give an introduction to the Property Directed Reachability algorithm which is one of the most efficient algorithm for safety checking problems.

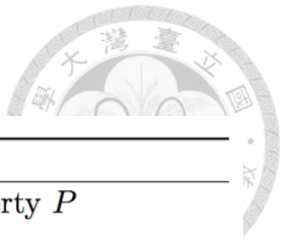
2.4 Property Directed Reachability

In this section, we give an overview of Property Directed Reachability algorithm. For further description and information, please refer to [6]. The high-level description of PDR is show in Figure 2.1 and Algorithm 1. PDR



doi:10.6342/NTU201601674

Figure 2.1: Program flow of PDR



Algorithm 1 PDR

Input: A transition system $M = \langle I, S, Init, Tr \rangle$ and a property P
Output: a counterexample or a fixpoint

```
1.   if (Init ∧ ¬P == SAT) then
2.     return counterexample
3.   end if
4.   F_0 ← Init
5.   F_1 ← newFrame()
6.   for (i ← 1 to ∞) do
       // blocking phase
7.     while (F_i ∧ ¬P == SAT) do
8.       cube ← getBadCube(¬P, i)
9.       if (!recBlockCube(cube, i)) then
10.        return counterexample
11.      end if
12.    end while
       // propagation phase
13.    F_{i+1} ← newFrame()
14.    if (propagateBlockedCubes(i)) then
15.      return fixpoint
16.    end if
17.  end for
```

From the high-level view, the algorithm maintains a sequence $\langle F_0, F_1, \ldots, F_k \rangle$ of over-approximations to reachable states within i steps from initial states. Each element in the sequence is set of clauses, denoted by F_i, called *frames.* PDR uses these frames to represent the reachabilities of the system.

The algorithm refines these frames for many iterations until they reached an *inductive fix-point*, or they found a real *counter-example*. At the end of each iteration k, PDR maintains following invariants:

1. F_0 only contains initial states ($F_0 = I_0$).

2. All frames F_i except F_0 are a set of clauses.

3. The clauses in F_{i+1} is a subset of F_i for $i > 0$, hence, every state in F_i is also in F_{i+1} $(F_i \Rightarrow F_{i+1})$

4. F_{i+1} is the over approximation of $F_i \wedge Tr$, the set of states reachable in one transition from F_i $(F_i \wedge Tr \Rightarrow F_{i+1})$

doi:10.6342/NTU201601674

5. Every frame implies the property under verification except for the last frame ($F_i \Rightarrow P$ except for F_N).

Algorithm 2 recBlockCube

Input: A cube to be blocked and a depth i
Output: True or False
1. **if** $(i == 0)$ **then**
2. **return** $False$
3. **end if**
4. **while** $(F_{i-1} \wedge \neg cube \wedge Tr \wedge cube' == SAT)$ **do**
5. $preCube \leftarrow getBadCube(cube, i-1)$
6. **if** $(!recBlockCube(preCube, i-1))$ **then**
7. **return** $False$
8. **end if**
9. **end while**
10. $cube \leftarrow generalize(cube, i)$
11. $addBlockedCube(cube, i)$ // strengthen F_i
12. **return** $True$

Algorithm 3 propagateBlockedCube

Input: a depth k
Output: True or False
1. **for** $(i \leftarrow 1$ **to** $k)$ **do**
2. **for each** $(cube \in F_i)$ **do**
3. **if** $(F_i \wedge \neg cube \wedge Tr \wedge cube' == UNSAT)$ **then**
4. $addBlockedCube(cube, i+1)$
5. **end if**
6. **end for**
7. **if** $(F_i == F_{i+1})$ **then**
8. **return** $True$
9. **end if**
10. **end for**
11. **return** $False$

In detail, each iteration of PDR consists of two phase: blocking phase and propagating phase. At the start of each iteration, the algorithm goes into the blocking phase, it checks if the property can be violated in one transition from the states in the highest frame, solving the SAT instance: $F_n \wedge Tr \wedge \neg P$. If this query is satisfiable, a satisfying state cube $s \vDash F_n \wedge Tr \wedge \neg P$ is extracted from the counterexample given by SAT solver, meaning that there's a chance for some states in the second highest frame reach this state s in one transition. So the algorithm then proceeds to recursively block cube s. To block s in previous frame, another SAT query: $F_{n-1} \wedge Tr \wedge s'$ is used

doi:10.6342/NTU201601674

where s' stands for cube s in next state variables. If this query is satisfiable, a satisfying state cube is extracted from this SAT instance. The algorithm continues to try to block cubes in each previous frame. If, eventually, a cube cannot be blocked in F_0, a counterexample trace is provided to disprove property P.

When a cube s is successfully blocked in some frame F_i, a clause blocking this cube that is, the negation of this cube, is added to every frame F_k where $k \leq i$. This maintains property 3 of the trace. And there is a clever improvement, if F_i is not the highest frame then we push s into the bad cube queue with higher frame variable. So the solver doesn't need to solve this same cube s in the next iteration, and this also makes it capable of finding a folded counterexample in small time frames.

Once the query: $F_n \wedge Tr \wedge \neg P$ is unsatisfiable, the algorithm then proceeds to propagation phase, and a new frame F_{n+1} is added. During this part of the algorithm, all blocking clauses c are checked to see if it can be blocked in later frames. A SAT query: $F_i \wedge c \wedge T \wedge \neg c'$ is made for each clause c in F_i. If this formula is unsatisfiable, then it means that c is inductive relative to F_i, and then we propagate c to F_{i+1}. If at any point some adjacent two frames become identical, $F_i \equiv F_{i+1}$, then the property is proved since the three conditions of an inductive invariant is satisfied, $I_0 \Rightarrow F_i$, $F_i \wedge Tr \Rightarrow F_i$ and $F_i \Rightarrow P$. And the inductive invariant consisting of the blocking clauses in F_i is found.

2.5 Temporal Decomposition

According to [15], logics that are only relevant during initial time frames exist in hardware designs due to numerous causes. One such reason is the initialization sequence that allows a design to power-up in a non-deterministic state to a "reset state".

doi:10.6342/NTU201601674

We address two particular type of these logics: transient signals which after a certain number of time frames settle to a fixed constant value, and initialization inputs used to encode intricate initial states which become irrelevant after a certain number of time steps. They also proposed automated techniques to detect and eliminate such irrelevant logic by unrolling the first few frame of circuit. Here we give a sketchy description.

2.5.1 Transient signals

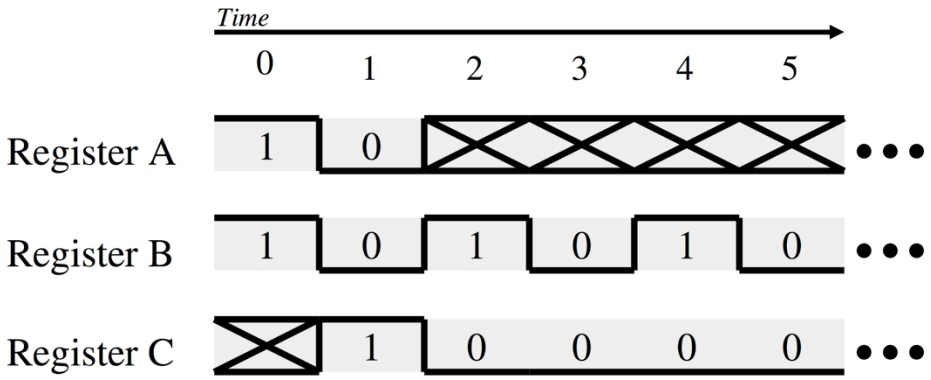


Figure 2.2: The definition of transient signals

Definition 1 *(Transient Signal)*: Let *x(t)* denote the value of signal *x* at time t. A transient signal is any Boolean signal s such that $\exists T \in \mathbb{N}. \exists C \ni \{0,1\}. \forall\, time\ t \geq T. s(t) = C$ here. The smallest T for which this holds is referred to as the transient duration and *C* as the settling constant.

In figure 2.2, register C is a transient because it takes value 0 starting at time 2 and holds this value for all time. Register A and B are not transients because they remain undetermined in value or not settle to a fixed constant value.

As in [15], we use ternary simulation to efficiently detect a subset of the transient signals, thus it is very fast and scalable. It works by conservatively modeling primary input with ternary X values and simulating until a state is repeated. Upon convergence, the set of observed 3-value state forms an over-approximation of the set of reachable

doi:10.6342/NTU201601674

states. An example ternary simulation run is depicted in Figure 2.3, where the state at time 2 is repeated.

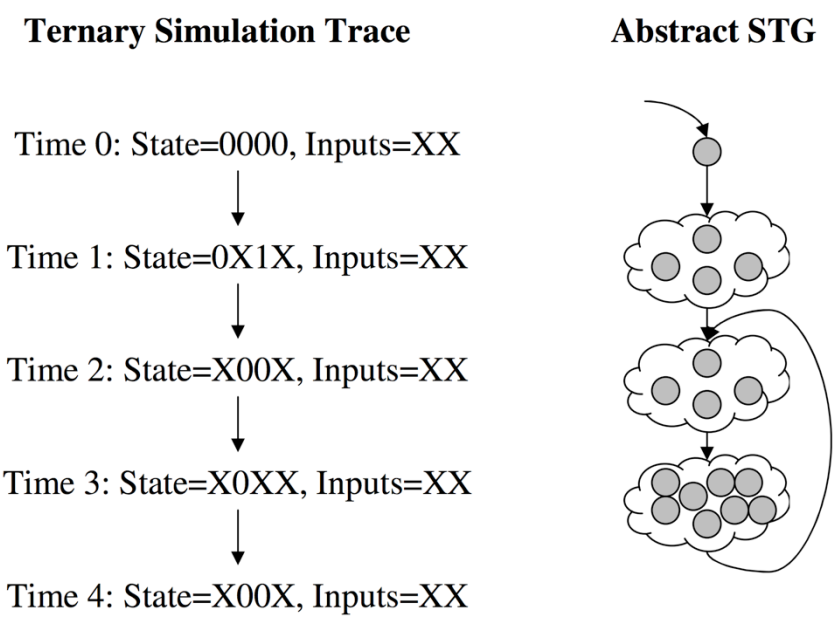


Figure 2.3: Using ternary simulation to detect transient signals.

By definition of transient signals, we know that they will be fixed to a constant value after the maximum transient duration, no matter the. Thus we can *time-shift* the design, unrolling *maxTransientDuration* times of circuit, and stitch to the original circuit corresponds in time 0. Figure 2.4 demonstrates how such shifting may be accomplished. The design's initial state is modified such that the time-shifted design starts in any reachable stats in *maxTransientDuration* time steps.

doi:10.6342/NTU201601674

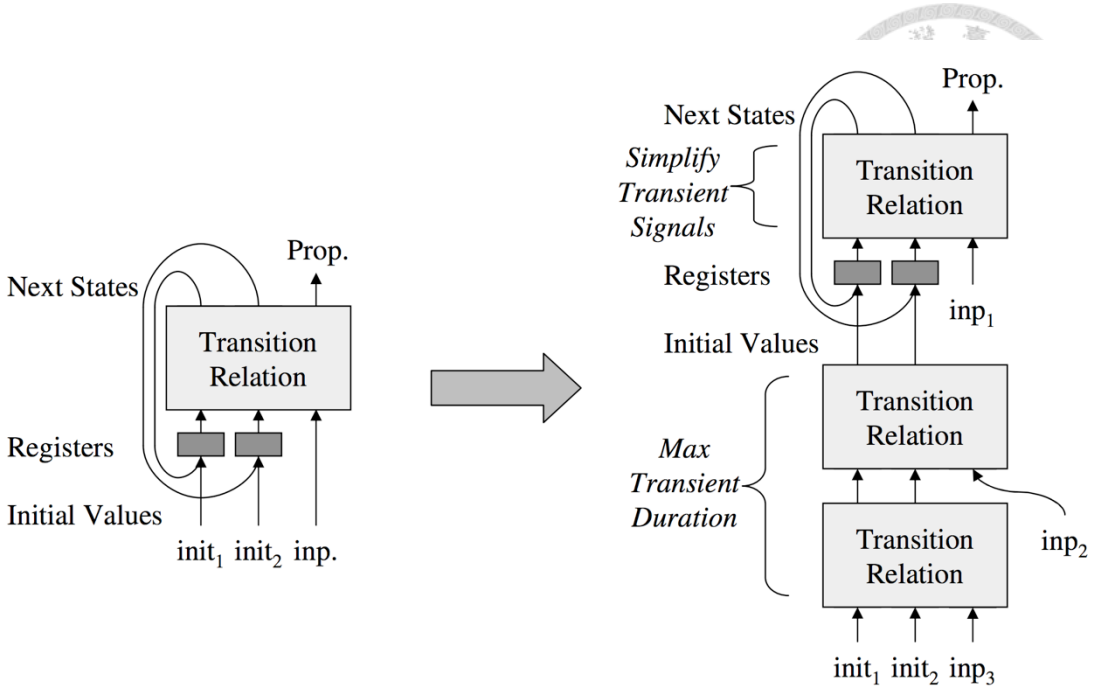


Figure 2.4: Time-shifting a design and simplifying transient signals

In this way, the netlist is simplified by merging transient signals with their respective settling constant. For every k signals being eliminated, the over-all search space is reduced by 2^k times, so even there aren't many transient signals in the benchmark designs, the reduction is still promising.

Before the maximum duration, these signals may be any values. We have to use bounded model checking (BMC) algorithm to check the properties before the maximum transient duration.

doi:10.6342/NTU201601674

2.5.2 Initialization inputs

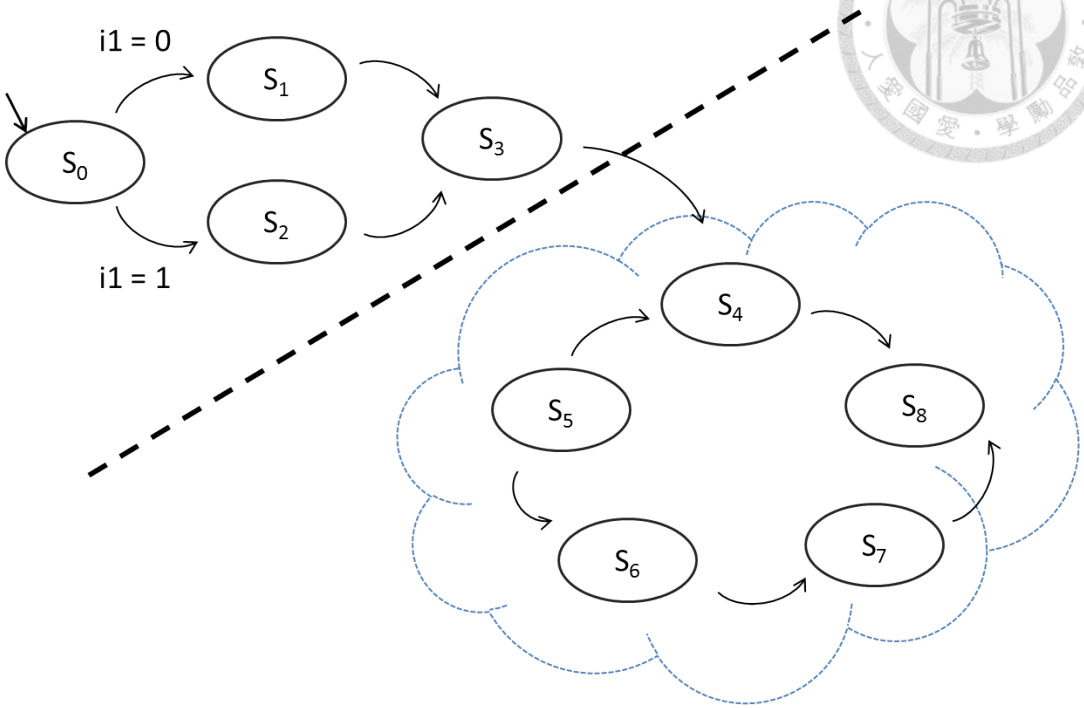


Figure 2.5: The input i1 is irrelevant to circuit after time frame 2.

We unrolled the circuit and add SAT constraints with the Cone of Influence reduction, so the initialization inputs will automatically be ignored by SAT solver for variable stands for the irrelevant inputs will be free variable in the SAT instance.

doi:10.6342/NTU201601674

Chapter 3 PDR with Temporal Decomposition

In this chapter, we give an introduction to the proposed algorithm, that is, the time-shifted PDR and multi-step PDR. We first state our motivation in section 3.1. In section 3.2 we add cubes getting by simulation to first few frames in PDR, try to represent the reachability in a more precise way. In section 3.3 and 3.4 we combine the concept of temporal decomposition and propose the novel time-shifted PDR and multi-step PDR.

.

3.1 Motivation

Nowadays, in the field of hardware verification the PDR is very efficient with great performance in many instances, there are some cases remain undecided or take too much time to solve. Through observation the solving process on these difficult cases, we find that in most of these cases make PDR timeout with two main reasons: 1. PDR keeps refining cubes in first few time frames and finds excessive numerous proof obligations. 2. PDR expands to a much higher time frame than normal cases without getting conclusion.

For the first case it often results from bad blocking cubes which are at lower time frame and cover smaller solution space. We can explain the first case by the existence of the input sequence in some instances. The reachability of first few frames is too coarse, and it will only converge after a certain number of time steps.

Unlike [12], against this phenomenon, they managed to find better quality blocking cube, we integrate the concept of temporal decomposition to the PDR, since the reachability of first few frame is too coarse to easily be blocked by good-quality cube,

doi:10.6342/NTU201601674

we want to make PDR doing verification from the later frames, which has a smaller reachability.

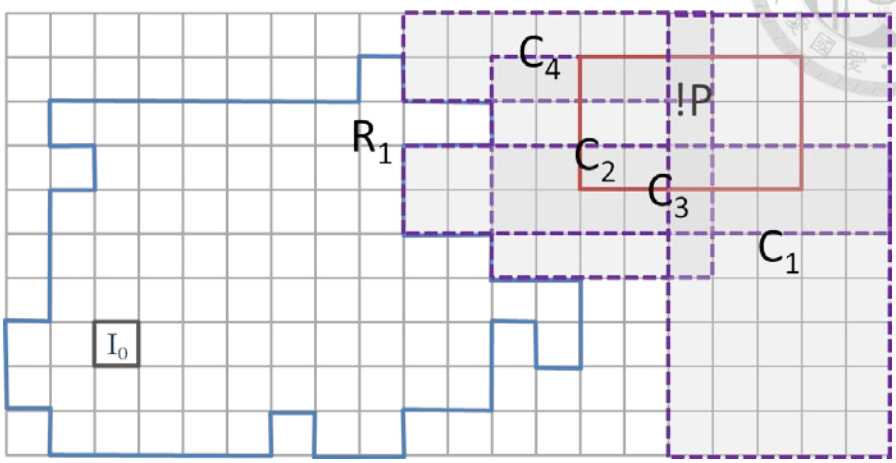


Figure 3.1: A too coarse R_1 causes the number of cube explodes

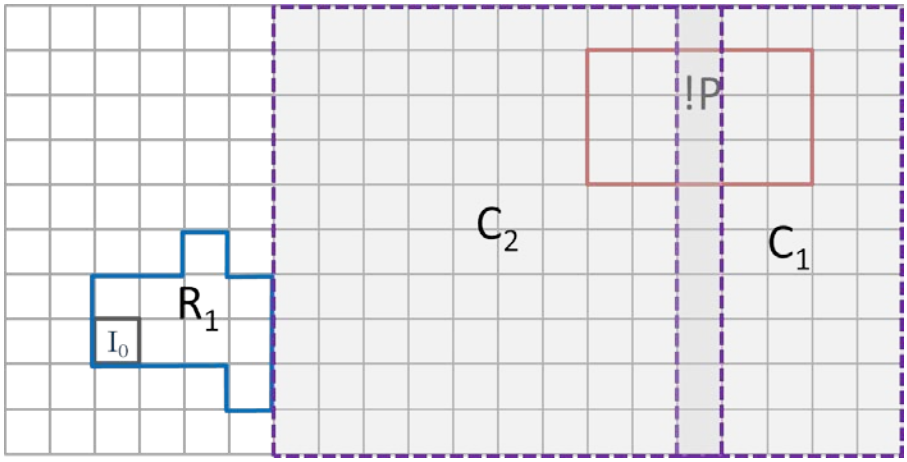


Figure 3.2: For a tighter R_1, the quality of cube raises, so we need only 2 cubes to successfully block ¬P.

On the other hand, the second issue arises from too coarse blocking cube, PDR cannot get to the fixed point since there always exists some blocking cube cannot be propagated to the next frame, and in the following blocking phase PDR generates some blocking cubes which are much similar to those cubes in previous frame, PDR have to refining these similar cube in repeat. That is, the over-approximated reachability was

doi:10.6342/NTU201601674

under-estimated by cubes in first few frames, it grows too large so it keeps intersecting with those blocking cube. Thus PDR have to add some similar blocking cubes and new frames to makes it inconclusive in the given time limit.

With this in mind, if we can start from a deep initial state, the reachability will be more precise, thus it won't be under-estimated.

These two types of problems are the fundamental weak point of PDR, the generalization process mainly relies on the result of SAT solver, and sometimes the blocking cube turns too coarse or too tight.

In this chapter, we use the concept of temporal decomposition to escape from these predicaments that cause PDR time-out.

3.2 Forward Reachability Constraint

As mentioned in previous chapter, we can use ternary simulation and temporal decomposition to get the possible value of each register at each time frame. Intuitively, we want to utilize this information to help PDR.

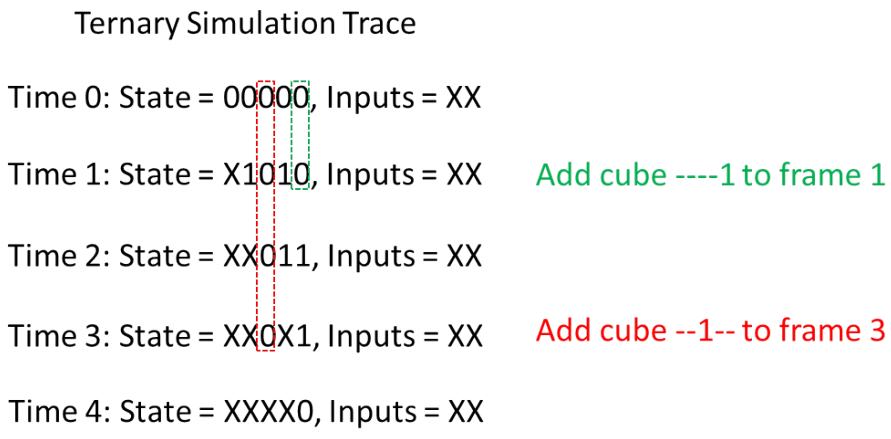
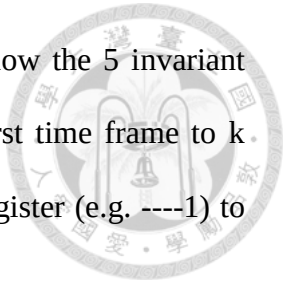


Figure 3.3: Illustration of how cubes added in frames of this method.

doi:10.6342/NTU201601674



Above shows this straight-forward cube extract algorithm, follow the 5 invariant that PDR keeps, if the value of some register is always 0 since first time frame to k frame, then we can add a cube that only contain the value of that register (e.g. ----1) to frame k to restrict the too coarse reachability.

In the perspective of state space, we can see that these cubes may help PDR to get a more precise reachability. We think this precise reachability can help PDR to block more spurious proof-obligation in higher time frame. However, although the quality of these cubes are good (contains only one literal) with this method, we may add too much unnecessary cubes to the solver, so that the SAT solver is severely burdened by these clauses.

But there are still some originally unsolved cases solved in an incredible short time by this method, which happened to block some critical space with these extra cubes.

3.3 Time-shifted PDR

As mentioned in previous chapter, when PDR found a proof-obligation, it will try to block that proof-obligation in previous frame. The whole process of recursive blocking cube in PDR can be viewed as a huge decision tree. The repeated tries and errors are one of the many main reasons that causes PDR run out of time. But it is often the case that PDR blocks wrong cubes again and again, and the real cube that causes the counter-example may be block in the very later position of the queue.

doi:10.6342/NTU201601674

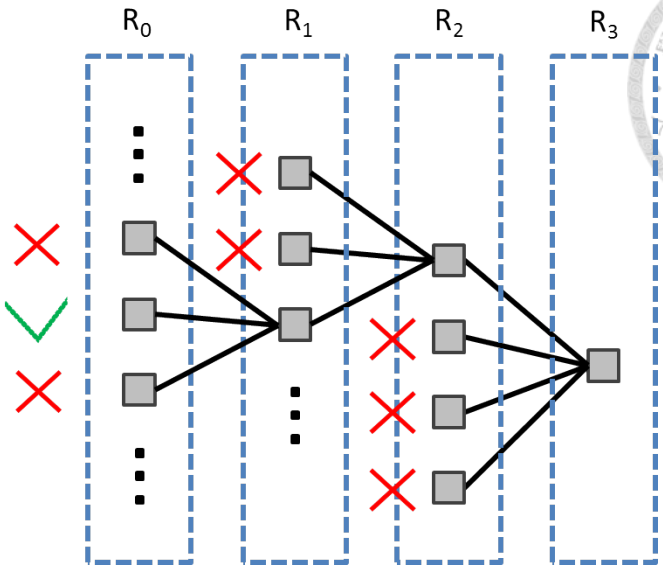


Figure 3.4: The recursive blocking decision tree of PDR

Intuitively, we consider that if we can find the right path to disprove the property earlier in the tree, we may not get into the bog of huge error decisions and save lots of time.

Hence we combine the concept of temporal decomposition into PDR, invented new algorithm call “time-shifted PDR” and “multi-step PDR” in the next section, trying to utilize the great power of BMC that it can find real counter-example quickly, to replace the lengthy iteratively decision process in original PDR.

As mentioned in section 3.1, we use the concept of temporal decomposition, unrolling several times of circuit, stitching this unrolled circuit to the original circuit in time 0 and simplify circuit with transient signals. In the perspective of state space, this equals to changing initial states from R_0 (i.e. all zero state) to R_k, like the example in section 3.1, we consider this would help PDR to solve the too coarse or too tight reachability issue.

doi:10.6342/NTU201601674

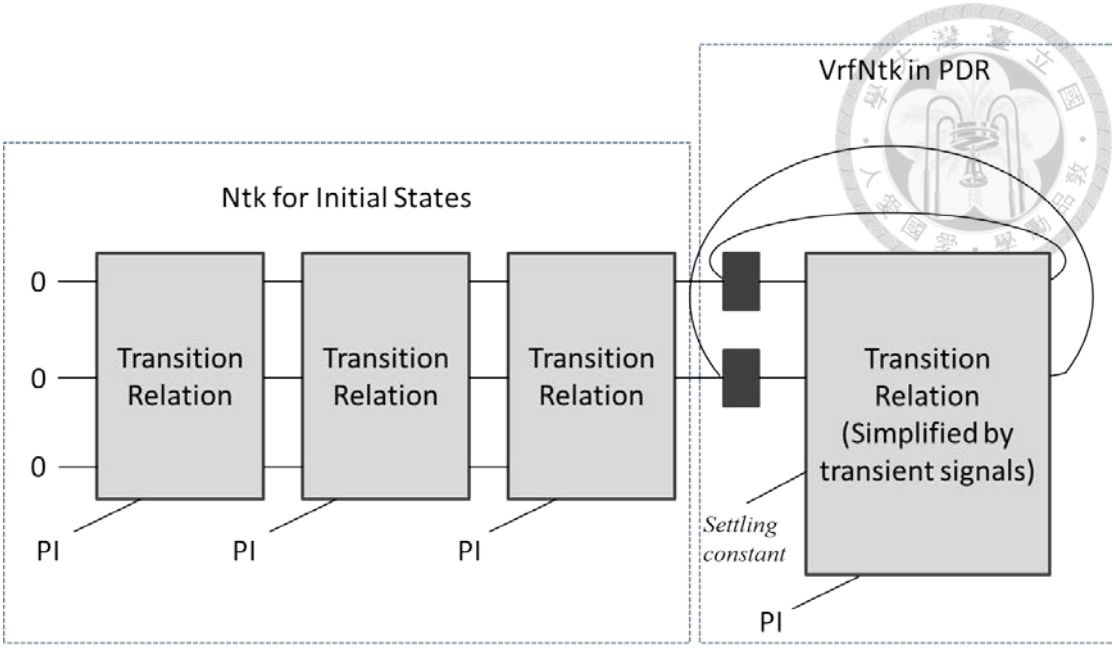


Figure 3.5: Apply temporal decomposition into PDR.

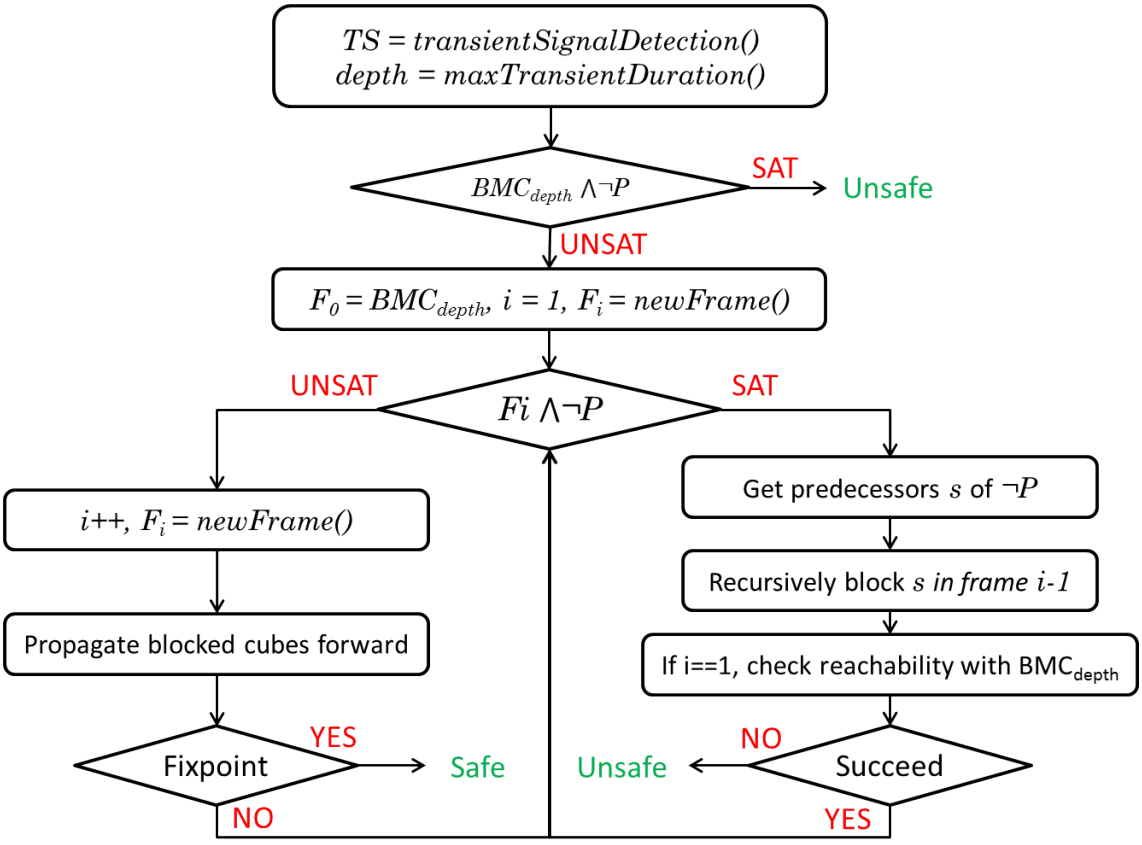


Figure 3.6: Program flow of time-shifted PDR.

doi:10.6342/NTU201601674

Algorithm 4 time-shifted PDR

Input: A transition system $M = \langle I, S, Init, Tr \rangle$ and a property P
Output: a counterexample or a fixpoint

```
1.  (transientSignals, d) ← XSimTillConverge()
2.  BMC_d ← unrollandSimpCircuit(transientSignals, d)
3.  for (i ← 1 to d) do
4.    if (BMC_i ∧ ¬P == SAT) then
5.      return counterexample
6.    end if
7.  end for
8.  F_0 ← BMC_d
9.  F_1 ← newFrame()
10. for (i ← 1 to ∞) do
      // blocking phase
11.   while (F_i ∧ ¬P == SAT) do
12.     cube ← getBadCube(¬P, i)
13.     if (!recBlockCube_TimeShifted(cube, i)) then
10.       return counterexample
11.     end if
12.   end while
      // propagation phase
13.   F_{i+1} ← newFrame()
14.   if (propagateBlockedCubes(i)) then
15.     return fixpoint
16.   end if
17. end for
```

Algorithm 5 $recBlockCube_{TimeShifted}$

Input: A cube to be blocked and a depth i
Output: True or False

```
1.  if (i == 1) then
2.    if (BMC_d ∧ Tr_Simp ∧ cube' == UNSAT) then
3.      cube ← generalize(cube, 1)
4.      cube ← checkExistInitial(cube)
5.      addBlockedCube(cube, 1)
6.      return True
7.    else
8.      preCube ← getBadCube(cube, 0)
9.      return False
10.   end if
11. end if
12. while (F_{i-1} ∧ ¬cube ∧ Tr_Simp ∧ cube' == SAT) do
13.   preCube ← getBadCube(cube, i − 1)
14.   if (!recBlockCube_TimeShifted(preCube, i − 1)) then
15.     return False
16.   end if
17. end while
18. cube ← generalize(cube, i)
19. cube ← checkExistInitialInBMC(cube)
20. addBlockedCube(cube, i)
21. return True
```


doi:10.6342/NTU201601674

To perform time-shifted PDR, we first decide a parameter *decompositionDepth*, to determine how many times the circuit should be unrolled, this parameter can either be decided by the maximum transient duration of the transient signals calculated with ternary simulation to primary inputs until convergence, or be decided manually. But in the second case we have to be careful that not all transient signal can be used to simplify the circuit.

Next, we must perform BMC up to time-step *decompositionDepth* to ensure the correctness of time-shifted PDR.

Then, like original PDR, time-shifted PDR maintains a sequence $\langle F_0, F_1, \ldots, F_k \rangle$ of over-approximations to reachable states which follows the five invariant:

1. $F_0 = R_d$ (where R_d is the d-step exact reachability).

2. All frames F_i except F_0 are a set of clauses. (F_0 is an unrolled circuit)

3. The clauses in F_{i+1} is a subset of F_i for $i > 0$, hence, every state in F_i is also in F_{i+1} $(F_i \Rightarrow F_{i+1})$

4. F_{i+1} is the over approximation of $F_i \wedge Tr$, the set of states reachable in one transition from F_i $(F_i \wedge Tr \Rightarrow F_{i+1})$

5. Every frame implies the property under verification except for the last frame $(F_i \Rightarrow P$ except for $F_N)$.

These 5 invariant establish the correctness of time-shifted PDR, we will show it in section 3.4.

Then, the time-shifted PDR goes into basically same blocking phase and refining phase like origin PDR, except 2 different things:

1. Using different solver to check if $F_0 \wedge \neg cube \wedge Tr \wedge cube'$ is SAT when timeframe is 0.

doi:10.6342/NTU201601674

2. The existInitial function is modified so that we make sure every cube will not contain the set of states in R_{d_exact}. That is, the reachability represented by each frames will not exclude the state in F_0, maintaining the key invariant in original $F_0 \Rightarrow F_k$, $k>0$.

The original PDR keeps the invariant that each frame stands for the set of state that reachable in k-step, and the $R_0 \Rightarrow R_1$, $R_1 \Rightarrow R_2$.... But in real cases, due to the initialization sequence there are many states that can only be reached in R_1, R_2…, and will never be reached again in later timeframes. But by the imply relation of each frames, the original algorithm of PDR has to keeps these states in later frames R_3, R_4, R_5… , making the quality of cubes decreases.

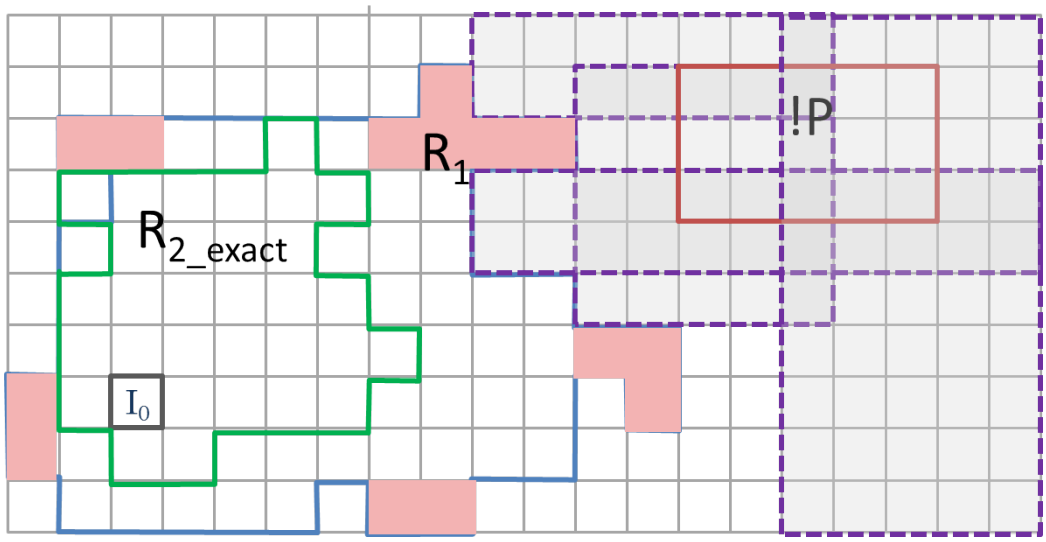


Figure 3.7: Illustration of those 1-time-reached states in R_1,

causes the R_2 to be too coarse as well.

These new 5 invariant allows the time-shifted PDR to escape the R_0, R_1, R_2…, it directly refine these over-approximations to reachable states from R_d, to avoid the reachability to contain too much unnecessary states.

We performed a series of experiments with different decomposition depth, the experimental result proves that this idea work, the time-shifted PDR solved many

doi:10.6342/NTU201601674

unsolved cases and gains a 3.8x runtime usage in common cases. Detail result and discussions will be presented in chapter 4.

3.3.1 Combine with transient signals

By the transient signal detection algorithm, we find that 541/802 cases in the benchmark suits have transient signal, 150 of them have transient signals higher than 2% of the latch size. We also performed an experiment that runs time-shifted PDR on the circuit that simplified by these transient signals. The *decompositionDepth* is decided by the *maxTransientDuration* found before. The result of experiment is significantly good, solved 29 cases among 72 unsolved cases. And for the common solved cases, the run time also decreases a lot.

3.3.2 Deep-Shift

We also tried a version of deep shift PDR, that is, runs BMC for 10 second to get a deep safe duration (like 30~50). Then we apply a very deep unrolling circuit to represent the initial states. However, the result shows that the runtime deteriorates severely, the overall runtime was wasted by the lengthy BMC check of R0, but the benefit of starting refining the reachability from a deep time frames seems has no prominent effect.

3.4 Multi-Step PDR

With the same intuitive thought, we want to avoid wrong cube blocking decision made in huge decision tree, and take advantage of the disproving power of BMC not

doi:10.6342/NTU201601674

only for the initial states. So we proposed a novel algorithm, called multi-step PDR, directly using BMC in the solving process of PDR.

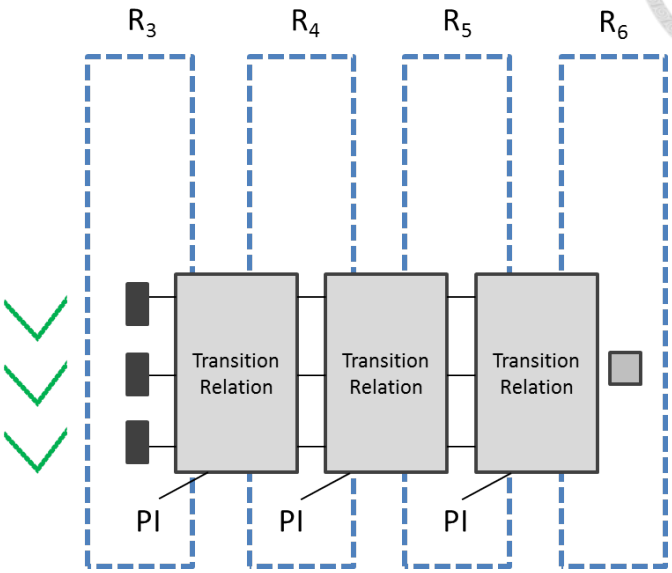


Figure 3.8: Using a BMC model to aggressively find proof-obligation in a deeper frame.

Algorithm 6 $recBlockCube_{MultiStep}$

Input: A cube to be blocked and a depth i
Output: True or False

$d \leftarrow UnfoldingDepth$
1. **if** $(i == 0)$ **then**
2. **return** $False$
3. **end if**
1. **if** $(i \geq d)$ **then**
4. **while** $(F_{i-d} \wedge \neg cube \wedge BMC_d \wedge cube' == SAT)$ **do**
5. $preCube \leftarrow getBadCube(cube, i-d)$
6. **if** $(!recBlockCube_{MultiStep}(preCube, i-d))$ **then**
7. **return** $False$
8. **end if**
9. **end while**
3. **end if**
4. **while** $(F_{i-1} \wedge \neg cube \wedge Tr \wedge cube' == SAT)$ **do**
5. $preCube \leftarrow getBadCube(cube, i-1)$
6. **if** $(!recBlockCube(preCube, i-1))$ **then**
7. **return** $False$
8. **end if**
9. **end while**
10. $cube \leftarrow generalize(cube, i)$
11. $addBlockedCube(cube, i)$
12. **return** $True$

doi:10.6342/NTU201601674

As the original PDR, multi-step PDR also keeps a sequence ⟨F0,F1,...,Fk⟩ of over-approximations to reachable states which follows the original five invariant in original PDR. The most different part is the process of recursive block cube.

Again we must decide an unfolding depth d, after PDR created d frames, the multi-step PDR starts aggressively to use an unfold BMC model to get a deeper proof-obligation in earlier time frame, this help multi-step PDR to get the correct proof-obligation faster, avoiding the the lengthy iteratively decision process in original PDR.

We also performed a series of experiments with different unfolding depth to look for the best

3.4.1 Activation condition

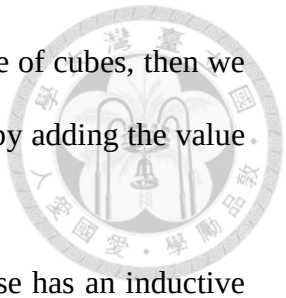
If this aggressively check is UNSAT, to keep the imply relation for correctness in PDR, nothing can be added in any frames. So for some cases, this query would be a huge overhead, we need to set some constraints to prevent the misuse of this aggressive method. An intuitive idea is only to apply this method when PDR stuck in some time frame. However, the result wasn't as good as expect.

3.5 Soundness and Completeness

Here we establish the correctness of time-shifted PDR and multi-step PDR.

Theorem 1. If the circuit stands for the initial states in time-shifted PDR represents R_d (the reachability within d-step), then the algorithm either terminates with a correct inductive invariant or with a concrete counter-example.

doi:10.6342/NTU201601674



Proof: If the time-shifted PDR gives a concrete counter-example of cubes, then we can find the real counter-example trace that starts from initial state, by adding the value of primary input in the unrolling circuit.

Else, the time-shifted PDR terminates with a result that this case has an inductive invariant only when there is a frame i-1 equals to frame i. It means $Ri \wedge Tr \Rightarrow Ri$, and $Ri \Rightarrow P$, by the imply relation, $R_0 \Rightarrow R_d \Rightarrow R_1 \ldots \Rightarrow R_i$, satisfying the condition of inductive invariant.

Theorem 2. The circuit stands for the initial states in time-shifted PDR represents R_{d_exact} (the d-step exact reachability)，and this algorithm is still correct.

Proof：Assume that the time-shifted depth is d, by the bounded model checking of depth d we do before the time-shifted PDR, if the result of this BMC is unsafe, then the overall algorithm terminate with a real counter-example. Else we can know that either there exists a counter-example that depth longer than d, or the case has an inductive invariant.

For the first case, the proof is same as above. And for the second case, if there exists a counter-example that depth longer than d, then it must also exist a state in this counter-example trace that falls into R_{d_exact}. Hence during the procedure of the recursive blocking cubes, the time-shifted PDR must have been find out this cube in the previous time frame, the circumstance in upper case of figure 3.8 will never happen.

doi:10.6342/NTU201601674

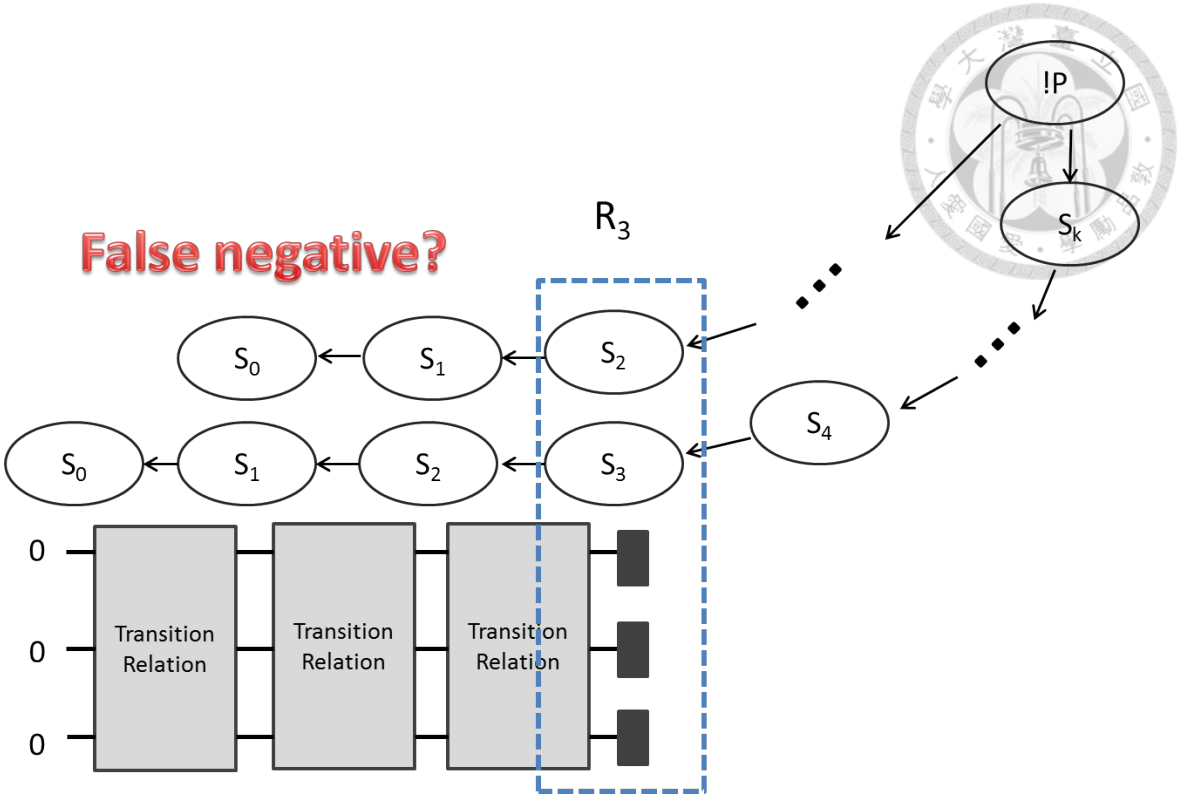


Figure 3.9: The false negative will never happen,

because the algorithm will find out this cube in the previous time frame.

Theorem 3. The multi-step PDR either terminates with a correct inductive invariant or with a concrete counter-example.

Proof: For the inductive case, the proof is as same the original PDR. The only difference is that the multi-step PDR aggressively checks the reachability for some cube in previous d frames, when this BMC query UNSAT, then nothing will be added in the frames. Only when the BMC query SAT, then we can extract the real counter-example trace from the latch of each time frame in the BMC model. So if the recursive blocking process fails, the unsafety of the property is self-evident.

doi:10.6342/NTU201601674

Chapter 4 Experimental Results

In this chapter we give the detail results of experiments and and analysis of data mentioned in Chapter 3.

Our experiment carried out on Ubuntu 14.04 server LTS, the CPU is Intel(R) Core(TM) i7-5960X 3.00GHz. The memory limit is 32GB and time limit is 900 seconds. The testcase suit is the combined from HWMCC11,12,13,14,15 downloaded from HWMCC official website [9], which contains 813 instances. We have implemented our method on the IPDR part of V3 framework [8]. We first run the transient signal extract algorithm and doing statistics on the testcase suit, and then we try different parameter of out proposed algorithms, then compare the result to original PDR algorithm.

4.1 Time-Shifted PDR

Table 4.1: Total result on the different decomposition depth and original PDR

Total 802 cases								
Decomposition depth	1(Original)	2	3	4	5	7	9	VB
Total Proved	329	334	328	325	321	316	313	352
Total Disproved	105	108	111	109	106	111	108	122
Total Solved	434	442	439	434	427	427	421	474
Solved Only	3	0	2	0	2	2	3	
Extra Solve	0	20	23	21	22	24	24	40
Unsolved Only	9	2	1	0	2	1	7	
BMC Solving Time (%)	0	5.27	9.81	14.39	18.42	22.18	27.64	
Average Time (sec.)	77.85	75.22	71.75	74.03	74.72	74.99	76.27	65.01
Average Time on common cases(sec.)	41.59	42.20	41.96	42.98	47.05	49.56	59.34	23.68

For the time-shifted PDR we can see that the number of total solved case varies with decomposition depth, the longer the depth is, the less cases solved. The depth 2 (unroll one circuit to represent initial states) can solve most cases, but depth 3 uses the least average runtimes. The total disprove (unsafe cases) increase in all decomposition

doi:10.6342/NTU201601674

depth. The time-shifted PDR of depth 2 and depth 3 can solve 442 and 439 cases more then Original PDR and uses also fewer running times. The trend is that the larger depths solve fewer cases than original method, however, this is not absolute. The runtime also decreases with the larger depths in some cases.

But note that these methods can still solve some cases other method cannot solve. Therefore, if we have multiple core and we can run these different depths in multithread, then we can reach the virtual best, which has solved 40 more cases than original PDR.

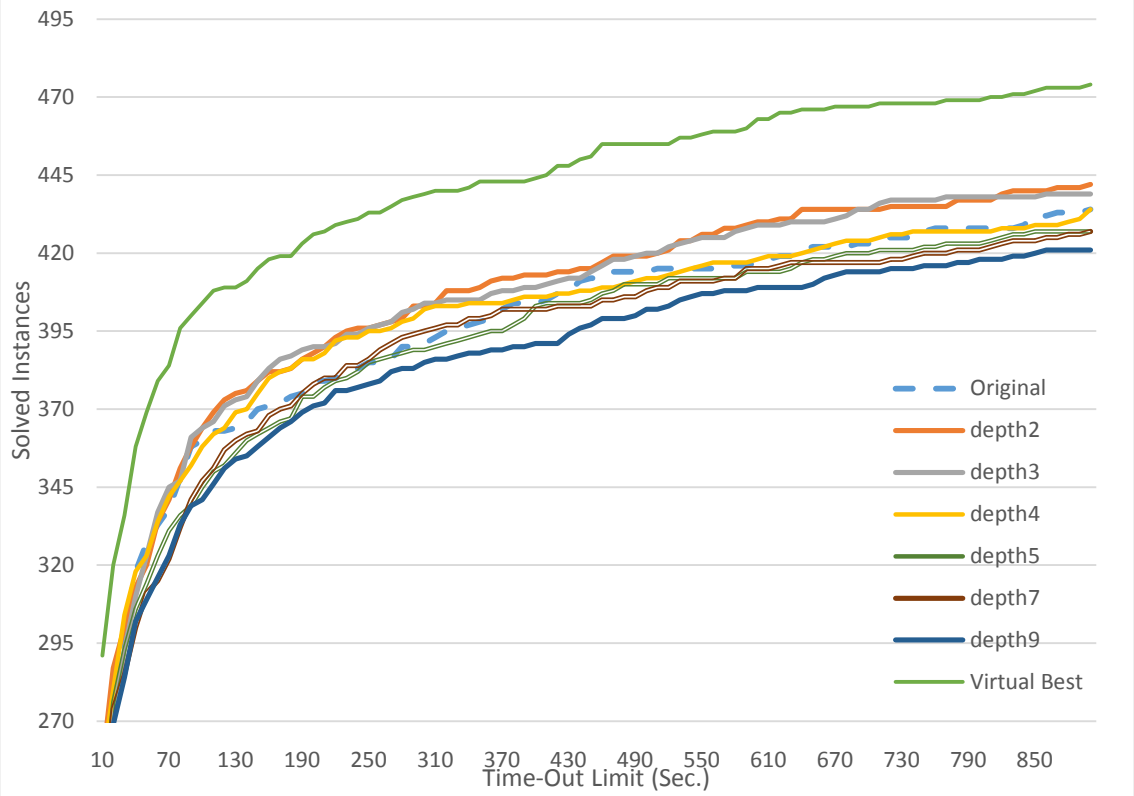


Figure 4.1: Comparison of different decomposition depth in solved cases.

The result shows that depth 3 gives the best result for time-shifted PDR, it solves 8 cases that was unsolved by originally PDR, and uses fewer runtime in commonly solved cases. It means that directly refining reachabilities from R_3 does help PDR to solve more cases.

doi:10.6342/NTU201601674

We found another prominent statistic, the runtime of unsafe cases improved a lot by our time-shifted PDR algorithm. But runtime of the safe cases deteriorates.

Table 4.2: Comparison of 3-time-shifted PDR and original PDR on unsafe cases

Case	Original	Depth=3	Ratio
6s301rb527	85.10	23.98	3.55
6s380b511	27.51	27.95	0.98
bc57sensorsp0	741.10	533.82	1.39
bc57sensorsp1	744.54	447.25	1.66
bc57sensorsp1neg	369.38	99.59	3.71
bc57sensorsp2neg	315.65	271.45	1.16
bc57sensorsp3	432.83	153.98	2.81
beemadd3b1	80.73	45.55	1.77
beemandrsn7f1	838.86	13.71	61.20
beembrptwo4b1	37.77	31.92	1.18
beemextnc3b1	365.53	11.61	31.49
beemfrogs1b1	895.00	521.98	1.71
beemfrogs1f1	362.39	36.04	10.06
beemldelec4b1	157.40	70.93	2.22
beemlifts8b1	71.64	272.17	0.26
beemtrngt3f1	293.30	62.86	4.67
bob9234spec4neg	12.89	13.99	0.92
bob9234spec5neg	36.45	53.34	0.68
bob9234spec6neg	71.48	41.73	1.71
bobsynth06neg	191.31	317.76	0.60
bobsynth07neg	59.26	115.07	0.51
bobsynth08neg	301.76	184.29	1.64
bobsynth11neg	32.26	46.51	0.69
bobsynth12neg	35.90	11.53	3.11
bobsynth13neg	35.05	59.10	0.59
bobsynthetic2	79.10	104.22	0.76
bobsynthetic	12.05	18.01	0.67
csmacdp0	20.23	36.29	0.56
csmacdp2neg	130.75	28.43	4.60
irstdme4	264.17	25.38	10.41
mentorbm1and	195.63	687.45	0.28
mentorbm1p11	633.60	668.58	0.95
mentorbm1p12	143.06	47.52	3.01
nusmvtcastp5	17.59	10.65	1.65
oski3ub1i	325.04	354.00	0.92

doi:10.6342/NTU201601674

oski3ub3i	346.67	387.93	0.89
oski3ub4i	612.75	483.35	1.27
oski3ub5i	755.86	572.86	1.32
prodcellp0neg	37.45	58.44	0.64
prodcellp1	39.08	120.13	0.33
prodcellp1neg	46.53	54.73	0.85
prodcellp2	47.15	89.37	0.53
prodcellp2neg	47.13	43.67	1.08
prodcellp3	53.00	63.33	0.84
prodcellp4	78.04	67.13	1.16
prodcellp4neg	45.28	64.69	0.70
Sum. / Avg. ratio	10525.22	7454.23	3.78

This result makes senses because that when we make the reachability more precise, each cube will become large, so it will be less easy to propagate that cube to next time frames, hence the runtime of finding the inductive invariant will be longer. However, for the safe cases, the overall runtime is 14213.78s / 15887.92s for the original/time-shifted PDR. And in our benchmarks suit the number of unsafe/safe case ratio is around 1:3, so even for this single core method, it still improves PDR in overall runtimes, and we can conclude that the phenomenon that the first few frames have the too coarse reachability is common in these industrial, and the time-shift PDR does help to improve this phenomenon.

4.1.1 Combine with transient signals

Table 4.3: Comparison of time-shifted PDR with original PDR

on cases that have transient signals ratio > 1/50

Case	Original	With TSs	depth	ratio
6s136	timeout	0.11	11	
6s160	timeout	55.97	48	
6s164	45.67	3.82	10	11.94
6s197	timeout	5.52	19	
6s198	timeout	19.10	19	

doi:10.6342/NTU201601674

6s206rb025	31.55	3.14	46	10.04
6s206rb103	57.84	3.13	46	18.47
6s207rb16	timeout	52.43	17	
6s207rb28	timeout	40.43	17	
6s209b0	12.04	0.27	22	45.25
6s209b1	timeout	214.49	22	
6s215rb0	4.92	1.49	35	3.30
6s216rb0	3.81	3.38	35	1.13
6s221rb14	timeout	8.75	43	
6s221rb18	8.03	8.78	43	0.92
6s273b11	271.68	445.35	44	0.61
6s273b37	178.06	196.84	44	0.90
6s275rb253	6.22	7.08	5	0.88
6s275rb318	6.38	6.84	5	0.93
6s276rb318	8.07	7.41	9	1.09
6s276rb342	6.54	5.90	9	1.11
6s277rb292	7.87	7.47	9	1.05
6s277rb342	6.47	5.79	9	1.12
6s282b15	78.44	88.59	17	0.89
6s288r	71.37	108.18	22	0.66
6s301rb106	timeout	162.12	49	
6s301rb527	85.10	141.71	49	0.60
6s302rb09	timeout	3.00	46	
6s305rb069	337.51	0.48	17	703.53
6s305rb108	timeout	0.48	17	
6s306rb03	72.04	0.49	18	147.26
6s306rb23	36.56	0.49	18	75.11
6s307rb06	timeout	1.28	49	
6s307rb09	timeout	0.67	49	
6s321b1	timeout	50.07	34	
6s321b5	timeout	2.97	34	
6s326rb02	10.24	0.08	11	136.43
6s326rb08	6.22	0.07	11	82.99
6s327rb10	2.52	0.07	10	36.58
6s330rb06	5.39	0.51	9	10.47
6s330rb11	1.41	0.65	9	2.16
6s344rb150	143.42	0.69	18	208.78
6s358r	timeout	0.13	7	
6s359r	timeout	0.11	7	
6s362rb0	timeout	240.27	50	
6s372rb26	1.15	0.10	21	11.05
6s372rb31	70.57	0.13	21	539.63
6s374b029	timeout	118.85	23	
6s374b114	84.19	516.63	23	0.16

doi:10.6342/NTU201601674

6s381rb051	217.11	564.61	27	0.38
6s381rb630	8.83	26.24	27	0.34
6s384rb024	6.32	1.47	51	4.30
6s384rb194	timeout	1.46	51	
6s385rb267	449.10	0.76	37	591.92
6s385rb444	timeout	0.76	37	
6s386rb07	timeout	1.31	22	
6s386rb08	timeout	1.39	22	
6s52	timeout	14.99	19	
6s6	51.03	43.93	19	1.16
beemandrsn7f1	838.86	198.49	4	4.23
beemat1f1	timeout	58.33	2	
beembrdg2f1	timeout	654.26	6	
beembrp1f1	278.65	239.36	9	1.16
beembrptwo1f2	34.64	108.28	17	0.32
beembrptwo5f2	timeout	21.88	17	
beemexit5f1	33.72	12.04	9	2.80
beemfish4f1	12.70	18.19	2	0.70
beemfrogs1f1	362.39	79.31	3	4.57
beemlann2f1	373.65	37.62	3	9.93
beemlmprt5f1	7.69	19.72	9	0.39
beemlptna5f1	38.97	197.52	9	0.20
beemmcs1f1	9.71	1.47	3	6.58
beemndhm2f2	41.82	73.63	7	0.57
beemskbn2f1	timeout	620.26	3	
beemsnpse4f1	26.42	61.42	9	0.43
beemszmsk1f1	465.26	141.35	9	3.29
beemtlphn5f1	196.51	163.03	3	1.21
beemtrngt3f1	293.30	242.60	9	1.21
boblivear	21.21	1.43	6	14.87
bobsm5378d2	6.59	0.10	26	64.53
eijks5378	61.32	323.53	25	0.19
mentorbm1and	195.63	29.22	40	6.70
mentorbm1p00	3.69	668.92	40	0.01
mentorbm1p01	4.04	39.16	40	0.10
mentorbm1p04	1.36	478.12	40	0.00
mentorbm1p05	45.93	694.13	40	0.07
mentorbm1p08	7.21	293.74	40	0.02
mentorbm1p10	timeout	34.20	40	
mentorbm1p11	633.60	36.33	40	17.44
mentorbm1p12	143.06	42.53	40	3.36
neclabakery001	4.49	18.72	9	0.24
neclaftp3001	1.17	2.70	8	0.43
neclaftp3002	1.29	3.17	8	0.41

doi:10.6342/NTU201601674

nusmvbrp	1.09	0.01	4	150.39
pdtpmsmiim	1.72	1.55	12	1.11
pdtpmsvsar	13.18	15.35	8	0.86
pdtvisbakery0	5.97	9.47	7	0.63
pdtvisbakery1	7.06	11.27	7	0.63
pdtvisbakery2	14.33	5.98	7	2.40
pdtvissoap1	2.10	3.21	8	0.65
pdtvissoap2	2.58	1.80	8	1.44
pdtvsar8multip05	2.01	13.70	8	0.15
pdtvsar8multip07	3.53	21.45	8	0.16
pdtvsar8multip10	2.14	15.28	8	0.14
pdtvsarmultip32	53.45	27.62	8	1.94
prodcellp0neg	37.45	0.25	22	152.64
prodcellp3	53.00	0.18	19	291.96
prodcellp4	78.04	0.19	19	412.47
prodcellp4neg	45.28	0.22	19	205.06
visbakery	2.52	0.01	4	336.52
Sum. (s)	6863.93	6555.90		
Sum. (s) with decomposition depth < 20	4162.41	1927.67		

After the ternary simulation, there are 541 cases have transient signals in 802 cases. We pick 159 of them that have transient signals higher than 2% of the latch size.

We run time-shifted PDR on these circuit that simplified by their transient signals, and do further analysis because the runtime of the other case will not be changed. The result show that the effect of simplification is significantly good. It solves 29 cases among 72 originally unsolved cases. And for the commonly solved cases, it also uses a slightly fewer runtime. However, by the former experiment, we know that the too high decomposition depth will burden PDR with the too long unrolling circuit in the frame 0. So we collected the runtime of the cases that *decompositionDeth* less than 20, and find in this condition, the simplified circuit uses only half runtime in the commonly solved cases.

doi:10.6342/NTU201601674

We can explain this by that for the cases that *decompositionDeth* less than 20, the benefit of simplify circuit with transient signals still surpasses the burden of too long BMC circuit. So if we got a circuit that the maxTransitionDuration is larger than 20, than we better not to do the temporal decomposition, just uses original PDR.

4.1.2 Forward Reachability Constraint

Table 4.4: Total result of Forward Reachability Constraint and time-shifted PDR

Method	1(Original)	2	3	FRC	VB
Total Proved	329	334	328	312	345
Total Disproved	105	108	111	99	115
Total Solved	434	442	439	420	470
Solved Only	5	1	4	9	
Extra Solve	0	20	23	22	36
Unsolved Only	9	3	6	28	
Average Time (sec.)	77.85	75.22	71.75	83.20	66.24
Average Time on common cases(sec.)	53.38	50.51	54.42	58.86	29.29

The time-shifted PDR is to use the BMC model to represent the initial states, while this method is to add the cubes get by ternary simulations in first few frames to refine the reachability. We can see that both the number total of solved cases as well as the average runtime is worse than original PDR, however, it solves 9 unique cases that it not solved by other method, proves that this method still works is some cases.

doi:10.6342/NTU201601674

4.2 Multi-Step PDR

Table 4.5: Total Result on the different unfolding depth and original PDR

Total 802 cases								
Unfolding depth	1(Original)	2	3	4	5	7	9	VB
Total Proved	331	328	320	317	315	306	305	343
Total Disproved	104	100	98	96	103	98	100	111
Total Solved	435	428	418	413	418	404	405	454
Solved Only	6	1	2	0	2	1	0	
Extra Solve	0	6	9	10	11	5	5	19
Unsolved Only	1	1	1	2	1	6	5	
Average Time (sec.)	71.49	80.41	69.88	78.95	88.87	87.43	87.83	69.38
Average Time on common cases(sec.)	33.95	37.51	40.33	49.12	49.61	60.16	64.78	23.00

For the multi-step PDR, we can see that the number of total solved case also varies with decomposition depth, the original PDR solves most cases, but the depth 3 (finding proof-obligation in previous 3 time frames) uses least times. The other depths are not very effective. However, these methods can still solve some cases that other method cannot solve. Therefore, in the multi-core environment, there methods can still 19 solve cases that was unsolved originally.

doi:10.6342/NTU201601674

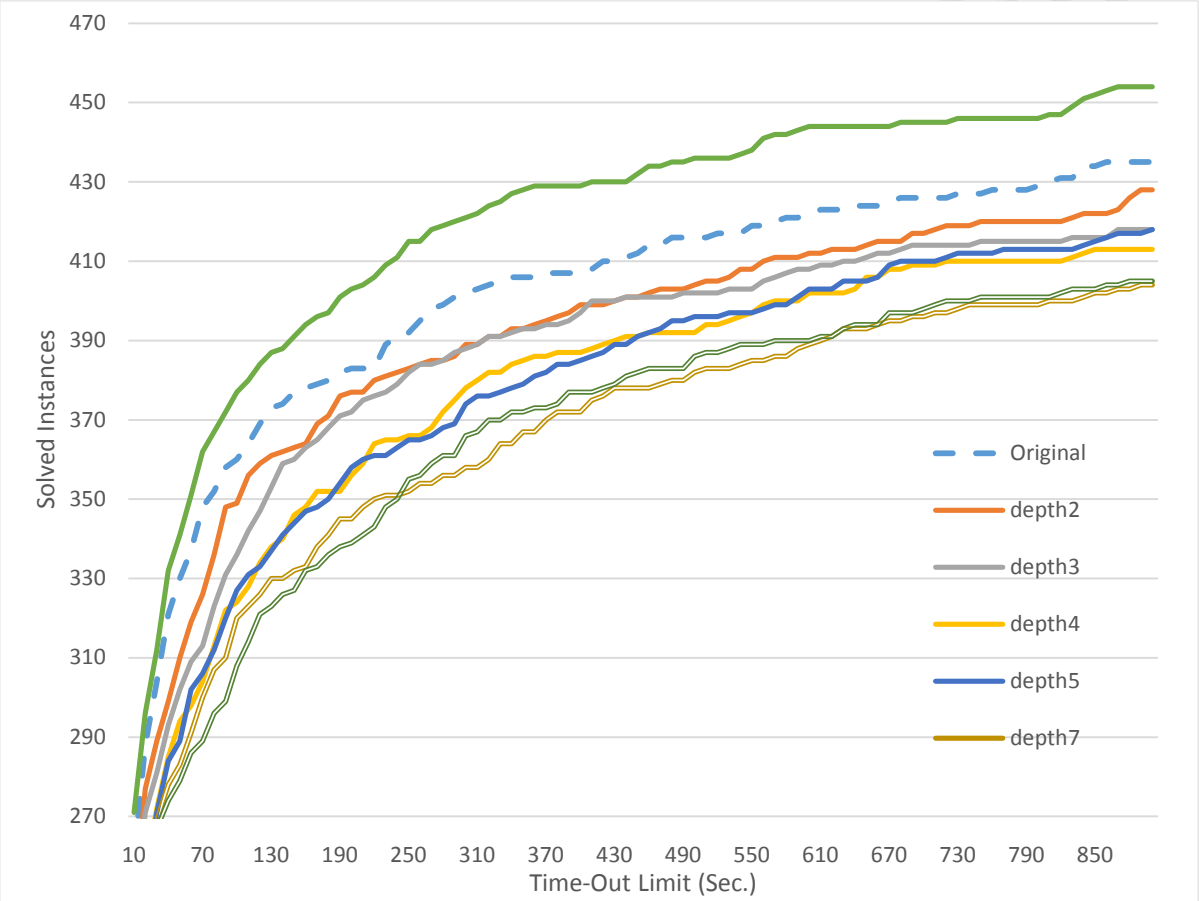


Figure 4.2: Comparison of different unfolding depth in solved cases.

The result was not as effective as time-shifted PDR, but the virtual best still gains 19 solved cases and uses much fewer times, means that these methods compliment original PDR on a large set of cases.

We also tried another interesting idea: check the current proof-obligation in previous d time frames, but we found that there are no gains in runtime at all by this method.

doi:10.6342/NTU201601674

Chapter 5 Conclusion and Future Work

In this thesis, we implemented the linear time transient signal detection algorithm mentioned in [15] into V3, collected statistics of the existence and frequency that transient signals occur in HWMCC benchmark suits. And we fully utilized the information of these signals, we proposed several alternative heuristics to improve PDR, the forward reachability constraint that simply add cubes obtained by the transient signals, the time-shifted PDR that unrolling first few time frames of circuit to be the new initial state and the multi-step PDR that using a BMC model to search proof obligation in recursive blocking phase.

In the experiment results we can see that these information, along with combining the concept of temporal decomposition into PDR really helps to solve some cases by changing the way the algorithm blocks the reachability. The time-shifted PDR further improves the performance of PDR, solves more cases than original PDR with 0.7x runtime in overall cases, because it can help PDR to escape from blocking some too coarse over-approximation of first few time frames with numerous cubes. All method solves some unique cases that were not able to be solved by original PDR, providing a great variability to original PDR. In modern verification environments multithread technique is usually available, hence we can run all these methods together with original PDR and get the best improvement.

For this thesis, there are still some points in our design that can be further improved. Firstly, the parameter decompostitionDepth is pre-decided in both time-shifted PDR and multi-step PDR. For the experimental result, we can find that the fastest runtime occurs in different depth in different cases. If multi-core environment isn't available, it needs a reasonable heuristic to decide what decompostitionDepth should be

doi:10.6342/NTU201601674

applied in different cases. Also, dynamically apply different depth during recursive blocking is also possible. For example, in time-shifted PDR, start with certain level of unrolling circuit, changing the decomposition depth during some point of the process and uses the reachability info in later time frames. In multi-step PDR, we can dynamically query the reachability of some cube in any d time frames earlier at our will. These parameters can be changed dynamically according to the solving situation at the moment to get higher performance.

Second, we focus on aggressively searching for the right counter-example path earlier, but in the safe cases, there is no right counter-example path at all. These heuristic we tried are more like an overhead and do little things to help PDR to find an inductive invariant, the experimental result also shows that out approach performs better on the unsafe cases. Alternatively, [11] proposed a method to improve the PDR on the safe cases, for the future works, it seems to be promising to combine our method with theirs.

Third, the experimental result of multi-step PDR is not good enough. If available, we will design other heuristic of the activation conditions.

Finally, this thesis only focuses on the effect of temporal decomposition. If applicable, we hope that these methods can be combined with other modern techniques of formal verification, like the abstraction, interpolation… etc., hoping to push the limits of the formal verification.

doi:10.6342/NTU201601674

REFERENCE

[1] Clarke, Edmund M., Orna Grumberg, and Doron Peled. *Model checking*. MIT press, 1999.

[2] McMillan, Kenneth L. "Symbolic model checking." *Symbolic Model Checking*. Springer US, 1993. 25-60.

[3] Sheeran, Mary, Satnam Singh, and Gunnar Stålmarck. "Checking safety properties using induction and a SAT-solver." *International conference on formal methods in computer-aided design*. Springer Berlin Heidelberg, 2000.

[4] McMillan, Kenneth L. "Interpolation and SAT-based model checking."*International Conference on Computer Aided Verification*. Springer Berlin Heidelberg, 2003.

[5] Bradley, Aaron R. "SAT-based model checking without unrolling." *International Workshop on Verification, Model Checking, and Abstract Interpretation*. Springer Berlin Heidelberg, 2011.

[6] Een, Niklas, Alan Mishchenko, and Robert Brayton. "Efficient implementation of property directed reachability." *Formal Methods in Computer-Aided Design (FMCAD), 2011*. IEEE, 2011.

[7] Brayton, Robert, and Alan Mishchenko. "ABC: An academic industrial-strength verification tool." *International Conference on Computer Aided Verification*. Springer Berlin Heidelberg, 2010.

[8] Cheng-Yin Wu and Chung-Yang (Ric) Huang. “V3: An extensible framework for hardware verification”, https://github.com/chengyinwu/V3.

[9] Hardware model checking competition. http://fmv.jku.at/hwmcc/.

doi:10.6342/NTU201601674

[10] Fan, Kuan, Ming-Jen Yang, and Chung-Yang Huang. "Automatic abstraction refinement of TR for PDR." *2016 21st Asia and South Pacific Design Automation Conference (ASP-DAC)*. IEEE, 2016.

[11] Ivrii, Alexander, and Arie Gurfinkel. "Pushing to the top." *Proceedings of the 15th Conference on Formal Methods in Computer-Aided Design*. FMCAD Inc, 2015.

[12] Hong-Syun Jiang and Chung-Yang (Ric) Huang. “Enhancing Property Directed Reachability Technique through Cube Analysis.” Master Thesis. National Taiwan University, 2015.

[13] Vizel, Yakir, and Arie Gurfinkel. "Interpolating property directed reachability."*International Conference on Computer Aided Verification*. Springer International Publishing, 2014.

[14] Backes, John D., and Marc D. Riedel. "Using cubes of non-state variables with property directed reachability." *Proceedings of the Conference on Design, Automation and Test in Europe*. EDA Consortium, 2013.

[15] Case, Michael L., et al. "Enhanced verification by temporal decomposition."*Formal Methods in Computer-Aided Design, 2009. FMCAD 2009*. IEEE, 2009.

doi:10.6342/NTU201601674