The Dissertation Committee for Hari Mony
certifies that this is the approved version of the following dissertation:

# Sequential Redundancy Identification using Transformation-Based Verification

Committee:

Adnan Aziz, Supervisor

Jason Baumgartner

J Strother Moore

Yale N. Patt

Gustavo de Veciana

# Sequential Redundancy Identification using Transformation-Based Verification

by

**Hari Mony, B.Tech.; M.S.E.**

**Dissertation**

Presented to the Faculty of the Graduate School of

The University of Texas at Austin

in Partial Fulfillment

of the Requirements

for the Degree of

**Doctor of Philosophy**

**The University of Texas at Austin**

May 2008

To my Appa and Amma and my wife Suchitra

# Acknowledgments

This dissertation would not have been possible without the influence of many exceptional individuals, including my teachers, collaborators, family and friends. I would like to take this opportunity to acknowledge their influence in shaping my work and thank them.

First of all, I would like to thank my adviser, Adnan Aziz. I was introduced to the insightful world of formal verification when I took the Formal Verification class taught by Adnan in 1999. I am indebted to Adnan for helping develop my formal reasoning skills through his classes and class projects. Adnan always encouraged me to "think outside the box" and I would like to express my deepest thanks for his guidance and support.

The lion's share of credit for this dissertation goes to Jason Baumgartner. At a time when I was not really sure whether I wanted to pursue a Ph.D. degree, Jason taught me the following maxim, "Do not try to be anything but what you are, and try to be that perfectly". I was inspired by Jason's passion for formal verification and his drive for perfection and excellence. Without Jason's constant encouragement, insightful critique and active involvement, this work would not have been possible. It has been my privilege to collaborate and develop ideas with him.

I would like to thank J Strother Moore, Yale Patt, and Gustavo de Veciana for serving in the dissertation committee and reviewing this dissertation.

I wish to acknowledge and thank my co-authors, Adnan Aziz, Jason Baumgartner, Viresh Paruthi, Robert Kanzelman, Geert Janssen, Andreas Kuehlmann, Rick Seigler, and Gary Van Huben for the stimulating discussions and insightful comments.

I have been fortunate to learn from several outstanding professors during my studies at UT: Adnan Aziz, Yale Patt, Craig Chase, Lorenzo Alvisi, Jacob Abraham, Martin Wong, Mootaz Elnozahy, Vijay Garg, Nur Touba, Scott Nettles, and Takis Konstantopoulos. I wish to acknowledge and thank them.

I next wish to acknowledge the SixthSense team and the Verification building block (VBB) management team at IBM. The techniques described in this dissertation have been implemented and tested within IBM's (semi)-formal and sequential-equivalence checking tool SixthSense. I would not have been able to easily prototype my research ideas and quickly obtain experimental results without the SixthSense tool. The SixthSense team is the best tool development team one could wish to be part of. I wish to acknowledge and thank the past and present members of SixthSense team: Jason Baumgartner, Viresh Paruthi, Robert Kanzelman, Mark Williams, Geert Janssen, Jiazhao Xu, Yee Ja, Gavin Meil, Fadi Zaraket, and Mike Case. I also wish to acknowledge and thank the management team at IBM, especially David King, Wolfgang Roesner, Hina Mehta and Victor Rodriguez for their support and encouragement and for approving the funding of my education through the Degree Work and Study program at IBM.

I additionally wish to thank the users of SixthSense tool at IBM who provided me with examples that motivated several of my contributions in this dissertation. I wish to acknowledge Paul Roessler, Jun Sawada, Christian Jacobi, Kai Weber, Rick Seigler, Gary Van Huben, Tilman Gloekler, Barinjato Ramanandray, Tobias Gemmeke, Udo Krautz, Mark Firstenberg, Balazs Sallay and Seth Wright.

Last, but not the least, I wish to acknowledge those who have influenced my life during this period. My family has been the source of much of my success. Doing research work as a married Ph.D. student while being employed in an extremely time-consuming full-time job is very stressful. Without my beloved wife Suchitra's sacrifices as well as her love, motivation, constant encouragement and support, this experience would not have been smooth and enjoyable.

I owe my passion for science to my Appa, Subramony Hariharaiyer and my drive to achieve to my Amma, Jayalakshmi Mony. The love of my Appa and Amma has always been an integral part of my life and this dissertation would not have been possible without their love and immense personal sacrifices. My sisters, Asha and Chitra were always there for me and always encouraged me to pursue my dreams.

My memories of graduate studies will always be defined by the friends I made at UT Austin. They include Krishna, Sandip-da, Arindam, Arnab, Dwip-da, Shovan-da, Sreangsu, Imtiaz, Jayanta-da, Vikas, Sugato, Manoj, and Kathy. My old DTW wingmates at Nehru Hall in IIT Kharagpur always made an effort to keep my spirits up whenever I felt down and out. I also wish to acknowledge my friends who were great sources for stress relief for the last couple of years: Jinson, Himyanshu, Mrinal, Raghu, and Sunitha. I am grateful to all my friends for their support.

HARI MONY

*The University of Texas at Austin*
*May 2008*

# Sequential Redundancy Identification using Transformation-Based Verification

Publication No. _____

Hari Mony, Ph.D.
The University of Texas at Austin, 2008

Supervisor: Adnan Aziz

The design of complex digital hardware is challenging and error-prone. With short design cycles and increasing complexity of designs, functional verification has become the most expensive and time-consuming aspect of the digital design process. Sequential equivalence checking (SEC) has been proposed as a verification framework to perform a true sequential check of input/output equivalence between two designs. SEC provides several benefits that can enable a faster and more efficient way to design and verify large and complex digital hardware. It can be used to prove that micro-architectural optimizations needed for design closure preserve design functionality, and thus avoid the costly and incomplete functional verification

regression traditionally used for such purposes. Moreover, SEC can be used to validate sequential synthesis transformations and thereby enable design and verification at a higher-level of abstraction. Use of sequential synthesis leads to shorter design cycles and can result in a significant improvement in design quality. In this dissertation, we study the problem of sequential redundancy identification to enable robust sequential equivalence checking solutions. In particular, we focus on the use of a transformation-based verification framework to synergistically leverage various transformations to simplify and decompose large problems which arise during sequential redundancy identification to enable an efficient and highly scalable SEC solution.

We make five main contributions in this dissertation. First, we introduce a novel sequential redundancy identification framework that dramatically increases the scalability of SEC. Second, we propose the use of a flexible and synergistic set of transformation and verification algorithms for sequential redundancy identification. This more general approach enables greater speed and scalability and identifies a significantly greater degree of redundancy than previous approaches. Third, we introduce the theory and practice of transformation-based verification in the presence of constraints. Constraints are pervasively used in verification testbenches to specify environmental assumptions to prevent illegal input scenarios. Fourth, we develop the theoretical framework with corresponding efficient implementation for optimal sequential redundancy identification in the presence of constraints. Fifth, we address the scalability of transformation-based verification by proposing two new structural abstraction techniques. We also study the synergies between various transformation algorithms and propose new strategies for using these transformations to enable scalable sequential redundancy identification.

# Contents

# List of Tables

# List of Figures

# Chapter 1

# Introduction

Computers have become irreplaceable components of almost every facet of modern life. They are central components of communication, medical and entertainment devices, and computing systems control banking, manufacturing, transportation, and many other operations. The design of complex digital hardware is challenging and error-prone. The digital design process takes place in stages, starting from an abstract specification and proceeding to a concrete implementation.

The first stage of the digital design process is the *architecture* stage, where the behavior of the design is quantitatively described. The desired functionality and performance is used to determine the type and number of logic components needed to realize the design. The next stage is the *implementation* phase, where the cycle-to-cycle behavior of the design is modeled in a hardware description language based on the architecture level specification. Implementation is often done at the RT-level specifying the placement of state elements. *Logic synthesis* is then applied to refine the RT-level models to gate-level models, which are further refined down to transistor-level models. The last stage involves *physical design*, where the design is specified as a set of geometric objects corresponding to electrical devices etched on silicon.

*Verification* is the process of validating that the design conforms to its specification and is performed at various stages of the digital design process. With short design cycles and increasing complexity of designs, functional verification

has become the most expensive and time-consuming phase of the digital design process [Int04]. The complexity of functional verification is exacerbated by the iterative nature of the design process. For example, a failure to achieve timing, area, power, testability and signal integrity goals (collectively known as design closure) would necessitate redesign/optimization at the architecture or RT-level and another iteration of the digital design process. The optimizations at the RT-level often result in sequential changes to datapath and control, and can cause a sizable impact on functional verification. Due to lack of a reliable method to validate that changes needed for design closure do not alter functionality, such changes often require time-consuming and incomplete regression of the functional verification process.

The regression of the functional verification process can be avoided if the sequential optimizations to achieve design closure can be done through validated logic synthesis transformations. Use of sequential transformations such as retiming [LS91], state re-encoding [Koh78], state minimization [Koh78], and redundancy removal using unreachability-based don't cares can result in extensive logic re-structuring and thus significant improvements in area, power and timing when compared to traditional combinational transformations. At the end of sequential synthesis process, the synthesis transformations needs to be validated. However, combinational equivalence checking (CEC) [Arm66, Bra93], which is the typical framework commonly used for validating logic synthesis, is appropriate only if the logic synthesis transformations are restricted to altering combinational logic.

Functional verification is often done at the RT-level and it has been observed that the number of bugs in the design is proportional to the number of lines of RTL-code [Spi04]. This implies that the efficiency of functional verification may be improved by increasing the level of abstraction in the implementation stage. The final low-level implementation can then be automatically synthesized through sequential synthesis [ABBSV00] from the abstract implementation. Increasing the level of abstraction not only improves the verification process, but also saves resources for designers since the designs can now be expressed in a more behavioral manner free from the clutter of low level design issues. However, to enable design at a higher-level abstraction, one must have the ability to verify the automatic

sequential synthesis transformation from abstract models to low-level models.

Sequential equivalence checking (SEC) is a verification framework that has been proposed to perform a true sequential check of input/output equivalence between two designs. SEC has several advantages when compared to CEC, the verification framework commonly used for validating that logic synthesis does not alter the functionality of the design. CEC does not perform any sequential analysis, it operates by correlating primary inputs and state elements across the two designs and proving that this pairing guarantees equivalence of all primary outputs and next-state functions. While powerful, CEC is for the most part limited in applicability to designs with 1:1 state element pairings. On the other hand, SEC performs sequential analysis and is not restricted to operation on designs with 1:1 state element pairings. The benefits of SEC are manifold. Due to its ability to performs a true sequential check of input/output equivalence, it can be effectively used to prove that micro-architectural optimizations needed for design closure preserve design functionality without the need for functional verification. Moreover, SEC is very well suited to validate sequential logic synthesis transformations. SEC also enables a more flexible set of applications than direct input/output equivalence checks. For example, SEC can be used to check the equivalence of specific *modes of operation* of a design, such as backward-compatibility modes of design evolutions. SEC can also be deployed against designs that are not even strictly equivalent – but can be made so by disabling initialization/test/debug logic, by ignoring output mismatches during *don't care* time-frames, by accounting for differing pipeline stages, etc.

## 1.1 SEC from designated initial states

The concept of applying same input sequence to two machines and comparing their output sequences to check their equivalence was first proposed in [Moo56]. Numerous notions of sequential equivalence have been proposed over the years. In broad terms, these approaches can be categorized into two classes: (**1**) Approaches that prove equivalence with respect to a specified initial state (e.g., [vE98]), (**2**) Approaches that attempt to demonstrate resetability across all states during the equiv-

alence proof (e.g., [Pix92, SPAB01, KH03]).

In this dissertation, we focus on sequential equivalence checking from designated initial states as formalized in Definition 2.27 in Chapter 2. In this paradigm, two designs are sequentially equivalent if and only if starting from their respective initial state sets, they produce the same output sequence for every possible input sequence. There are several compelling reasons for choosing this approach.

**(1)** Initialization data tends to be readily available in some form in most industrial design methodologies. Once the design is mature, its initialization logic is integrated, and its initialization sequence is known. Before the design is mature, it is often implemented with some initial state set in mind. Furthermore, before a design is mature, we have often observed that it has no reset logic incorporated, preventing the use of alignability analysis. It is additionally noteworthy that optimal synthesis requires initial state knowledge, since most states are actually unreachable states which may be used as don't cares during technology independent logic synthesis.

**(2)** The SEC problem given designated initial states is much more scalable as opposed to alignability style of analysis. Though techniques for alignability style of analysis have progressed substantially, e.g., through the use of SAT-based analysis [KH03], such approaches tend to be limited in applicability to designs with hundreds of state elements, or in cases, to a few thousand. In contrast, techniques for performing SEC against known initial states can readily scale up to designs with 10,000s of state elements [MBPK05]. This scalability implies dramatic reductions in manual effort due to lesser need to manually partition large designs, in turn enhancing the applicability of SEC.

**(3)** For some of the sequential design transformations against which we care to prove equivalence, traditional alignability-style analysis is inapplicable. Using scan-based initialization schemes, for example, even optimizations such as retiming may alter the necessary initialization mechanism to ensure equivalent functionality of a design. This may preclude a common initialization sequence for the designs being equivalence checked, even though independent initialization mechanisms do render the designs into a state from which they are *functionally* equivalent.

### 1.1.1 Challenges to Scalable SEC

A variety of algorithms have been proposed for scalable SEC from designated initial states [vE00, SK97, HCC+00, BC00]. The key observation is that the two designs being compared are related since one design is derived from the other through a sequence of transformations. Based on this observation, all these approaches try to exploit internal equivalences between the designs. Essentially, they all focus on identifying *sequentially redundant* gates to make verification tractable. Another common factor among these algorithms is that most of them rely on *induction* [SSS00] as the base technique for sequential redundancy identification. Though powerful, there are several challenges to the successful deployment of these techniques to verify large and complex industrial designs.

**Scalability:** SEC is computationally very expensive and suffers from the inherent state explosion problem. SEC, as opposed to CEC, does not assume 1:1 latch correspondence or 1:1 design hierarchy equivalence. This severely limits its application to smaller design units and demands a fair amount of user sophistication for successful application on larger design units.

**Applicability:** For effectiveness and industry-wide adoption SEC applications must enable equivalence proofs across a wide range of design transformations. Induction-based techniques are incomplete even when applied to prove correctness of retimed designs [RH07]. The range of transformations supported by SEC frameworks must include a variety of techniques, from retiming and resynthesis to replication of subcircuitry for reduced propagation delay to redundancy removal based on unreachability-based don't cares to outright re-encoding of parts of the design.

Additionally, when applying SEC at the design unit level, the design environment needs to be modeled correctly to avoid false fails due to illegal input scenarios. Modeling of design environments using constraints has gained widespread industrial application, and most verification languages include constructs for specifying constraints. However, little prior research has addressed the applicability of sequential redundancy identification algorithms to designs with constraints.

Motivated by the above challenges, we study the problem of sequential redundancy identification. Though the main motivation for scalable and efficient

sequential redundancy identification is improving SEC, techniques for identifying sequential redundancies have many applications. Identification and removal of sequential redundancy has been shown to be very useful in improving the performance of formal verification algorithms that are used for property checking [MBP+04]. Also, sequential redundancy identification and removal is a commonly used sequential logic synthesis optimization.

Improving the scalability and applicability of sequential redundancy identification is the focus of this dissertation. We further focus on overcoming the incomplete nature of induction-based techniques through the use of the general and modular *transformation-based verification* framework.

## 1.2 Transformation-based Verification

*Transformation-based verification* (TBV) was proposed in [KB01, Bau02] as a framework wherein one may synergistically utilize the power of various transformation algorithms to iteratively simplify and decompose complex problems until they become tractable for automated formal verification. All algorithms are encapsulated as *engines*, each interfacing via a common modular API. Each engine receives a verification problem represented as a netlist, then performs some processing on that problem. This processing could include an attempt to solve the problem (e.g., with a bounded model checking [BCCZ99] or reachability [CBM89] engine) or it could include an attempt to simplify or decompose the verification problem using a transformation (e.g. with a retiming or redundancy removal engine). In the latter case, it is generally desirable to pass the simplified problem to another engine to further process that problem. Note that the problem transmitted by an engine is generally not identical to the one received. Instead, as the problem flows from one engine to another, it is iteratively transformed into a simpler problem. As verification results are obtained on the simplified problem, those results propagate through the sequence of engines in reverse order, with each transformation engine undoing the effects of the transformations it performed to present its parent engine with results that are consistent with the netlist that its parent transmitted. A particular

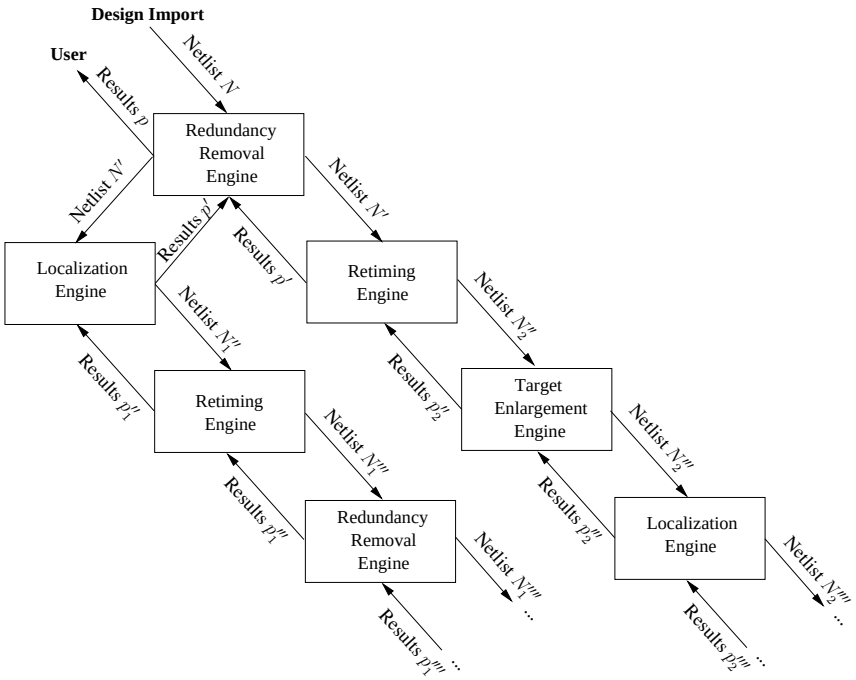



Figure 1.1: Example flow of a transformation-based verification system

instance of a TBV system is depicted in Figure 1.1.

While the complexity of a verification problem is not necessarily related to its size, the complexity class of verification algorithms indicates an exponential worst-case relationship between these metrics, which is validated by practical experience. By resource-bounding any possibly costly BDD [Bry86] or SAT [MMZ$^+$01] based analysis, it is possible to limit the complexity of most transformations used in a TBV system to polynomial while exploiting their ability to render exponential speedups to the overall verification flow as noted in [KB01, Bau02].

TBV has several features that can enable scalable sequential redundancy identification. In particular, transformations have the ability to automatically *undo* many of the commonly employed high-performance micro-architecture and design techniques such as pipelining and addition of redundant logic to minimize propagation delays (e.g., replicating a lookup queue in two places in the circuit), which otherwise make sequential redundancy identification using induction-based tech-

niques incomplete. Moreover, TBV enables the application of a variety of different complementary transformations to successively chip away at a complex problem until it can be handled by a terminal decision procedure, and thus enhances the scalability of redundancy identification.

## 1.3 Contributions

We make the following contributions in this dissertation,

- We introduce a novel and efficient sequential redundancy identification framework in [MBPK05] which significantly increases the scalability of sequential equivalence checking solutions.
- We propose the idea of using a flexible and robust set of transformation and verification algorithms for sequential redundancy identification in [MBPK05, MBP+06]. This more general approach enables greater speed and scalability and identifies a significantly greater degree of redundancy than previous approaches.
- We introduce the theory and practice of transformation-based verification in the presence of constraints in [MBA05].
- We develop the theoretical framework with corresponding efficient implementation to enable the optimal sequential redundancy identification for designs with constraints.
- We propose two new structural abstraction techniques: **(a)** structural reparameterization, and **(b)** min-cut based localization in [BM05] for yielding a netlist with minimal input count. We study the synergy that these transformations have with each other, and also with other transformations such as retiming and redundancy removal. We also propose new strategies for using these transformations to enable scalable sequential redundancy identification.

## 1.4 Organization

This dissertation is organized as follows. We first define the syntax and semantics of our netlist-based representation of the verification problem in Chapter 2. The various transformation and verification algorithms used in our experiments are described in Section 2.1. We discuss the topics of sequential redundancy identification and redundancy removal in Chapter 3. This chapter extends results of collaborative work with Jason Baumgartner, Viresh Paruthi, Robert Kanzelman and Geert Janssen in [MBPK05, MBP+06]. We introduce two new transformations to improve scalability of TBV framework and study the synergies between various transformations in Chapter 4, extending the results of collaborative work with Jason Baumgartner in [BM05]. The topic of optimal design simplification in the presence of constraints is discussed in Chapter 5. This work was done in collaboration with Adnan Aziz and Jason Baumgartner and details the extension of scalable *assume-then-prove* sequential redundancy removal frameworks [vE98, MBPK05] to optimally leverage constraints. The final topic is exploiting constraints in a TBV framework and is discussed in Chapter 6. In this Chapter, we introduce the theory and practice of transformation-based verification in the presence of constraints and extend the results of collaborative work with Jason Baumgartner and Adnan Aziz in [MBA05]. We conclude the dissertation and discuss future research directions in Chapter 7.

# Chapter 2

# Netlist:Syntax and Semantics

In this Chapter, we provide the formalisms used throughout the thesis. A reader well-versed in hardware verification may wish to skip this chapter, using it as a reference.

Our verification problem is expressed as a netlist. This netlist represents a composition of the *design under verification*, and netlist-based representations of its *environment assumptions* and *property automata* as illustrated in Figure 2.1.

As in [Bau02], we define our netlist based upon a directed graph model.

**Definition 2.1.** A *directed graph* $G = \langle V, E \rangle$ consists of a finite set of vertices $V$, and a set of directed edges between vertices $E \subseteq V \times V$. For edge $(u, v)$, we refer to $u$ as the *source* vertex and $v$ as the *sink* vertex.

**Definition 2.2.** A *directed cycle* is an ordered set of vertices $\langle v_0, \ldots, v_n \rangle$ such that $\forall i \in [0, n-1]. \big((v_i \in V) \wedge \big((v_i, v_{i+1}) \in E\big) \wedge (v_0 = v_n)\big)$.

**Definition 2.3.** We define $\mathit{inlist}(u) = \{v : (v, u) \in E\}$ as the set of vertices sourcing input edges to vertex $u$. We define the indegree of a vertex $u$ by $\mathit{indegree}(u) = |\mathit{inlist}(u)|$.

**Definition 2.4.** We define $\mathit{outlist}(u) = \{v : (u, v) \in E\}$ as the set of vertices sinking output edges from vertex $u$. We define the outdegree of vertex $u$ by $\mathit{outdegree}(u) = |\mathit{outlist}(u)|$.

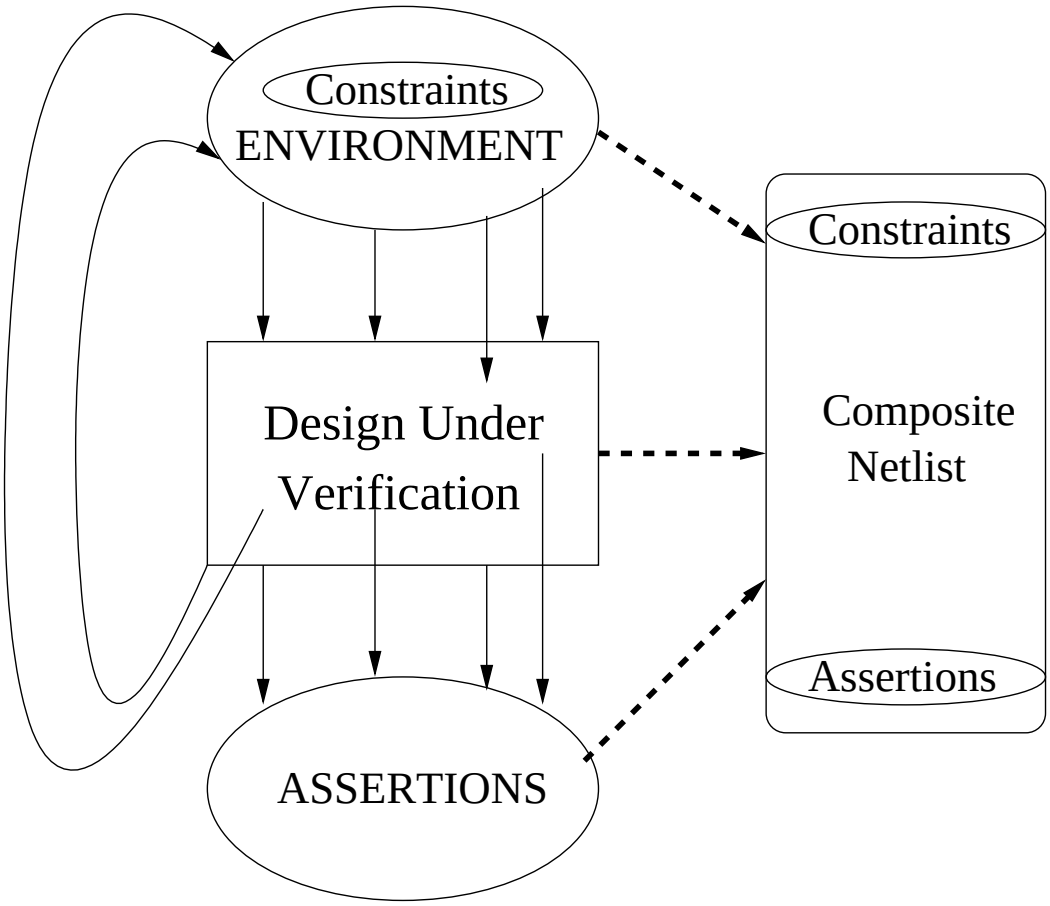



Figure 2.1: Netlist representation of Verification Problem

**Definition 2.5.** The *fanin* of vertex $u = \{u\} \bigcup_{v \in inlist(u)}$ *fanin*$(v)$. Due to the monotonicity of evaluation of this definition, and the finiteness of $G$, this set is well-formed.

**Definition 2.6.** The *fanout* of vertex $u =$ *outlist*$(u) \bigcup_{v \in outlist(u)}$ *fanout*$(v)$. This set is well-formed as per the analysis of Definition 2.5.

**Definition 2.7.** A *netlist* is a tuple $N = \langle\langle V, E\rangle, G, T, C, Z, O\rangle$ comprising a finite directed graph with vertices $V$ and edges $E \subseteq V \times V$. Function $G : V \mapsto types$ represents a mapping from vertices to a set of gate $types$ given in Definition 2.8.

Function $Z : V \mapsto V$ is the initial value mapping $Z(v)$ of each gate $v$. The set of *targets* $T \subseteq V$ corresponds intuitively to a set of properties to be checked, as will be discussed in Definition 2.12; the set $C \subseteq V$ represents the *constraints*, the significance of which will be described in Definition 2.11. The set $O \subseteq V$ represents the set of *primary outputs*.

**Definition 2.8.** The set $types$ (which is the range of function $G$) = { PRIMARY INPUT, ZERO, INVERTER, AND, REGISTER}.

Hereafter we denote the set of vertices of type REGISTER as $R$, the set of vertices of type PRIMARY INPUT as $I$, and the vertices of type INVERTER sourced by vertices of type ZERO as constant ONE.

**Definition 2.9.** Well-formed netlists are required to satisfy the following constraints:

1. The indegree of each vertex is consistent with its specified type. Vertices of type ZERO or type PRIMARY INPUT have indegree of 0, vertices of type INVERTER or type REGISTER have indegree of 1 and vertices of type AND have indegree of 2.

2. The netlist does not contain any combinational cycles: directed cycles in $\langle V, E\rangle$ comprising no vertices of type REGISTER.

3. The initial value mapping of gates of type REGISTER must be entirely combinational, i.e. *fanin cone*$\big(Z(R)\big) \cap R = \emptyset$.

**Definition 2.10.** The semantics of netlist $N$ are defined in terms of *traces*: $0, 1$ valuations to gates over time. We denote the set of all legal traces associated with a netlist by $P \subseteq [V \times \mathbb{N} \mapsto \{0, 1\}]$, defining $P$ as the subset of all possible functions from $V \times \mathbb{N}$ to $\{0, 1\}$ which are consistent with the following rule. The value of

gate $v$ at time $i$ in trace $p$ is denoted by $p(v, i)$.

$$p(v,i) = \begin{cases} 0 & : v \text{ is ZERO} \\ \neg p(u_1) & : v \text{ is an INVERTER} \\ s^i_{v_p} & : v \text{ is a PRIMARY INPUT with sampled value } s^i_{v_p} \\ \big(p(u_1,i) \wedge p(u_2,i)\big) & : v \text{ is a 2-input AND gate} \\ p(u_1, i-1) & : v \text{ is a REGISTER and } i > 0 \\ p\big(Z(v), 0\big) & : v \text{ is a REGISTER and } i = 0 \end{cases}$$

Term $u_j$ denotes the source vertex of the $j$-th incoming edge to $v$, implying that $(u_j, v) \in E$.

Netlists whose semantics are defined according to Definition 2.10 and which consists of gate types defined in Definition 2.8 are sufficient to succinctly model sequential digital hardware. Note that any sequential hardware design can be mapped onto a netlist representation containing only constants, PRIMARY INPUTS, two-input AND gates, inverters, and REGISTERS, using straight-forward logic synthesis techniques.

**Definition 2.11.** The length of a trace *p* is denoted as $length(p)$, and defined as $\min\{\text{i} : \exists c \in C.\ p(c,i) = 0\}$. Throughout this thesis, we use the convention that $\max\{\emptyset\} = 0$ and $\min\{\emptyset\} = \infty$. Thus if $|C| = 0$ or $\forall i \in \mathbb{N}.\ \forall c \in C.\ p(c,i) = 1$, then $length(p) = \infty$.

A trace is hereafter understood to be restricted to its valid prefix wherein all constraint gates evaluate to 1. Practically, it is understood that a trace is to be reasoned about with respect to finite prefix of interest, e.g., until the assertion of a target.

**Definition 2.12.** A target $t$ is said to be *hit* in a trace $t$ at time $i$ iff $\big(p(t,i) = 1\big) \wedge \big(i < length(p)\big)$. A target *t* is *not hittable* iff $\forall p \in P.\ \big(\forall i < length(p)\big).\ p(t,i) = 0$. We say that a target is *reachable* if there is a trace that hits the target, and one which is not hittable is *unreachable*.

**Verification Goal:** The verification goal associated with a netlist is to obtain a trace illustrating an assertion of a target within its valid prefix (such a trace is hereafter referred to as a *counterexample*), or to prove that no such trace exists.

**Definition 2.13.** The *cone of influence* of vertex $u$ is denoted as $\mathit{coi}(u)$, and defined as $\mathrm{fanin}(u) \bigcup_{v \in Z(R \cap fanin(u))} \mathit{fanin}(v)$.

**Definition 2.14.** The *combinational fanin* of vertex $u$ is denoted as $\mathit{cfi}(u)$, and defined as $u$ if $u \in R$, else $u \bigcup_{v \in inlist(u)} \mathit{cfi}(v)$ if $u \notin R$. This set is well-formed as per the analysis of Definition 2.5.

**Definition 2.15.** The *combinational fanout* of vertex $u$ is denoted as $\mathit{cfo}(u)$, and defined as $\mathit{outlist}(u) \bigcup_{v \in \{outlist(u) \setminus R\}} \mathit{cfo}(v)$. This set is well-formed as per the analysis of Definition 2.5.

**Definition 2.16.** A *state* is a valuation to the REGISTERs of a netlist. A *reachable state* is one which may occur in a trace, and an *initial state* is a reachable state which may occur at time 0. A *dead-end state* is an unreachable state for which no valuation to the PRIMARY INPUTs will satisfy the constraints.

**Definition 2.17.** A *cut of a netlist* is a partition of $V$ into two sets: $\mathcal{C}$ and $\overline{\mathcal{C}} = V \setminus \mathcal{C}$. A cut induces two sets of *cut gates* $V_{\mathcal{C}} = \{u \in \mathcal{C} : \exists v \in \overline{\mathcal{C}}. \big(((u,v) \in E) \vee (v \in R \wedge u = Z(v))\big)\}$, and $V_{\overline{\mathcal{C}}} = \{u \in \overline{\mathcal{C}} : \exists v \in \mathcal{C}. \big(((u,v) \in E) \vee (v \in R \wedge u = Z(v))\big)\}$.

One may visualize a cut of netlist $N$ as the composition [GL94] of netlists $N_{\mathcal{C}} \parallel N_{\overline{\mathcal{C}}}$, with $V_{\mathcal{C}}$ denoting inputs to $N_{\overline{\mathcal{C}}}$ which are closed under the composition, and with $V_{\overline{\mathcal{C}}}$ denoting inputs to $N_{\mathcal{C}}$ which are closed under the composition.

**Definition 2.18.** An *s-t cut* is a cut seeded with vertex sets $s \subseteq \mathcal{C}$ and $t \subseteq \overline{\mathcal{C}}$. An *s-t min-cut* refers to an *s-t cut* where $V_{\mathcal{C}}$ is of minimal cardinality.

Algorithmically, when computing an *s-t min-cut*, sets $s$ and $t$ will be selected according to some application-specific criteria, and provided as constraints to the min-cut solver. Numerous algorithms have been proposed for the efficient computation of *s-t min-cut*, in this thesis, we use the *augmenting path* algorithm [FF56].

**Definition 2.19.** Gate sets $A \subseteq V$ and $A' \subseteq V'$ of netlists $N$ and $N'$, respectively, are said to be *trace equivalent* iff there exists a bijective mapping $\psi : A \mapsto A'$ such that:

- $\forall p \in P. \exists p' \in P'.\ \big(\forall i < length(p)\big).\ \forall a \in A.\ p(a,i) = p'\big(\psi(a), i\big)$
- $\forall p' \in P'. \exists p \in P.\ \big(\forall i < length(p')\big). \forall a \in A.\ p(a,i) = p'\big(\psi(a), i\big)$

**Definition 2.20.** Netlists $N$ and $N'$ are said to be *trace equivalent* with respect to target sets $T$ and $T'$ respectively, iff there exists a bijective mapping $\psi : T \mapsto T'$ such that:

$\forall p \in P. \exists p' \in P'. \forall t \in T.\ \big(\forall i < length(p)\big).\ p(t,i) = p'\big(\psi(t), i\big)$

$\forall p' \in P'. \exists p \in P. \forall t \in T.\ \big(\forall i < length(p')\big).\ p(t,i) = p'\big(\psi(t), i\big)$

**Definition 2.21.** A netlist transformation $\mathbb{T} : \mathcal{N} \mapsto \mathcal{N}$ is a function from set of all well-formed netlists $\mathcal{N}$ (as per Definition 2.9) to set of all well-formed netlists. The netlist transformation $\mathbb{T}$ preserves property checking (said to be "*property-preserving*") iff $\forall N \in \mathcal{N}$, $N$ and $\mathbb{T}(N)$ are trace-equivalent with respect to target sets $T_N$ and $T_{\mathbb{T}(N)}$ respectively.

**Definition 2.22.** Given two netlists $N$ and $N'$, we say that we say that $N'$ *trace-contains* $N$ with respect to target sets $T$ and $T'$ respectively, iff there exists a bijective mapping $\psi : T \mapsto T'$ such that:

$\forall p \in P. \exists p' \in P'. \forall t \in T.\ \big(\forall i < length(p)\big).\ p(t,i) = p'\big(\psi(t), i\big)$

**Lemma 2.1.** If $N$ is trace-equivalent or property-preserving trace-equivalent to $N'$, verifying $N'$ in place of $N$ is *sound* and *complete*: a proof that a target cannot be asserted, or the generation of a counterexample, obtained on $N'$ may be reused as a result for the corresponding target in $N$. If $N$ is trace-contained by $N'$, verifying $N'$ in place of $N$ is *sound* but *incomplete*: a proof that a target cannot be asserted which is obtained on $N'$ may be reused as a correct result for the corresponding target of $N$, though a counterexample on $N'$ may not be valid for $N$.

**Definition 2.23.** Vertices $v$ and $v'$ of $N$ are said to be *equivalent* iff $\forall p \in P.\ \big(\forall i < length(p)\big).\ p(v,i) = p(v',i)$.

**Definition 2.24.** A *merge* from gate $g_1$ onto gate $g_2$ consists of replacing every fanout edge $(g_1, g_3) \in E$ with $(g_2, g_3)$. To facilitate subsequent reasoning (e.g., trace analysis), gate $g_1$ is next made a *buffer* – an AND gate with both its inputs tied together – which entails removing its fanin edges, and adding edge $(g_2, g_1)$. Gate $g_1$ is referred to as the *merged from* gate and $g_2$ is referred to as the *merged onto* gate.

Without loss of generality we assume that $g_1 \not\equiv g_2$, and that merges will yield valid netlists – i.e., be initiated only if no combinational cycles will result.

**Definition 2.25.** Given a netlist $N$, gates $g_1, g_2 \in V_N$, a *miter* $m_{g_1,g_2}$ over gates $g_1$ and $g_2$ is a gate representing the function $g_1 \not\equiv g_2$.

**Definition 2.26.** Consider two disjoint netlists $N_1 = \langle\langle V_1, E_1\rangle, G, T_1, C_1, Z_1, O_1\rangle$, $N_2 = \langle\langle V_2, E_2\rangle, G, T_2, C_2, Z_2, O_2\rangle$, and bijective mappings $\psi : I_1 \mapsto I_2$, $\phi : O_1 \mapsto O_2$, where $I_1$ and $I_2$ are the sets of PRIMARY INPUT vertices in $N_1$ and $N_2$ respectively. The product netlist $N_{1\times2}\langle\langle V_{1\times2}, E_{1\times2}\rangle, G, T_{1\times2}, C_{1\times2}, Z_{1\times2}, O_{1\times2}\rangle$ is constructed as follows,

- $V_{1\times2} = V_1 \cup V_2$
- $E_{1\times2} = E_1 \cup E_2$
- $Z_{1\times2}(v) = Z_1(v)\ if\ v \in V_1,\ else\ Z_2(v)$
- $C_{1\times2} = C_1 \cup C_2$
- $O_{1\times2} = \emptyset$
- $\forall i \in I_1.\ merge\big(i, \psi(i)\big)$
- $T_{1\times2} = \{m_{o,\phi(o)} : o \in O_1\}$

Conceptually, the product netlist is formed as the union of $N_1$ and $N_2$, merging corresponding PRIMARY INPUTS to ensure than an identical sequence of inputs are applied to both $N_1$ and $N_2$, and adding miters over corresponding primary outputs of $N_1$ and $N_2$ and labeling them as targets.

**Definition 2.27.** Given two netlists $N_1$ and $N_2$, and the corresponding product machine $N_{1\times2}$, netlist $N_1$ is sequentially equivalent to $N_2$ iff $\forall p \in P.\, \forall t \in T_{1\times2}.\, \big(\forall i < length(p)\big).\, p(t,i) = 0$. Conceptually, starting from the specified initial states, $N_1$ is sequentially equivalent to $N_2$ if $N_1$ and $N_2$ exhibit identical sequences of valuations to their primary outputs under all possible sequences of valuations to their primary inputs.

## 2.1 Transformation & Verification Algorithms

In this section, we describe the various transformation and verification engines used in experimental results throughout the thesis. These engines are components of a transformation-based verification framework.

- **COM**: a redundancy removal engine which uses combinational techniques such as structural hashing and resource-bounded BDD- and SAT-based analysis (which are NP-complete sub-problems) to identify gates which are functionally redundant across all states [KPKG02], as well as a variety of rewriting techniques [MCB06] to reduce netlist size.

- **RET**: a min-area retiming engine, which reduces the number of REGISTERs by shifting them across combinational gates [KB01].

- **CUT**: a reparameterization engine, which replaces the fanin-side of a *cut* of the netlist graph with a trace-equivalent, yet simpler, piece of logic [BM05].

- **LOC**: a localization engine, which isolates a cut of the netlist local to the targets by replacing gates by primary inputs. **LOC** is an overapproximate transformation, and uses a SAT-based refinement scheme to prevent spurious counterexamples [BM05].

- **ISO**: a structural isomorphism detection engine, which equivalence-classes isomorphic targets, such that only one representative target per equivalence class needs to be solved by a child engine flow [MHB98].

- **MOD**: a structural state-folding engine used to abstract certain clocking and latching schemes, generalizing the techniques presented in [BK05, BTA+00].
- **SAT**: a hybrid-algorithm SAT solver based upon [KPKG02], which interleaves redundancy removal and structural rewriting with BDD- and SAT-based analysis.
- **RCH**: a BDD-based reachability engine [CBM89]
- **IND**: a SAT-based induction [SSS00] engine which uses unique-state constraints.
- **BIG**: a structural target-enlargement engine, which replaces a target by the simplified characteristic function of the set of states which may hit that target within $k$ time-steps [BKA02].
- **SCH**: a semi-formal search engine, which interleaves random simulation (to identify *deep*, interesting states) and symbolic simulation (using either BDDs [PJW05] or the **SAT** engine) to branch out from states explored during random simulation.
- **EQV**: a sequential redundancy removal engine based upon the algorithm described in Section 3 [MBPK05].
- **ITP**: a SAT-based reachability engine, which performs unbounded reachability analysis using interpolation [McM03].

# Chapter 3

# Sequential Redundancy Identification

## 3.1 Overview

In this chapter, we focus on sequential redundancy identification. We are motivated by the fact that scalable and efficient sequential redundancy identification algorithms are at the core of a scalable sequential equivalence checking framework. Moreover, it has been shown that sequential redundancy removal has a huge impact on the performance of a variety of formal verification algorithms [MBP+04]. Once a pair of redundant gates are identified, the netlist may be simplified by replacing each reference to one gate by a reference to the other; i.e., by merging one of the gates onto the other as per Definition 2.24.

Our general-purpose sequential redundancy identification framework is based on the *assume-then-prove* [vE98, HCC+00, NPMJ03] paradigm. We generalize the domain specific use of assumptions under the framework of *speculative reduction* which results in a significant improvement to the scalability of sequential redundancy identification. For redundancy identification, we propose the use of a larger and more robust variety of synergistic transformation and verification algorithms. This is in contrast to prior work which relied upon a smaller set of algorithms, such

as induction[vE98, BC00, HCC+00] or approximate-reachability-based [HCC+00] fixed-point computation. Experiments confirm that this more general approach enables greater speed and scalability, and identifies a significantly greater degree of redundancy, than previous approaches. We also discuss a more general paradigm that allows us to exploit *even redundancy that holds only for an initial bounded time-frame, but not across all time-frames*. This paradigm enables faster and deeper exhaustive search and has benefits to both redundancy identification and falsification of properties.

The rest of this chapter is organized as follows. In Section 3.2 we review sequential redundancy identification framework and illustrate the technique of speculative reduction. In Section 3.2.1, we prove the correctness of the use of the speculatively-reduced model to identify redundancy. We address the use of a variety of transformation and verification algorithms to identify redundant gates in Section 3.3. In Section 3.4, we propose novel techniques to improve scalability through intelligent refining of redundancy candidates. We propose the creation of a transformation engine based on sequential redundancy removal in Section 3.5. In Section 3.6, we discuss the use of the speculatively-reduced model for falsification. We provide experimental results to illustrate the power of these techniques in Section 3.7.

## 3.2 Redundancy Identification Framework

Given a netlist, redundancy identification frameworks based on the assume-then-prove paradigm such as those of [vE98, HCC+00] operate as per the algorithm of Figure 3.1 .This paradigm often begins with a redundant gate guessing approach as in Step 1, using a variety of techniques which includes both *functional* and *non-functional* approaches. Non-functional approaches approximate the set of truly redundant gates using name- and structure-based comparisons [CP], hence are useful in equivalence checking applications since often only a fraction of the circuit is redesigned. However, they are often not applicable to identify redundancy to enhance property checking, and tend to break down within substantially redesigned

1. Use an arbitrary set of algorithms to compute the *redundancy candidates* of $N$: Sets of equivalence classes of gates, where every gate $g$ in equivalence class $Q(g)$ is suspected to be equivalent to every other gate in the same equivalence class, along every trace. Validate that the redundancy candidates are accurate in the initial states.

2. Select a *representative gate* $R\big(Q(g)\big)$ from each equivalence class $Q(g)$.

3. Construct the *speculatively-reduced netlist* $N'$ from $N$ by replacing the source gate $g$ of every edge $(g, h) \in E$ by $R\big(Q(g)\big)$. Additionally, for each gate $g$, add a miter $T_g$, which is a target representing function $g \not\equiv R\big(Q(g)\big)$.

4. Attempt to prove that each of the miters in $N'$ is semantically equivalent to 0.

5. If any miters cannot be proven equivalent to 0 due to obtaining a trace illustrating their assertion or due to an inconclusive result from the proof algorithm, refine the equivalence classes to separate the corresponding gates and goto Step 2; else proceed to Step 6.

6. All miters have been proven equivalent to 0; return the accurate equivalence classes.

Figure 3.1: Sequential redundancy identification framework

sub-circuits. Functional approaches [CP, ADMS02] overapproximate the set of redundant gates by analyzing traces produced by random simulation, approximate symbolic search, semi-formal analysis, etc.. Note that semantic analysis is generally necessary for optimality; e.g., in case sequential redundancy was added to one design, $N$:$M$ (vs. 1:1) REGISTER grouping may be needed.

Once the redundancy candidate gates are guessed, the next step is to create the model to be equivalence checked which incorporates redundancy candidate assumptions and the checks to validate the assumptions (the "assume" step). We introduce a novel technique called speculative reduction to perform the "assume" step. A representative gate is selected for each equivalence class and every candidate gate in the equivalence class other than the representative is merged onto the representative gate. Note that the selection of representatives must be performed such that the speculatively-reduced netlist $N'$ is well-formed as per Definition 2.9; this is often accomplished by selecting the representative of each equivalence class as an arbitrary topologically-shallowest gate with respect to combinational fanin

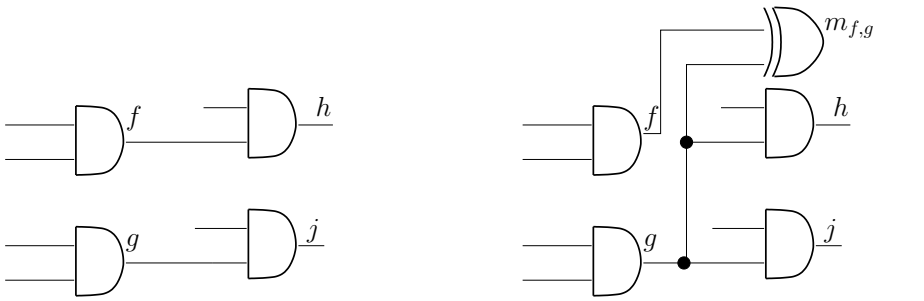



Figure 3.2: Illustration of speculative reduction

DAGs. Next, to prove the correctness of the assumption, we add a miter checking the exclusive-OR of the candidate gate and representative gate. An illustration of speculative reduction is depicted in Figure 3.2. Finally, proof analysis is performed on the speculatively-reduced model to attempt to validate the correctness of the redundancy candidates (the "prove" step). This correctness is represented by the unreachability of the miters added to validate the speculative merges.

Speculative reduction is very powerful and instrumental in enhancing the scalability of sequential redundancy removal framework. Speculative reduction offers several advantages.

- The speculative merging facilitates logic reduction in the fanout of the redundancy candidates. Propagating these changes downstream often results in a huge reduction in the size of the speculatively-reduced netlist and the number of distinct miters to be solved. This greatly enhances proof algorithms such as induction resulting in big speedups. Experiments confirm that the gain in runtime due to speculative reduction can be several orders of magnitude on large designs [MCBJ08].

- It plays a similar role to cut-pointing in CEC frameworks and often helps reduce the complexity of identifying sequential redundancy from PSPACE-complete to NP-complete.

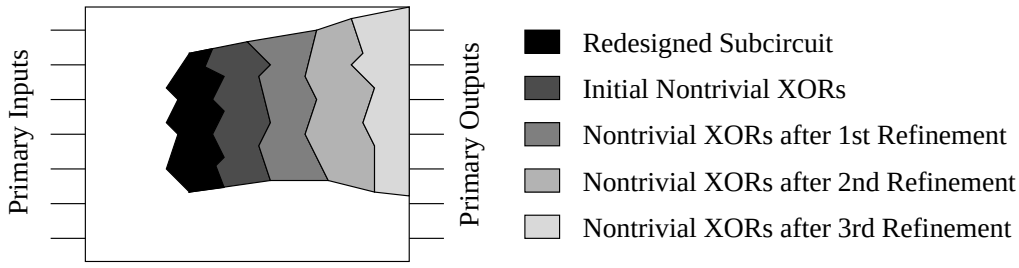



Figure 3.3: Failed proofs increase post-refinement complexity

- It enables the proof algorithms to focus on harder miters in the redesigned portions of the netlist by trivializing the miters in the non-redesigned portions of the netlist.
- It is the key to enable non-inductive proof algorithms to be used to discharge the miters as described in Section 3.3.

Failed proofs, whether falsified or inconclusive (e.g., due to an ineffective algorithm or insufficient resources), cause a refinement of the candidates and another assume-then-prove iteration. There are two potential causes of failure to prove a miter as unreachable in Step 4. First, some of the candidates may be incorrect, i.e., not truly equivalent in all reachable states. This is reflected by the generation of a trace asserting the corresponding miters. Second, resource limitations (or an incomplete proof technique) may preclude the solution of a subset of the miters. For example, induction may become computationally expensive after a particular depth, hence miters that are not provable within that threshold will remain unsolved. Discarding miters due to resource limitations not only immediately prunes the equivalence classes, but also tends to result in a significant amount of future resource-gated refinements. This is because, by diminishing the amount of speculative merging, each refinement effectively weakens the redundancy *induction hypothesis* – requiring greater resources to complete proofs across refinements.

To illustrate this problem, assume that we wish to perform sequential equivalence checking over primary outputs for two versions of the netlist depicted in Figure 3.3. Only the darkest-shaded portion of the circuit was redesigned, hence

each gate outside of this region will have a unique correspondent in the modified design. Only those miters over gates which have a source edge from the redesigned region will initially be nontrivial; the others will be trivially of the form $g \not\equiv g$ due to the speculative merging. If any of these cannot be solved due to resource limitations, the subsequent refinement will cause a set of gates in the immediate fanout of the refined gates to become nontrivial. Since their fanin gates were too difficult to prove equivalent, these gates also are likely to be too difficult to prove. This refinement ripples forward, increasing the resources needed for the subsequent proofs, pushing more miters from provable to unsolvable. Ultimately, many gates in the shaded region may remain unsolved, including some of the primary outputs which are the goal of the equivalence check.

### 3.2.1 Correctness of the Speculatively-Reduced Model

In this section we discuss the correctness of the use of speculatively-reduced model, constructed as per step 3 of the algorithm of Figure 3.1.[1], for redundancy identification. Theorem 3.1 provides the theoretical justification of how we may utilize the speculatively-reduced model both for proofs and falsification, as will be discussed in Section 3.6.

**Theorem 3.1.** Assume that we apply the same sequence of test vectors to the corresponding PRIMARY INPUTS of the original and the speculatively-reduced model. This simulation process will expose its first mismatch within the equivalence classes in the original design at time $i$ if and only if it first asserts one or more miters in the speculatively-reduced model at time $i$.

*Proof.* We prove this theorem by induction on the length of the trace produced by the simulation run.

*Base Case:* In our framework, all gates are equivalent to their representatives in the initial states. This implies that at time 0, all redundancy candidates are

[1] If both $g$ and $R\big(Q(g)\big)$ are REGISTERS, we instead create the miter over their next-state functions to reduce the size of the speculatively-reduced model, similarly to [HCC+00].Our technique also includes *antivalent* gates – which evaluate to opposite values in all reachable states – in the equivalence classes, similarly to [vE98].

correct – hence the speculative merging does not alter any gate's valuation in the trace at time 0, and no miter assertions may occur at time 0.

*Inductive Step:* By the induction hypothesis, assume that no mismatches nor miter assertions occurred at times $0, \ldots, j$. We prove that the theorem holds at time $j+1$. If no mismatches are exposed, and no miters are asserted, this trend trivially holds. Otherwise, first consider the case that at time $j+1$, there is a nonempty set of gates $A$ which each differ from their representatives in the trace over the original design. Consider the subset $B \subseteq A$ such that for every $b \in B$, neither $b$ nor $R\big(Q(b)\big)$ contain any other elements of $A$ in their combinational fanin cones – the set of gates which may be reached fanin-wise without traversing through a REGISTER. Clearly $B$ is non-empty: gate $b$ cannot directly be speculatively merged, nor transitively through a sequence of speculative merges, onto a representative which lies in its combinational fanout without introducing a combinational cycle. Note that any speculative merging in the fanin cone of $b \in B$ and $R\big(Q(b)\big)$ cannot alter their valuation at time $j+1$, since that merging was correct for all such gates (other than those in B themselves) at times $0, \ldots, j+1$. Therefore, if the simulation exposes a mismatch on set $B$ in the original design at time $j+1$, the miters $T_B$ must be asserted in the speculatively-reduced model at time $j+1$. By the same argument, if miters $T_A$ are asserted in the speculatively-reduced model at time $j+1$, we may similarly compute a nonempty subset $B \subseteq A$ for which the simulation must expose a mismatch in the original design at time $j+1$. □

## 3.3 Proving Redundancy Candidates

In this section, we discuss algorithms used in the "prove" step of assume-then-prove paradigm. What is particularly novel in our framework vs. prior approaches is the much richer set of algorithms that we use in the "prove" step. We propose a more general scheme which leverages arbitrary synergistic transformation-based verification (TBV) [KB01] flows to prove redundancy candidates.

Discarding redundancy candidates during refinement effectively weakens the induction hypothesis of the assume-then-prove framework. Discarding candidates due to ineffective proof techniques thus precludes the leveraging of those

internal equivalences to simplify the proof of other miters after the refinement, which often ultimately avalanches into a failed proof of output equivalence as depicted in Figure 3.3. Simply stated, even within a single design component, every miter is essentially a distinct verification problem. It is well-known that using the proper set of algorithms for a given verification problem can be exponentially faster than using an inferior set. Prior work has relied primarily upon induction for the proof step [vE98, BC00], possibly augmented with localized reachability analysis for redesigned regions which cannot be well-paired [HCC$^+$00]. While such algorithms are indeed key in our framework, we additionally may leverage a variety of synergistic transformation and verification engines to attempt to discharge the miters. Our TBV framework can thus be seen as a robust and flexible *extension* of induction-based frameworks. Using a TBV framework to solve redundancy candidates has several benefits.

First, TBV often substantially reduces the size of the speculatively-reduced model, and the number of distinct miters therein. This tends to significantly reduce resource requirements, regardless of the proof techniques used to solve the miters.

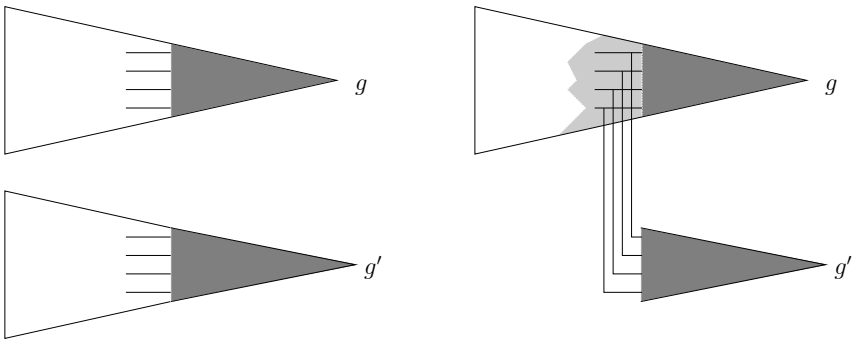
Second, the transformations themselves are sufficient to solve many of the miters. For example, given two sub-circuits differing only by retiming and resynthesis, polynomial-resource retiming and redundancy removal engines [MBP$^+$04] alone are often able to trivialize the resulting equivalence checking problem without a need for more costly inductive analysis. Several techniques have been proposed to simplify the verification of retiming- and resynthesis-optimized designs. In [MS03], a technique is proposed for extracting a *retiming invariant* to be inductively discharged more simply than could direct equivalence candidates. A SAT-based inductive technique [RH07] has been proposed to enable scalable equivalence checking if the design transformations are restricted to retiming+resynthesis+retiming. An approach orthogonal to above approaches attempts to maintain synthesis history [BM07]. Keeping track of synthesis history enables generation of inductive invariants to simplify redundancy identification. However, we have found our approach tends to be more robust, and a practical superset to the prior research in several ways. The generation of inductive invariants [MS03] requires a prepro-

cessing step on a rigidly-transformed intermediate version of the design, e.g., one derived only through retiming with limited resynthesis. The approach in [RH07] is only applicable to limited retiming and resynthesis transformations. In contrast, many design evolutions intertwine such transformations [BK01] along with a variety of other (often manually-performed) transformations such as state-machine re-encoding. The approach of using synthesis history [BM07] is promising, however, it is inapplicable when synthesis history is not available. This is typically the norm since most of the transformations are done manually and the sequential synthesis tools do not yet have the ability to maintain the synthesis history. Overall, our TBV approach is able to efficiently discharge these simpler retiming and resynthesis subproblems without dedicated preprocessing, and also more generally scales to efficiently compensate for more aggressive design modifications.

The third, and possibly greatest, benefit of the use of TBV for proving redundancy candidates comes through the ability to leverage independent algorithm flows on each individual miter. Different transformation and verification algorithms are better-suited for different problems – often exponentially so [MBP+04]. Even the most complex miter may often be rendered sufficiently small to be reliably solved using a variety of proof techniques, instead of relying solely upon possibly inconclusive induction or approximate analysis.

**Localization for Identifying Redundancy**

One particularly pronounced benefit we have noted lies in the use of localization in the per-miter transformation flows. This is particularly useful when a redesigned portion of the design relies upon satisfiability don't-cares (SDCs) from its fanin cone to ensure equivalence, and when the sequential depth of logic in that cone precludes inductivity. By using a localization-refinement scheme to discharge the miters in the speculatively-reduced model, only the subset of the cone of influence necessary to ensure those SDCs will be included for proof analysis. Figure 3.4 illustrates an example problem, where only a portion of the two cones has actually been redesigned. Each gate in the non-shaded region of the top design of Figure 3.4a has a unique correspondent in the bottom design, resulting in trivial miters. Gate $g$ has




(a) Original designs. Redesigned subcircuit darkly shaded.

(b) Speculatively-reduced model. Localized circuit needed to prove $g \equiv g'$ lightly shaded.

Figure 3.4: Use of localization to prune speculatively-reduced model

$g'$ as a correspondent, resulting in a nontrivial miter over the two darkly shaded regions plus the original cone driving one of those regions as depicted in Figure 3.4b. In cases, the redesign may not rely upon any SDC conditions in its fanin cone to ensure this equivalence, hence localization may require no gates outside of the darkly shaded region. Otherwise, localization may further reduce the amount of logic from that cone needed to discharge the miter as depicted by the lightly shaded region in Figure 3.4b. This localization often prunes the resulting subproblem to tens or (few) hundreds of REGISTERs, regardless of the size of the corresponding cone. The smaller subproblem is much more easier for a verification algorithm such as reachability analysis or SAT-based interpolation to solve as compared to original design.

TBV-based localization may be viewed as a generalization of using design hierarchy boundaries [AMP04] or speculative merge points [HCC+00] to insert cutpoints. TBV offers greater reduction potential through localization since it is not limited to specific boundaries, and offers more robust refinement algorithms (e.g., [Wan03]) in case the chosen cutpoints result in spurious mismatches. TBV additionally enables a robust variety of synergistic transformations before and after localization [MBP+04], unlike prior approaches.

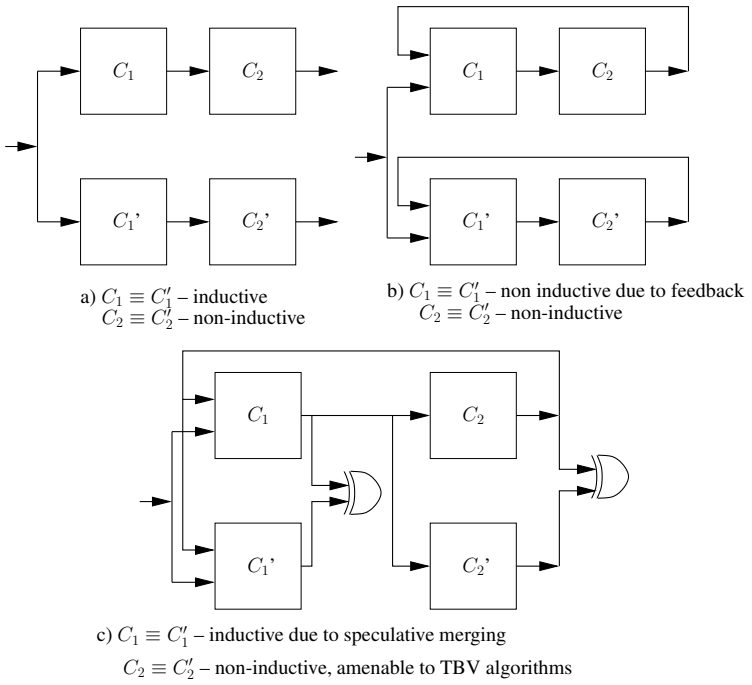



Figure 3.5: Advantages of applying TBV on spec-reduced design

### TBV on the Original Design vs. Speculatively-Reduced Design

An interesting question is whether we need to run the transformation and verification algorithms on the speculatively-reduced model. Would it not be equally powerful to either apply transformations on the original design followed by induction-based redundancy identification on the transformed design or perform induction-based redundancy identification followed by the application of transformation and verification algorithms on the simplified model? It is noteworthy that the benefits of our technique to enhance redundancy identification cannot be fully realized without the ability to process the speculatively-reduced model using TBV flows.

To illustrate the advantages of using speculative-reduction, consider the two designs being equivalence checked in Figure 3.5(a). There are two components $C_1$ and $C_2$. The changes made in $C_1$ are such that induction-based techniques can prove $C_1 \equiv C_1$'. However induction-based techniques cannot prove $C_2 \equiv C_2$'. For the design in Figure 3.5(a), we can apply induction-based techniques to first

merge the outputs of $C_1$ and $C_2$' and have TBV algorithms tackle $C_2 \equiv C_2$'. Now consider the design in Figure 3.5(b), where there is feedback from $C_2$ to $C_1$. In this particular case, the equivalence check between $C_1$ and $C_1$' is no longer inductive. The inability to inductively prove the equivalence between $C_1$ and $C_1$' will make it doubly difficult for TBV algorithms since it now needs to prove both $C_1 \equiv C_1$' and $C_2 \equiv C_2$'. However, if we use speculative reduction as illustrated in Figure 3.5(c), the feedback from $C_2$ is no longer a bottleneck for induction to prove equivalence between $C_1$ and $C_1$'.

Moreover, transformation and verification algorithms are more effective *after* the speculative merging. For example, in Figure 3.4, localization will not be able to isolate the smaller redesigned portion for analysis without the speculative merging. Verification algorithms also face bottlenecks without speculative reduction. Interpolation [McM03] is a verification algorithm that we have found to be very useful to solve non-inductive miters. Experiments demonstrate that in the absence of speculative reduction, interpolation rarely converges and has runtimes and memory consumption that are several orders of magnitude higher when compared to application of interpolation in speculatively-reduced model. Speculative merging also enables a pronounced synergistic increase in the reduction potential of other transformations such as combinational rewriting [MCB06]. Rewriting techniques can be used to find redundancies across both the designs being equivalence checked if it is applied on the spec-reduced model. Rewriting techniques would be unable to find such redundancies in the absence of speculative reduction.

Furthermore, certain methodologies may limit the type of transformations which may be applied prior to the sequential redundancy identification. For example, equivalence checking frameworks may use name- and structure-based comparisons to guess candidates [CP]. Prior transformations such as retiming with intertwined resynthesis [BK01] may render such comparisons ineffective. As illustrated in Figure 3.3, on the most complex problems it is imperative to leverage as robust a variety of transformation and verification algorithms as possible to ensure that the truly correct candidates may be proven equivalent; otherwise, an avalanche of resource-gated refinements may occur, resulting in suboptimal merg-

ing. On such complex problems, we have found that either applying TBV only to the sub-optimally-merged netlist (e.g., derived using induction alone to discharge the miters) or applying TBV on the original model followed by induction-based redundancy identification cannot compensate for the strength of applying TBV to solve all miters of the speculatively-reduced model.

## 3.4 Refining the Redundancy Candidates

If a miter cannot be proven unreachable, its equivalence class must be refined to separate the corresponding candidate from its representative. As discussed in Section 3.2, there are two causes of such failed proofs: a miter is asserted (i.e., a trace is generated which differentiates the corresponding candidates), or a miter cannot be solved within the available resource limits.

As per Theorem 3.1, a trace asserting any of the miters in the speculatively-reduced model must differentiate at least one pair of redundancy candidates in the original design. Note, however, that the speculative merging may cause certain gates in the fanout of incorrectly-merged gates to either mismatch when they should not, or to not mismatch when they should. An effective way to determine precisely which candidates have been differentiated by the corresponding miter-asserting trace is to simulate the original design with the input sequence illustrated in that trace. By additionally injecting random stimulus to any don't-cares therein and extending the sequential length of that trace, we may obtain a useful set of patterns from which we may refine all candidates. [2] However, if the processing of a trace is not possible, the proof of Theorem 3.1 nonetheless indicates a minimal subset of gates $B \subseteq A$ which must be refined given only the knowledge of the set $A$ of gates whose miters have been asserted.

For miters which cannot be proven due to resource limitations, we may again utilize the proof of Theorem 3.1 to choose a minimal subset of candidates to refine

[2] As noted in [Kue04], we have found that a trace showing how to differentiate one pair of redundancy candidates often exposes an interesting corner-case behavior of the design which is also of utility to further refine other candidates.

by taking $A$ to be the set of miters which could not be proven unreachable. However, this subsetting tends to be of lesser practical utility since the gates in the fanout of this refined subset will tend to become even more difficult to prove after the refinement as illustrated in Figure 3.3, unless the subset $B$ includes truly incorrect candidates whose incorrectness was the cause of the difficulty of proving $A \setminus B$.

### 3.4.1 Refinement Iterations

Based on the discussion above, poor equivalence classing due to incorrect candidate guessing or a very weak proof algorithm could result in a large number of refinement loops in the algorithm of Figure 3.1. Excessive number of refinements could be fatal to the scalability and conclusiveness of assume-then-prove paradigm, especially when applied on netlists with millions of AND gates and hundreds of thousands of REGISTERS. There are two approaches to avoid excessive number of refinements:

1. If using a strong resource-intensive proof algorithm, it is important to obtain a valid set of redundancy candidates from which the speculatively-reduced model is constructed. This implies more resources to be allocated to identify incorrect redundancy candidates through falsification using deeper symbolic analysis and semi-formal runs. The techniques we discuss in Section 3.6 were found to be particularly useful in such cases.

2. If using a faster and weaker proof algorithm such as $k$-step induction, it may not be advisable to spent large amount of resources upfront trying to weed out incorrect candidates. The highly inaccurate guessing in the beginning implies that the speculatively-reduced model will be easier to analyze in the earlier iterations. The approach is to make use of information from failed proof attempts to infer future failed proofs and pro-actively refine equivalence classes in the earlier iterations. We address these techniques in Section 3.4.2.

### 3.4.2 Induction Counterexamples

A weaker proof algorithm such as $k$-step induction is often not resource intensive, hence larger number of refinements is not necessarily a bottleneck when applying induction in the "prove" step. However, to ensure scalability and applicability of induction on designs with hundreds of thousands of gates, techniques nonetheless must be utilized to speed up induction and reduce the overall number of refinement iterations. $K$-step induction frameworks used to prove miters work as follows,

- **Base Case** - Validate that the miters are unassertable in the first $k$ time-steps starting from initial state.
- **Inductive Case** - Assuming that the miters are unassertable for the first $k$ time-steps starting from *any* state, check whether they can be asserted at time-step $k + 1$.

The key observation is that when induction fails to prove a miter unassertable, it will provide a trace starting from an inductive state showing how the miter can be asserted. The original design may then be initialized into that inductive state, then simulated using the input sequence from the counterexample [MCBJ08]. Any PRIMARY INPUTS that were not assigned in the original induction trace can be randomly assigned and this would provide us with an useful set of simulation patterns. The simulation patterns can be used for two purposes.

- If the simulation pattern shows other miters also being asserted, we can refine the particular redundancy candidates over which those miters were built without relying upon a failed induction run on those miters. This saves significant resources since simulation is often faster than induction.
- Secondly, refinement of a single equivalence class may lead to the creation of multiple new equivalence classes. We can use the simulation patterns generated through simulating the induction trace to check if the gates in the new equivalence classes can also be differentiated. Essentially, we are trying to

check whether an attempt to prove the gates in the newly generated equivalence class would be destined to fail through induction during the next refinement iteration. Every refinement weakens the induction hypothesis. If the gates in the new equivalence classes can be differentiated using the prior simulation patterns, we can refine the new equivalence classes since a failed proof attempt under a stronger induction hypothesis implies that proof attempts will fail under a weaker induction hypothesis. This is referred to as *proactive early refinement* since we are refining equivalence classes even without building miters over the redundancy candidates in the new equivalence classes.

The induction counterexamples can be used for refining as noted above irrespective of the nature of the SAT-solver used for induction, i.e., whether it is CNF-based or circuit-based. In practice, we have observed that circuit SAT solver [KPKG02] is much more powerful compared to a CNF-based SAT solver [MMZ+01] due to its ability to produce minimal assignments to PRIMARY INPUTS in its counterexample. The unassigned PRIMARY INPUTS can be randomly assigned which further randomizes the inductive initial state. This enables a huge reduction in the number of counterexamples generated by SAT-solver and thereby resources consumed by SAT-solver. Without this capability, resimulation is often less effective, motivating the use of approximating tricks such as *distance-1* simulation [MCBE06]. Post-processing CNF traces for minimal assignments [RS04] may be equally effective, though at a considerable added cost.

## 3.5 Sequential Redundancy Removal Engine

Using the sequential redundancy identification framework, we may compute the exact set of gates which are sequentially redundant. In theory, sequential redundancy identification is capable of proving every unassertable target in the netlist, since the unassertable targets correlate to gates which are semantically equivalent to 0. However, either due to computational resource limitations which result in suboptimal redundancy identification, or due to targets which are truly assertable, some targets may remain unsolved after the redundancy identification process.

If any targets remain unsolved after the redundancy identification process, one generally wishes to leverage the identified redundancy to simplify the netlist. This process is called redundancy removal, wherein once a pair of redundant gates are identified, the netlist is simplified by merging one of the gates onto the other. We propose a transformation engine (**EQV**) as part of the TBV framework which transforms the design through sequential redundancy removal. It is well known that verification algorithms often benefit from netlist simplification based on sequential redundancy removal [Kue04, MBP+04]. Examples of known algorithmic synergies which benefit from redundancy elimination include: faster and deeper exhaustive bounded search using SAT; greater reduction potential through transformations and abstractions such as combinational rewriting, retiming, and localization reduction; and enhanced inductiveness [MBP+04].

Encapsulating the sequential redundancy removal flow as a transformation engine has many benefits. This enables us to apply an arbitrary sequence of engines *before* **EQV**, to exploit characteristics of the problem which may enable alternate algorithms such as structural symmetry detection engine **ISO** to more quickly reduce its domain – as well as *after* **EQV**, to leverage the sequential redundancy removal as a synergistic preprocessing to subsequent engines. This flexibility is often critical for leveraging **EQV** in property checking, wherein redundancy removal alone is often inadequate to solve the problem.

Encapsulating the sequential redundancy elimination flow as the **EQV** engine has non-obvious benefits even within the "prove" step. For example, on very large designs with millions of gates, the cost of computing an optimal set of equivalence classes may become prohibitive. Additionally, if the design is non-inductive (which is often the case), one would need expensive transformation and verification algorithms to complete the proof. This implies that each refinement when would be very expensive. It may thus be advantageous to start with a conservative, more accurate, but incomplete set of equivalence classes. Starting with an accurate set of equivalence classes will help avoid the most costlier refinements on the larger design. One can then perform redundancy identification in phases. E.g., we may first instantiate an **EQV** that attempts only an accurate (yet incomplete) 1:1 REGISTER

pairing. We may then leverage a more aggressive **EQV** instance on the suboptimal speculatively-reduced model, attempting to find 1:1 *gate* pairings. This may be followed by yet another **EQV** instance attempting to find general $N{:}M$ gate pairings, possibly leveraging a variety of other transformations in this flow to further reduce the domain of the problem. For the largest designs, often the most efficient strategy is that of finding efficient transformations that safely chip away at size, until more exhaustive, albeit expensive, algorithms become applicable. In this example, we may defer the need for heavy-weight proof analysis until the most deeply-nested **EQV** instance when the problem domain is at its smallest, vs. suffer suboptimal merging or prohibitive proof resources directly in the first **EQV** instance.

## 3.6 Falsification using Speculatively-Reduced Model

Theorem 3.1 implies that we may perform incomplete search upon the speculatively-reduced model, and provided that none of the miters are asserted during that search, the verification results obtained during that effort are directly valid for the corresponding targets on the original design. This basically casts the miters from being *assumption checkers* [HCC$^+$00] for a redundancy removal proof to being *filters* through which to confirm that any analysis performed upon the speculatively-reduced model remains within the state-space for which the redundancy candidates are correct – even if they are not correct in all reachable states. Note that such an incomplete application is a generalization of the traditional use of these miters for redundancy-removal proofs; any proven miter may be safely discarded, as it cannot *invalidate* the analysis performed on the speculatively-reduced model.

For example, if we apply a sequence of test vectors to both models, and none of the miters in the speculatively-reduced model are asserted during that simulation, a target in the reduced model will be asserted if and only if that target is asserted in the original design under that simulation. This observation in turn implies that any number of symbolic evaluation steps – whether exact or underapproximate – performed without asserting any of the miters preserves the results obtained upon the targets during that analysis. This result allows us to exploit *even redundancy*

*that holds only for an initial bounded time-frame, but not across all time-frames*, to leverage the speculatively-reduced model for applications such as increasing the depth to which bounded falsification may be performed on the targets. A noteworthy dual of this reduction is that of [Kue04], allowing unfoldings to exploit redundancies that are proven to hold only *after* a certain number of time-steps. TBV will yield such a reduction only if a technique such as retiming sufficiently skews those gates to enable their merging. This result also allows us to construct and use the speculatively-reduced model in alternative proof-incapable frameworks such as simulators and hardware emulators. We have found several applications for this theory.

First, we have found many cases where we could perform bounded falsification on targets many times faster, and also deeper, on the speculatively-reduced model than on the original design. Second, we have found this approach useful in the candidate guessing process; after preliminary guessing via low-cost analysis of the original design, we build a speculatively-reduced model and apply a sequence of transformations to further reduce that model, and then apply more extensive semi-formal analysis on that reduced model to further attempt to differentiate the candidates.

Our third application allows us to reuse the knowledge that $a$ and $b$ cannot be differentiated for times $0, \ldots, i$ across refinements. In particular, if the equivalence class containing $a$ and $b$ is unaltered, we may immediately infer that the XOR of $a$ and $b$ cannot be asserted for times $0, \ldots, i$ regardless of the refinement of any other classes. Furthermore, if we refine the equivalence class containing $a$ and $b$ resulting in miters ($a$ XOR $c$) and ($b$ XOR $d$), we may immediately infer that these two new miters cannot be asserted for times $0, \ldots, i$. This optimization holds because refinements only split a class into several new classes, but never group gates from previously-incompatible classes. Practically, this application serves two goals. First, it enables us to reuse the discharging of the induction hypothesis from prior proof attempts to speed up later ones. Second, one may wish to intermix semi-formal analysis to assert miters with proof analysis to demonstrate their unreachability; this optimization helps reuse falsification effort across refinements.

| Design Info | | Spec. Red. Disabled | Spec. Red. Enabled | | Spec. Red. & Cex Sim | | Spec. Red. & Cex Analysis | | |
|---|---|---|---|---|---|---|---|---|---|
| **Name** | **Gates** | **Time (s)** | **Time (s)** | **Improve. w.r.t. prev** | **Time (s)** | **Improve. w.r.t. prev** | **Time (s)** | **Improve. w.r.t. prev** | **Improve. w.r.t. Col 3** |
| **IBUF** | 11424 | 9963.5 | 7525.1 | 1.32 × | 1670.7 | 4.50 × | 27 | 61.88 × | 367.56 × |
| **SSC** | 16385 | 6493 | 5235 | 1.24 × | 1121 | 4.67 × | 74.5 | 15.05 × | 87.15 × |
| **MIS3** | 18094 | 39752 | 6869 | 5.79 × | 1156.7 | 5.94 × | 88.9 | 13.01 × | 447.45 × |
| **MIS1** | 21044 | 22928.2 | 1304.8 | 17.57 × | 288.3 | 4.53 × | 18.5 | 15.58 × | 1240.05 × |
| **MIS2** | 22772 | 67861 | 1040.27 | 65.23 × | 222.5 | 4.67 × | 20.2 | 11.01 × | 3353.91 × |
| **SDQ** | 26666 | 16736.3 | 7263.2 | 2.30 × | 2082.6 | 3.49 × | 84.8 | 24.56 × | 197.14 × |
| **SBIU** | 30442 | 8408 | 4828 | 1.74 × | 820.2 | 5.89 × | 88.5 | 9.27 × | 95.00 × |
| **DAA** | 31424 | 41027.9 | 32449.8 | 1.26 × | 5926 | 5.48 × | 199.4 | 29.71 × | 205.14 × |
| **SMM** | 85574 | 846 | 83.5 | 10.13 × | 81.6 | 1.02 × | 8.5 | 9.6 × | 99.20 × |
| **FIER** | 89943 | 158938 | 15028 | 10.57 × | 9357.6 | 1.61 × | 20.3 | 461.00 × | 7845.16 × |
| **CIU** | 92232 | 6744.6 | 166 | 40.63 × | 138.6 | 1.2 × | 12.7 | 10.91 × | 531.92 × |
| **L2** | 374977 | 172800 (4) | 172800 (420) | 105 × | 28404.7 | 96.75 × | 214.3 | 132.54 × | 1346440 × |

Table 3.1: Induction-based redundancy identification results

In [NPMJ03], it is noted that one need not re-prove any miters that had no refined gates in their fanin because the prior proof is guaranteed valid after the refinement. We may extend our third application to generalize that of [NPMJ03] by noting that the assertion of a miter over $a$ and $R\big(Q(a)\big)$ at time $i$ only risks invalidating results beyond time $i$ for miters which contain $a$ in their fanin.

## 3.7 Experimental Results

In this section we provide experimental results to illustrate the power of our techniques. All experiments were run on a 2.1GHz POWER5 processor, using the IBM internal transformation-based verification tool *SixthSense* [MBP$^+$04].

We first provide experimental results for induction-based sequential redundancy identification, focusing on the advantages of using speculative reduction and proactive early refinement as described in Section 3.4.2 to improve scalability and reduce runtime. The results for induction-based redundancy identification is presented in Table 3.1. The benchmarks include industrial sequential equivalence checking examples as well as difficult industrial invariant checking examples. In the sequential equivalence checking examples, designs optimized via manual or automatic sequential synthesis optimizations are compared against original circuits. Four different flavors of SAT-based induction were run on these benchmarks for redundancy identification. These include:

1. SAT-based induction with speculative reduction disabled

2. SAT-based induction with speculative reduction enabled

3. SAT-based induction is done with speculative reduction enabled. In addition, counterexample traces from induction are simulated on the original model and the results of simulation are used to check for miters assertions [Kue04].

4. SAT-based induction is done with speculative reduction enabled. Counterexample traces from induction are simulated on the original design. In addition to using these traces to check if unsolved miters can be asserted, they are also used to perform proactive early refinement as described in Section 3.4.2.

The first two columns in Table 3.1 indicate the name of the design and size of the original netlist. Columns 3-9 indicate the runtime for each flavor of SAT-based induction along with the improvement in runtime for each flavor as compared to previous technique. Column 10 indicates the improvement in runtime when using induction with speculative reduction and proactive early refinement compared to default induction without speculative reduction.

Column 4 illustrates the importance of speculative reduction for improving the scalability of induction-based redundancy identification. Speculative reduction results in an order of magnitude speedup when compared to induction without speculative reduction. Simulation of induction counterexamples is surprisingly very powerful. It enables 3–4 $\times$ speedup on average by transferring some of the burden to solve the miters from SAT-based analysis to simulation-based analysis. Simulation is often faster than SAT in asserting miters and counterexample simulation is able to take advantage of this fact. The biggest surprise is the utility of proactive early refinement, it enables an order of magnitude speedup over the use of counterexample simulation. As illustrated through the experiments, combining speculative reduction with counterexample simulation and proactive early refinement is critical in improving the scalability of induction-based redundancy identification.

A particularly interesting testcase is **L2**. **L2** is a property checking testcase where certain features of the L2 cache are verified. The large arrays of the L2 cache

| Design Info | | Spec. Red. Disabled | Spec. Red. Enabled | | Spec. Red. & Cex Analysis | | |
|---|---|---|---|---|---|---|---|
| **Name** | **Gates** | **Miters Solved** | **Miters Solved** | **Improvement w.r.t. prev** | **Miters Solved** | **Improvement w.r.t. prev** | **Improvement w.r.t. Col3** |
| **IBUF** | 11424 | 1970075 | 1588746 | 1.24 × | 11978 | 132.64 × | 164.47 × |
| **SSC** | 16385 | 1182197 | 794239 | 1.49 × | 35554 | 22.34 × | 33.29 × |
| **MIS3** | 18094 | 3351149 | 1374897 | 2.44 × | 46505 | 29.56 × | 72.13 × |
| **MIS1** | 21044 | 2850774 | 642594 | 4.44 × | 13466 | 47.72 × | 211.88 × |
| **MIS2** | 22772 | 3180878 | 351192 | 9.06 × | 13784 | 25.49 × | 230.94 × |
| **SDQ** | 26666 | 719262 | 567757 | 1.27 × | 18456 | 30.76 × | 39.07 × |
| **SBIU** | 30442 | 1705755 | 1054606 | 1.62 × | 24062 | 43.83 × | 71.00 × |
| **DAA** | 31424 | 4836934 | 4147228 | 1.67 × | 44949 | 92.27 × | 154.09 × |
| **SMM** | 85574 | 172579 | 161487 | 1.07 × | 1944 | 83.07 × | 88.88 × |
| **FIER** | 89943 | 40101056 | 5161148 | 7.77 × | 7806 | 661.18 × | 5137.37 × |
| **CIU** | 92232 | 334274 | 92709 | 3.61 × | 4545 | 20.40 × | 73.64 × |
| **L2** | 374977 | 6207648900 | 61660373 | 100.07 × | 60234 | 1023.68 × | 102439.66 × |

Table 3.2: Miters solved during induction-based redundancy identification

are blackboxed to enable formal analysis on the design. Since we are targeting specific assertions to check certain properties of the design, some of the features of L2 cache were turned off and this results in a large amount of redundancy in the design. Using the techniques proposed in this dissertation, we were able to identify all redundant gates that can be inductively proved in the design in approximately 214 seconds. If speculative reduction is disabled, we are only able to complete 4 iterations of the "assume-then-prove" meta algorithm before we hit the timeout (which was set to 2 days or 172800 seconds). Based on the run which simulated counterexamples to cancel unsolved miters, 6680 iterations are required to reach the fixed-point. Since only 4 iterations were completed in 2 days, by extrapolating, it would have taken almost 10 years to reach the fixed-point if we had continued with the run without speculative reduction.

In order to understand the reason behind the improvement in runtimes through the use of speculative reduction, counterexample simulation and proactive early refinement, we obtained more data from induction runs. In particular, we obtained data on the total number of miters solved as well as the number of counterexamples from induction during the whole process.

The data on total number of miters solved during induction-based redundancy identification is presented in Table 3.2. For each benchmark (name and size of original netlist listed in Columns 1 and 2), we present the total number of miters

| Design Info | | Spec. Red. Enabled | Spec. Red. & Cex Sim | | Spec. Red. & Cex Analysis | | |
|---|---|---|---|---|---|---|---|
| **Name** | **Gates** | **Induction Cexs** | **Induction Cexs** | **Improvement w.r.t. prev** | **Induction Cexs** | **Improvement w.r.t. prev** | **Improvement w.r.t. Col 3** |
| **IBUF** | 11424 | 552623 | 13712 | 40.30 × | 1561 | 8.78 × | 353.83 × |
| **SSC** | 16385 | 202892 | 20974 | 9.67 × | 1869 | 11.22 × | 108.50 × |
| **MIS3** | 18094 | 368629 | 13060 | 28.23 × | 2145 | 6.09 × | 171.92 × |
| **MIS1** | 21044 | 230457 | 9310 | 24.75 × | 1511 | 6.16 × | 152.46 × |
| **MIS2** | 22772 | 136217 | 8275 | 16.46 × | 1391 | 5.95 × | 97.94 × |
| **SDQ** | 26666 | 177672 | 10827 | 16.41 × | 1974 | 5.48 × | 89.93 × |
| **SBIU** | 30442 | 372830 | 24403 | 15.28 × | 4711 | 5.18 × | 79.15 × |
| **DAA** | 31424 | 969845 | 49374 | 19.64 × | 6118 | 8.07 × | 158.49 × |
| **SMM** | 85574 | 157002 | 6097 | 25.75 × | 380 | 16.04 × | 413.03 × |
| **FIER** | 89943 | 5157805 | 638443 | 8.08 × | 581 | 1098.87 × | 8878.87 × |
| **CIU** | 92232 | 89946 | 8263 | 10.89 × | 434 | 19.04 × | 207.35 × |
| **L2** | 374977 | 29625219 | 271791 | 109.00 × | 1756 | 154.78 × | 16870.75 × |

Table 3.3: Induction counterexamples during redundancy identification

solved in each of the induction flavors and also indicate the improvement with respect to previous technique listed in the table. A particular flavor of induction is supposed to have 5 × improvement with respect to another flavor of induction if the total number of miters solved by the different flavor is 5 × more than the total number of miters solved by this induction flavor. We have not listed induction with counterexample simulation in this table since the total number of miters solved using this flavor of induction is not different from the flavor of induction that just uses speculative reduction.

As illustrated in Table 3.2, speculative reduction results in a 2 – 10 × reduction in the total number of miters solved by SAT. This is due to the fact that speculative merging facilitates logic reduction in the fanout of the redundancy candidates. The propagation of these logic reduction changes results in reduction in size of speculatively reduced netlist and number of distinct miters. The reduction in total number of miters solved explains part of the speedup in induction obtained through speculative reduction. Proactive early refinement further enables several orders of magnitude reduction in the number of miters solved by SAT. There are two reasons for this speedup, **(1)** Ability of early refinement to shield SAT from even attempting to solve certain miters that are destined to fail through induction, **(2)** The ability to prevent induction from solving the same set of unassertable miters multiple number of times through reduction in the number of refinement iterations.

Table 3.3 illustrates the advantage of using simulation to assert miters instead of relying on SAT-based analysis. Columns 1 and 2 indicate design info, Columns 3 – 8 provide details on number of counterexamples generated by SAT during induction (which is the same as the number of miters asserted by SAT). The columns also provide information on the improvement in terms of reduction in the number of induction counterexamples when comparing between two different flavors of induction. As illustrated in the table, use of counterexample simulation enables an order of magnitude reduction in the number of counterexamples generated during induction. The reduction in number of counterexamples is due to the ability of counterexample simulation to cancel unsolved miters. This reduces the amount of work for SAT and explains the improvement in runtime when using counterexample simulation in addition to speculative reduction. Proactive early refinement further reduces the number of induction counterexamples, in some cases by several orders of magnitude. This is due to the ability to drastically reduce the number of refinement iterations through early refinement of equivalence classes that are destined to fail through induction.

The next set of experimental results illustrate the impact of using flexible and robust set of transformation and verification algorithms to identify redundancy. As our solution is more general than previous approaches, we have chosen our experiments from difficult cases for which induction alone – even with variable-depth unique-state constraints [BC00] – was inadequate to solve the targets. We use the engines described in Section 2.1 as part of our TBV flow.

We present three sets of experiments in Table 3.4. The first set consists of industrial sequential equivalence checking examples, where manually redesigned circuits are compared against the original circuit. The redesigns include a variety of techniques, from retiming and resynthesis to replication of subcircuitry for reduced propagation delay to outright re-encoding of portions of the design. FPU is a floating-point unit; IFU is an instruction-fetch unit; SDQ is an issue queue; SMFC is a memory flow controller; BIU is a bus interface unit; and FXU is a fixed point unit. The second set consists of difficult industrial invariant checking problems. SCNTL is an issue controller; and SMM is a memory management unit. The third

| Name / Metric | Original Model | After Merging via Induction | After Merging via TBV |
|---|---|---|---|
| **FPU** | | 135 s / 255 MB | 2966 s / 789 MB |
| REGISTERS | 29030 | 20470 | 11600 |
| ANDS | 336060 | 305000 | 166749 |
| TARGETS | 263 | 258 | 0 |
| **IFU** | | 421 s / 412 MB | 646 s / 832 MB |
| REGISTERS | 80927 | 53402 | 33231 |
| ANDS | 645270 | 446579 | 296693 |
| TARGETS | 756 | 718 | 0 |
| **SDQ** | | 1079 s / 128 MB | 80 s / 98 MB |
| REGISTERS | 3790 | 2765 | 1833 |
| ANDS | 22466 | 19224 | 10950 |
| TARGETS | 310 | 261 | 0 |
| **SMFC** | | 2798 s / 357 MB | 173 s / 523 MB |
| REGISTERS | 39079 | 32797 | 17873 |
| ANDS | 270797 | 251273 | 151833 |
| TARGETS | 500 | 415 | 0 |
| **BIU** | | 609 s / 326 MB | 59 s / 448 MB |
| REGISTERS | 27461 | 15018 | 11334 |
| ANDS | 150402 | 76830 | 61272 |
| TARGETS | 908 | 554 | 0 |
| **FXU** | | 84 s / 236 MB | 63 s / 274 MB |
| REGISTERS | 12321 | 7395 | 4011 |
| ANDS | 76412 | 47935 | 31910 |
| TARGETS | 903 | 544 | 0 |
| **SCNTL** | | 377 s / 192 MB | 3065 s / 1113 MB |
| REGISTERS | 12401 | 3195 | 2121 |
| ANDS | 57706 | 22983 | 15563 |
| TARGETS | 57 | 53 | 0 |
| **SMM** | | 54 s / 448 MB | 284 s / 295 MB |
| REGISTERS | 2460 | 2214 | 1339 |
| ANDS | 70900 | 69611 | 45535 |
| TARGETS | 1 | 544 | 0 |
| **SYNTH1** | | 9 s / 55 MB | 18 s / 134 MB |
| REGISTERS | 1998 | 1815 | 954 |
| ANDS | 14757 | 11396 | 5885 |
| TARGETS | 68 | 65 | 0 |
| **SYNTH2** | | 2 s / 52 MB | 15 s / 123 MB |
| REGISTERS | 1568 | 924 | 720 |
| ANDS | 11649 | 8234 | 5733 |
| TARGETS | 242 | 72 | 0 |
| **SYNTH3** | | 430 s / 388 MB | 53 s / 175 MB |
| REGISTERS | 5180 | 4192 | 3845 |
| ANDS | 64431 | 57753 | 51965 |
| TARGETS | 969 | 277 | 0 |

Table 3.4: Sequential Redundancy Removal results

set, SYNTH1, SYNTH2, and SYNTH3, consists of sequential equivalence checking problems, where the original circuits are compared to the result of applying automatic retiming and resynthesis transformations using IBM's internal sequential synthesis tool.

The first column indicates the name of the corresponding design and the size metric being tracked in the corresponding row. The second column reflects the size of the target cones of the original, unreduced verification problem; simple structural hashing [KPKG02] was used to simplify them all. The sequential equivalence checking targets represent miters over corresponding primary outputs. The third column is the size of the original design after merging the gates proven equivalent using only induction. The induction depths were chosen on a per-design basis to minimize runtimes. The initial induction was limited to one hour to find the maximum depth at which any of the miters was proven; the reported results bounded each induction run to that depth. We also disabled unique-state constraints [BC00] if they did not yield greater merging. The fourth column is the size after merging the gates proven equivalent using TBV flows, after a low-cost induction preprocessing to eliminate the easier miters. The reported resources include all aspects of the verification process, including candidate guessing.

These experiments clearly demonstrate that the TBV approach results in significantly greater merging than possible with induction alone, and ultimately completes all unreachability proofs even though induction alone fails on most. Additionally, TBV is often faster than induction alone. Most of the TBV flows ultimately relied upon using interpolation or the use of localization followed by reachability analysis or interpolation to solve the more complex miters. Several exploited the ability of transformations such as retiming and structural state folding to enhance the inductiveness of targets [MBP+04]. Two flows relied upon identifying sequential redundancy in phases, starting with an accurate and conservative equivalence classing, followed by transformations to further reduce the speculatively-reduced model and another sequential redundancy identification using optimal equivalence classing.

To illustrate the power of TBV in processing the speculatively-reduced model,

| FPU | Initial | LOC | COM | CUT | LOC | RCH | |
|---|---|---|---|---|---|---|---|
| REGISTERS | 11600 | 1008 | 959 | 927 | 29 | | |
| ANDS | 167347 | 23543 | 20409 | 18838 | 132 | 2875s | |
| PRI. INPUTS | 1091 | 1059 | 1058 | 726 | 38 | 789 MB | |
| TARGETS | 112 | 1 | 1 | 1 | 1 | 0 | |
| **SDQ** | **Initial** | **COM** | **ITP** | | | | |
| REGISTERS | 1833 | 1313 | | | | | |
| ANDS | 11220 | 7728 | 70s | | | | |
| PRI. INPUTS | 401 | 301 | 98 MB | | | | |
| TARGETS | 61 | 52 | 0 | | | | |
| **SCNTL** | **Initial** | **EQV** | **LOC** | **CUT** | **RET** | **LOC** | **RCH** |
| REGISTERS | 1605 | 1515 | 216 | 169 | 125 | 76 | |
| ANDS | 11447 | 10572 | 819 | 690 | 540 | 312 | 3010s |
| PRI. INPUTS | 183 | 183 | 153 | 60 | 189 | 52 | 1113 MB |
| TARGETS | 62 | 44 | 1 | 1 | 1 | 1 | 0 |
| **SYNTH3** | **Initial** | **COM** | **MOD** | **RET** | **COM** | | |
| REGISTERS | 3856 | 2439 | 2437 | 7 | | | |
| ANDS | 54701 | 32534 | 25196 | 39518 | 48s | | |
| PRI. INPUTS | 854 | 428 | 428 | 1532 | 175 MB | | |
| TARGETS | 701 | 294 | 294 | 294 | 0 | | |
| **SMFC** | **Initial** | **COM** | **IND** | **MOD** | **COM** | **MOD** | **EQV** |
| REGISTERS | 17895 | 17888 | 17888 | 8678 | 8670 | 7062 | |
| ANDS | 155992 | 123247 | 123207 | 71699 | 71000 | 60812 | 165s |
| PRI. INPUTS | 523 | 522 | 522 | 363 | 325 | 275 | 523 MB |
| TARGETS | 1044 | 745 | 707 | 117 | 117 | 106 | 0 |

Table 3.5: TBV results on speculatively-reduced model

we detail several TBV results in Table 3.5, where the columns indicate the size of the problem *after* the corresponding transformation engine (indicated in the top row) was run. Under the final engine in the TBV flow, the resources consumed by the TBV flow on the speculatively-reduced model is listed. Note how the transformation engines synergistically reduce the size of the netlist [MBP+04]. For SDQ, interpolation was sufficient to solve all the non-inductive miters. For FPU, iterative application of localization renders three orders of magnitude reduction on the size of one of the miters. For SYNTH3, state folding followed by retiming and resynthesis is able to solve all the miters. Without the prior MOD, there are feedback loops which preclude retiming from optimally retiming the netlist. For SMFC, the initial EQV performed an accurate but incomplete redundancy candidate guessing. SAT-based induction using the IND engine was then used to solve some of the miters. The speculatively-reduced model was further simplified using an iterative application of MOD engine. The simplified speculatively-reduced model now becomes amenable to an aggressive application of induction-based sequential redundancy

| **SDQ** | **Spec. Red. Enabled** | **ITP** | | | | | | | **Spec. Red. Disabled** | **ITP** |
|---|---|---|---|---|---|---|---|---|---|---|
| REGISTERS | 1833 | | | | | | | | 3655 | |
| ANDS | 11220 | 51s | | | | | | | 27429 | 178200 s |
| PRI. INPUTS | 401 | 54 MB | | | | | | | 401 | 10575.5 MB |
| TARGETS | 61 | 0 | | | | | | | 2141 | 2141 |
| **IFU** | **Spec. Red. Enabled** | **LOC** | **ITP** | | **Spec. Red. Disabled** | **LOC** | **ITP** | | **Spec. Red. Enabled** | **ITP** |
| REGISTERS | 33231 | 4 | | | 66421 | 15441 | | | 33231 | |
| ANDS | 303620 | 22 | 646 s | | 686790 | 241461 | 178200s | | 303620 | 178200s |
| PRI. INPUTS | 1371 | 10 | 832MB | | 1371 | 12156 | 1678 MB | | 1371 | 7184 MB |
| TARGETS | 1035 | 1 | 0 | | 33879 | 1 | 1 | | 1035 | 435 |
| **SYNTH3** | **Original Model** | **COM** | **MOD** | **RET** | **COM** | **EQV** | | | | |
| REGISTERS | 5180 | 5180 | 5167 | 30 | 30 | 21 | | | | |
| ANDS | 64431 | 64408 | 49071 | 64952 | 47310 | 31590 | | | | |
| PRI. INPUTS | 854 | 854 | 930 | 6476 | 938 | 622 | | | | |
| TARGETS | 969 | 969 | 969 | 969 | 935 | 929 | | | | |
| **SYNTH4** | **Original Model** | **COM** | **MOD** | **EQV** | | | | **Original Model** | **COM** | **EQV** |
| REGISTERS | 4343 | 4343 | 4330 | 3370 | | | | 4343 | 4343 | 3371 |
| ANDS | 53754 | 53688 | 40817 | 35111 | | | | 53754 | 53688 | 52151 |
| PRI. INPUTS | 539 | 539 | 615 | 615 | | | | 539 | 539 | 331 |
| TARGETS | 597 | 597 | 597 | 0 | | | | 597 | 597 | 590 |
| **TLB** | **Original Model** | **COM** | **ISO** | **MOD** | **EQV** | | **Original Model** | **COM** | **MOD** | **EQV** |
| REGISTERS | 58202 | 56766 | 3133 | 3122 | 1570 | | 58202 | 56766 | 54042 | 28169 |
| ANDS | 609831 | 592728 | 20942 | 20935 | 19307 | | 609831 | 592728 | 325575 | 247956 |
| PRI. INPUTS | 356 | 81 | 29 | 29 | 29 | | 356 | 81 | 77 | 77 |
| TARGETS | 217 | 216 | 8 | 8 | 0 | | 217 | 216 | 208 | 208 |

Table 3.6: Advantages of using speculatively-reduced model

removal and all the miters are solved.

To illustrate the advantage of applying the transformation and verification algorithms on the speculatively-reduced model instead of the original model, we performed a set of experiments which are detailed in Table 3.6. We also did a couple of experiments to illustrate the utility of transformation algorithms in increasing the scalability and applicability of sequential redundancy identification. These results are also presented in Table 3.6. For SDQ, we ran the ITP engine on both the speculatively-reduced model with miters and the original model with miters. ITP was able to solve all the miters in the speculatively-reduced model in 51 seconds, consuming only 54 MB of memory. On the other hand, ITP was not able to even solve one miter in the original model with miters, it hit the timeout of 48 hours and consumed a total of 10575 MB of memory. This clearly illustrates

the advantage of utilizing the speculatively-reduced model for interpolation. For IFU, we did three sets of experiments, (**1**) we ran localization followed by interpolation on the speculatively-reduced model, (**2**) we ran interpolation directly on the speculatively-reduced model, (**3**) we ran localization on the original model with miters. Localization enables several orders of magnitude reduction in the number of REGISTERs for each miter (the statistics of only one such miter is shown in the table) and this enables ITP to easily solve the miter. Though SDQ illustrated the ability of interpolation to solve non-inductive miters, on bigger designs, the use of ITP alone is not sufficient. This is illustrated in the application of ITP directly on the speculatively-reduced model. ITP hits the timeout of 48 hours and there are 435 miters still unsolved at the time of the timeout. The advantage of utilizing speculatively reduced model for localization is illustrated through the size of the localized model after the application of LOC on the original model with miters. Applying LOC on the original design results in a localized design with 15,000 REGISTERs as opposed to just 10 REGISTERs when applying LOC on the speculatively-reduced model. Needless to say, ITP hits the timeout when applied on the localized model obtained from applying LOC on the original design.

For SYNTH3, we apply the same sequence of transformations that was applied to the speculatively-reduced model (illustrated in Table 3.5) to the original model followed by induction-based sequential redundancy removal using the EQV engine. There are 929 targets still left unsolved after EQV engine illustrating the advantage of applying TBV on the speculatively-reduced model. For SYNTH4 (a sequential equivalence checking problem), we illustrate the advantage of using the MOD engine to enable induction-based sequential redundancy removal.

TLB is a translation-lookaside buffer and is an equivalence checking problem where a simpler behavioral model of the TLB array used for simulation-based functional verification is compared against the synthesized model of TLB array consisting of large number of manual circuit design optimizations. Since arrays are regular structures, we use the structural isomorphism detection engine (ISO) to first equivalence-class isomorphic targets such that only representative target per equivalence class needs to be solved by sequential redundancy removal engine. Use of

| Name | Bounded Steps Completed | | | Col. 4 Time to Surpass Col. 2 Depth |
|---|---|---|---|---|
| | Original Design | Original with Miters | Spec. Reduced with Miters | |
| **FPU** | 17 | 12 | 24 | 604 s |
| **IFU** | 12 | 8 | 14 | 795 s |
| **SDQ** | 23 | 16 | 178 | 51 s |
| **BIU** | 105 | 101 | 32082 | 10 s |
| **SMFC** | 86 | 79 | 148 | 204 s |
| **SCNTL** | 30 | 22 | 61 | 197 s |
| **SMM** | 570 | 559 | 1131 | 591 s |
| **SYNTH1** | 88 | 63 | 143 | 1092 s |
| **SYNTH2** | 42 | 34 | 45 | 1716 s |
| **SYNTH3** | 66 | 51 | 70 | 2508 s |

Table 3.7: Search depths reached within one hour

ISO followed by MOD enables a huge reduction in the size of the netlist which enables induction-based EQV engine to solve all the miters. Without ISO, the problem is too big to be solved by induction-based EQV engine.

To illustrate the advantage of utilizing the speculatively-reduced model for incomplete search, we performed experiments to assess the effectiveness of SAT-based bounded analysis which are detailed in Table 3.7. We ran independently on the targets of the original design (Column 2); on the miters validating the equivalence classes on the original design *without* speculative merging, e.g., if using such bounded analysis to guess the redundancy candidates (Column 3); and on the corresponding miters of the speculatively-reduced model after a sequence of inexpensive transformations (Column 4). Recall that the miters are a superset of the original targets, and that our SAT-solver integrates redundancy removal algorithms; we additionally ran COM on the designs of columns 2 and 3 prior to unfolding. These columns indicate the bounded depth reached within a one-hour time limit, inclusive of all transformations. The fifth column indicates how long it took to reach the depth of the second column using the speculatively-reduced model. As expected, the third column attains a lesser depth than the second, illustrating how SAT-based candidate guessing tends to degrade due to the increased number of targets. Note that the fourth column enables orders of magnitude deeper miter validation in cases, in addition to significantly deeper validation of the original targets. The fifth column further illustrates the speed benefit of the fourth.

## 3.8 Related Work

Techniques for identifying redundancy in combinational designs have become quite robust, using an efficient hybrid of random simulation, BDD-based analysis, and SAT-based analysis [KPKG02, Kue04, GPB00]. These techniques may be readily extended to sequential designs to merge gates which may be demonstrated to be redundant in *any possible* state (reachable or unreachable), effectively treating state elements as PRIMARY INPUTS during their analysis. While such an approach is useful for efficiently reducing the size of sequential designs, it is too weak to identify gates which are redundant in all reachable states, but do not appear redundant in certain unreachable states.

There have been several approaches to compensate for the weakness of combinational redundancy removal upon sequential designs. For example, Huang et al. proposed a heuristic approximate reachability approach to capture certain unreachability constraints in an equivalence checking framework [HCC+00]. The technique of van Eijk uses BDD-based induction to demonstrate that suspected redundancy candidate gates truly are redundant [vE98]. Bjesse et al. increased the scalability of induction-based sequential redundancy removal by applying SAT instead of BDDs, and by using a *complete* variant of induction [BC00]. These approaches help to prevent spurious mismatches in unreachable states without suffering the computational expense of exact reachability analysis. However, some inconclusive results – which must be conservatively treated as mismatches – are still a practical inevitability with such techniques. Reachability analysis is computationally expensive, and the approximation necessary for its scalability weakens its conclusiveness; resource bounds weaken the conclusiveness of even complete techniques such as variable-depth unique-state induction [BC00] on larger designs.

The approach of using speculative reduction to enable faster and scalable induction-based sequential redundancy identification is also discussed in [MCBJ08, LC06]. In [MCBJ08], the authors further discuss the use of induction counterexamples and REGISTER partitioning to improve the scalability of induction-based sequential redundancy identification.

# Chapter 4

# Structural Reparameterization and Localization

## 4.1 Overview

As discussed in Chapter 3, use of a transformation-based verification framework consisting of a robust variety of synergistic transformation and verification algorithms is critical in ensuring the scalability and applicability of a sequential redundancy identification framework. Of particular importance are REGISTER reducing transformations such as localization and retiming, whose application is often unavoidable to identify all redundancy on designs with hundreds of thousands of REGISTERs. Though retiming and localization can enable dramatic REGISTER reductions, they have the drawback of increasing the PRIMARY INPUT count in the transformed netlist [KB01, WHL$^+$01, MBP$^+$04]. Automatic formal verification techniques generally require exponential resources with respect to the number of PRIMARY INPUTs of a netlist. For example, the complexity of a transition relation may grow exponentially with respect to the number of PRIMARY INPUTS, in addition to REGISTERs. The initial state encoding of a netlist may also grow exponentially complex with respect to the number of PRIMARY INPUTs used to encode that relation. Symbolic simulation – used for bounded model checking and induction – may require exponential resources with respect to the number of PRIMARY

INPUTs multiplied by the unfolding depth. A large PRIMARY INPUT count may thus render proof as well as falsification efforts inconclusive and thus can be bottleneck to scalable redundancy identification.

In this chapter, we propose a novel set of fully-automated techniques to enable maximal reduction of PRIMARY INPUTs in sequential netlists.

First, we present a novel form of reparameterization: as a sound and complete structural abstraction. We prove that this technique renders a netlist with PRIMARY INPUT count at most a constant factor of REGISTER count, and discuss how it heuristically reduces REGISTER count and correlation. These reductions may thereby enhance the application of a variety of verification and falsification algorithms, including semi-formal search and reachability analysis. More significantly, our structural reparameterization enables synergistic application with various other transformations such as retiming [KB01] and localization [WHL+01], which overall are capable of yielding dramatic iterative netlist reductions.

Second, we present a novel min-cut based localization refinement scheme tuned for yielding an overapproximated netlist with minimal PRIMARY INPUT count. Unlike traditional localization approaches which refine entire next-state functions or individual gates, ours augments gate-based refinement by adding gates within a min-cut over the combinational logic driving the localized cone to minimize localized PRIMARY INPUT count. We also address the use of localization to simplify initial value logic. Complex initial value cones arise in a variety of applications such as retiming, and may otherwise be fatal to proof analysis. Localization refinement algorithms may be used to reduce the PRIMARY INPUT count of such logic, effectively attempting to overapproximate the *initial states* of the design in a property-preserving manner.

Third, we detail the synergy that these reparameterization and localization transformations have with each other, and also with other transformations such as retiming and redundancy removal [vE98, KPKG02, MBPK05]. For example, the former approaches break interconnections in the design and reduce correlation among its REGISTERS, enabling greater REGISTER reductions through subsequent retiming and localization. Retiming and localization eliminate REGISTERS

which constitute bottlenecks to the reduction potential of reparameterization, enabling greater PRIMARY INPUT reductions through subsequent reparameterization.

## 4.2 Structural Reparameterization

In this section we discuss our structural reparameterization technique. We prove the correctness and optimality of this fully-automated abstraction, and discuss the algorithms used for performing the abstraction as well as for lifting abstract traces to ones consistent with the original netlist.

**Definition 4.1.** Consider a cut $N_{\mathcal{C}} \parallel N_{\overline{\mathcal{C}}}$ of netlist $N$ where $N_{\mathcal{C}}$ comprises PRIMARY INPUTS and combinational logic but no REGISTERS or target gates. A *structural reparameterization* of $N$ is a netlist $N' = N'_{\mathcal{C}} \parallel N_{\overline{\mathcal{C}}}$ such that $V_{\mathcal{C}}$ of $N$ is trace-equivalent to $V'_{\mathcal{C}}$ of $N'$ under the bijective mapping implied by the composition onto $N_{\overline{\mathcal{C}}}$.

**Theorem 4.1.** Let $N_{\mathcal{C}} \parallel N_{\overline{\mathcal{C}}}$ be a cut of netlist $N$, and $N' = N'_{\mathcal{C}} \parallel N_{\overline{\mathcal{C}}}$ be a structural reparameterization of $N$. The gates of $N_{\overline{\mathcal{C}}}$ in composition $N_{\mathcal{C}} \parallel N_{\overline{\mathcal{C}}}$ are trace-equivalent to those in $N'_{\mathcal{C}} \parallel N_{\overline{\mathcal{C}}}$ under the reflexive bijective mapping.

*Proof.* By Definition 2.17, any gate $u \in N_{\mathcal{C}}$ which sources an edge whose sink is in $N_{\overline{\mathcal{C}}}$, or is the initial value of a REGISTER in $N_{\overline{\mathcal{C}}}$, is an element of $V_{\mathcal{C}}$. Definition 2.10 thus implies that we may evaluate $N_{\overline{\mathcal{C}}}$ of $N$ from valuations to $V_{\mathcal{C}}$ independently of valuations to gates in $N_{\mathcal{C}} \setminus V_{\mathcal{C}}$; similarly for $N'$ and $V'_{\mathcal{C}}$. Since we compose each gate of $V_{\mathcal{C}}$ onto a trace-equivalent gate of $V'_{\mathcal{C}}$, this implies that $N_{\overline{\mathcal{C}}}$ of $N$ is trace-equivalent to $N_{\overline{\mathcal{C}}}$ of $N'$. □

Theorem 4.1 is related to the result that simulation precedence is preserved under Moore composition [GL94]. This theorem establishes the soundness and completeness of our structural reparameterization: we wish to replace $N_{\mathcal{C}}$ by a simpler netlist which preserves trace-equivalence, while ensuring that every target is in $\overline{\mathcal{C}}$ and thereby preserving property checking. Numerous aggressive state-minimization techniques have been proposed for such purposes such as bisimulation minimization; however, such approaches tend to outweigh the cost of invariant

1. Compute a cut $N_{\mathcal{C}} \parallel N_{\overline{\mathcal{C}}}$ of $N$ using an *s-t min-cut* algorithm, specifying the PRIMARY INPUTS as $s$, and the initial value gates, next-state function gates, REGISTERS, and target gates as $t$.

2. Compute the range of the cut as the set of minterms producible at $V_{\mathcal{C}}$ as a function of the REGISTERS in its combinational fanin.

3. Synthesize the range via the algorithm of Figure 4.2. The resulting netlist $N'_{\mathcal{C}}$ is combinational, and includes $V'_{\mathcal{C}}$
   which is trace-equivalent to $V_{\mathcal{C}}$ under composition with $N_{\overline{\mathcal{C}}}$.

4. Replace each $v \in V_{\mathcal{C}}$ by its correspondent in $V'_{\mathcal{C}}$, yielding abstract netlist $N' = N'_{\mathcal{C}} \parallel N_{\overline{\mathcal{C}}}$.

Figure 4.1: Structural reparameterization algorithm

checking [FV99]. Structural reparameterization is a more restrictive type of abstraction, though one which requires only lower-cost combinational analysis and is nonetheless capable of offering dramatic enhancements to the overall verification process.

We use the algorithm depicted in Figure 4.1 to perform the structural reparameterization. In Step 1, we compute an *s-t min-cut* of $N$. In Step 2, we compute the range of the cut using well-known algorithms as follows. For each $c_i \in V_{\mathcal{C}}$, we introduce a distinct parametric variable $p_{c_i}$, and we denote the function of $c_i$ – over REGISTERS and PRIMARY INPUTS in its combinational fanin – as $f(c_i)$. The range of the cut is $\exists I. \bigwedge_{i=1}^{|V_{\mathcal{C}}|} \left(p_{c_i} \equiv f(c_i)\right)$. In Step 3, we compute the replacement logic for $N_{\mathcal{C}}$ from the range. The replacement gate $r_{c_i}$ for $c_i$ may be computed using the algorithm of Figure 4.2, assuming that the range is represented as a BDD.[1] Note that the approach of [KS00] may also be used for this synthesis; the algorithm of Figure 4.2 is merely an alternative included herein for completeness, implemented using common algorithms and applicable to BDDs with inverted edges. When completed, each produced gate $r_{c_i}$ is trace-equivalent to $c_i$.

[1] In [CCK04], it is proposed to perform the range computation for symbolic simulation using SAT; their technique is also applicable in our framework for structural reparameterization.

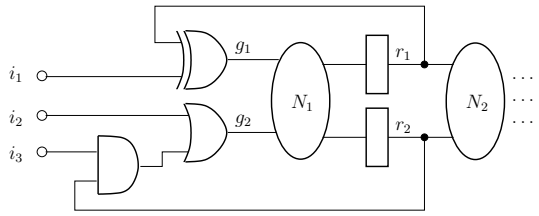
```
for (i = 1, ..., |V_C|) {    // Process i in rank order of variables p_{c_i} in BDD range
  b_i          = ∃p_{c_{i+1}}, ..., p_{c_n}.range;
  forced_0_i   = ¬b_i|_{p_{c_i}=1};
  forced_1_i   = ¬b_i|_{p_{c_i}=0};

  // SYNTH creates logic gates from BDDs. It creates a distinct PRIMARY INPUT to
  // synthesize each p_{c_i}. It processes "forced" terms using a standard multiplexor-based
  // synthesis, using r_{c_1}, ..., r_{c_{i-1}} as selectors for nodes over p_{c_1}, ..., p_{c_{i-1}} variables,
  // and using REGISTERs as selectors for nodes over their corresponding variables.
  // OR, AND, NOT create the corresponding gate types.
  r_{c_i}  =  OR ( SYNTH ( forced_1_i ),
                   AND ( SYNTH (p_{c_i} ),
                         NOT ( SYNTH ( forced_0_i ) ) ) );
}
```

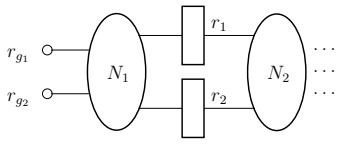
Figure 4.2: Range synthesis algorithm

Figure 4.3a illustrates an example netlist, where we wish to reparameterize a cut at gates $g_1$ and $g_2$. Gate $g_1$ has function $i_1 \not\equiv r_1$, and $g_2$ has function $i_2 \vee (i_3 \wedge r_2)$, where $i_1, i_2, i_3 \in I$ and $r_1, r_2 \in R$. The range of this cut is $\exists i_1, i_2, i_3.\big((p_{g_1} \equiv (i_1 \not\equiv r_1)) \wedge (p_{g_2} \equiv (i_2 \vee (i_3 \wedge r_2)))\big)$ which simplifies to $\top$. Replacement gates $r_{g_1}$ and $r_{g_2}$ are thus parametric inputs $p_{g_1}$ and $p_{g_2}$, respectively, and $r_1$ and $r_2$ are eliminated from the support of $V'_{\mathcal{C}}$ as illustrated in Figure 4.3b. While this abstraction is primarily intended for input elimination, this example illustrates its heuristic ability to reduce *correlation* between REGISTERs, here breaking any correlation through $N_1$ between the next-state functions of $r_1$ and $r_2$ and their respective present-state values. Additionally, note that if $N_2$ does not depend upon either of these REGISTERs (say $r_2$), that REGISTER will be eliminated from the abstracted netlist by reparameterization alone, illustrating the heuristic REGISTER elimination capability of this technique. This correlation reduction synergistically enables greater structural reductions through other transformation techniques such as retiming, as will be discussed in Section 4.4.

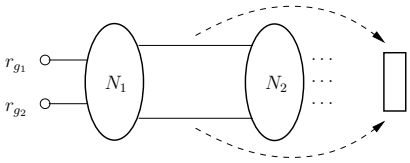
**Theorem 4.2.** The maximum number of PRIMARY INPUTs of the abstracted netlist




(a) Original netlist $N$. Cut gates are $g_1$ and $g_2$.




(b) Reparameterized netlist $N'$.




(c) Reparameterized and retimed netlist $N''$.

Figure 4.3: Reparameterization example

$N'$ generated by the algorithm of Figure 4.1 is $|T| + 2 \times |R|$.

*Proof.* An *s-t min-cut* algorithm may be guaranteed to return a netlist cut with $|V_{\mathcal{C}}| \leq \min(|s|, |t|)$, as follows from the following analysis. The bound of $|s|$ follows from the existence of a cut where $\mathcal{C} = s$. Noting that the min-cut is seeded with $s = I$, this guarantees that our algorithm cannot increase input count. The bound of $|t|$ follows by automatically preprocessing the netlist to ensure that each element of $t$ has indegree of one,[2] and selecting $\overline{\mathcal{C}} = t$. The seeded set $t$ comprises the target gates, as well as the REGISTERS' initial value and next-state function gates – a set of cardinality $|T| + 2 \times |R|$. The resulting cut $V_{\mathcal{C}}$ may thus be upper-bounded in cardinality by $\min(|I|, |T| + 2 \times |R|)$. At most one input is required per element of $V'_{\mathcal{C}}$ in $N'$, used in the synthesis of the parametric variable for that cut gate. The structural reparameterization thus replaces the $|I|$ PRIMARY INPUTS of $N_{\mathcal{C}}$ with the $|V'_{\mathcal{C}}|$ PRIMARY INPUTS of $N'_{\mathcal{C}}$.

Though we also add $R$ to $t$, this does not alter the above bound because the

[2]This preprocessing entails "splitting" a gate $v$ into gates $v_1$ and $v_2$. Gate $v_1$ has input connectivity and type identical to that of $v$, and fans out exclusively a new *buffer* gate $v_2$, which in turn inherits all fanout references of $v$ (including fanout edges, as well as target and initial value references). A similar approach is used to ensure that $s \cap t = \emptyset$ in Step 1 of the algorithm of Figure 4.1, e.g., in case a next-state function is also an input.

only gates sourcing an edge into the REGISTERs – their next-state functions – are seeded into $\overline{\mathcal{C}}$. This inclusion serves only to facilitate compositional reasoning, in that REGISTERs in the support of the synthesized range will appear in $N'$ – whereas $N_{\mathcal{C}}$ and $N'_{\mathcal{C}}$ are disjoint.

Let $U$ represent the set of gates which contain an input in their combinational fanin. Straight-forward analysis will demonstrate that $N'$ will have at most $(|T \cap U| + |\{r \in R : Z(r) \in U\}| + |\{r \in R : \exists u_1 \in U.(u_1, r) \in E\}|)$ PRIMARY INPUTs, which often yields a significantly tighter bound in practice. □

There are several noteworthy points relating to Theorem 4.2. First, note that at most one parametric input may be required per REGISTER for abstract initial values. This illustrates the duality between structural initial values and reachable state data, which is often represented with one variable per REGISTER. Certain techniques have been proposed which lock reachability data into structural initial values. For example, retiming [KB01] uses symbolic simulation to compute retimed initial values. If an input is retimed by $k$ time-steps, there may be $k$ unfolded copies of that input in the retimed initial values. Our structural reparameterization offsets this input amplification within the initial value data, similarly to how reparameterizing symbolic simulators operate [MKK$^+$02, CCK04]. As another example, one may underapproximate the reachable states (e.g., via symbolic simulation), then form a new netlist by altering the initial values of the original netlist to reflect the resulting state set [AS04, CCK04]. Second, aside from initial values, note that at most one parametric input per REGISTER is necessary for abstract next-state functions. This bound has significant potential for enhancing a variety of verification paradigms, especially when coupled with synergistic REGISTER-reduction techniques (e.g., localization and retiming).

Because our abstraction preserves trace-equivalence of all targets in $N_{\overline{\mathcal{C}}}$, demonstrating that a target cannot be asserted within a bounded or unbounded timeframe on the abstracted netlist implies the same result on the original netlist. However, if a trace is obtained asserting a target in the abstracted netlist, that trace must be *lifted* to indicate an assertion of the corresponding target in the original netlist. Our algorithm for trace lifting is provided in Figure 4.4. In Step 1, we simulate

1. Given partial trace $p'$ of $N'$, fully populate that trace up to the necessary length to assert the abstracted target, using binary simulation as per Definition 2.10 and injecting arbitrary values to any *don't cares* (unassigned values) of any PRIMARY INPUTS.

2. Cast a satisfiability check over $V_{\mathcal{C}}$ to obtain the same sequence of valuations as witnessed to $V'_{\mathcal{C}}$ in the populated trace $p'$. This check must be satisfiable since $V'_{\mathcal{C}}$ is trace-equivalent to $V_{\mathcal{C}}$ under composition with $N_{\overline{\mathcal{C}}}$, and yields trace $p''$.

3. Return trace $p$ produced by composing values to $N_{\overline{\mathcal{C}}}$ from $p'$ with values to $N_{\mathcal{C}}$ from $p''$.

Figure 4.4: Structural reparameterization trace lifting algorithm

the abstracted trace to ensure that we have adequate deterministic valuations to $V'_{\mathcal{C}}$ and $R'$ to enable the lifting. This is necessary because many verification algorithms produce *partial traces*, where certain valuations may be omitted for certain gates at certain time-steps. For example, in Figure 4.3b, parametric input $r_{g_1}$ replaced gate $g_1$ of function $i_1 \not\equiv r_1$, eliminating $r_1$ from the support of $V'_{\mathcal{C}}$. The abstracted trace $p'$ is thus less likely to include valuations to $r_1$. In order to lift $p'$, and thereby provide the proper sequence of valuations to $i_1$ to yield an identical sequence of valuations to $V_{\mathcal{C}}$, the trace-lifting process must be aware of the valuations to $r_1$. After simulation populates the necessary valuations to $p'$, a bounded satisfiability check in Step 2 will yield a trace $p''$ over $N_{\mathcal{C}}$ which provides the identical sequence of valuations to $V_{\mathcal{C}}$. This check tends to require only modest resources regardless of netlist size, since REGISTER valuations in $p'$ effectively break the $k$-step bounded analysis into $k$ one-step satisfiability checks, each injecting the netlist into the state reflected in the corresponding time-step of the trace. Step 3 splices $p'$ and $p''$ together, producing a consistent trace over the original netlist asserting the original target. This algorithm is similar to those for lifting traces over localized netlists (e.g., [CGKS02]); its primary difference is the binary simulation step, which reduces satisfiability resources and is enabled due to the soundness *and completeness* of our abstraction as per Theorem 4.1.

**Optimality.** Note that the algorithm of Figure 4.2 uses a single parametric input per cut gate. One may instead attempt a more aggressive synthesis of the range, using $\lceil \log_2 m \rceil$ variables to directly select among the $m$ possible minterms on a per-state basis (for maximal $m$), similarly to the approach proposed in [JG94]. While this may yield heuristically lesser input count, we have found this approach to be inferior in practice since $\lceil \log_2 m \rceil$ is often nearly equivalent to the cut-width due to the density of the range, and since the resulting encoding tends to be of significantly greater combinational complexity resulting in an increase in the analysis resources needed by virtually all algorithms, including simulation, satisfiability, and BDD-based algorithms (the latter was also noted in [AJS99]).

We may readily eliminate the $|T|$ contribution of the bound proven in Theorem 4.2 by using the structural target enlargement technique of [BKA02]. In particular, we may replace each target $t_i \in T$ by the synthesis of the characteristics function of the set of states for which there exists an input valuation which asserts that target, i.e., by $\exists I.f(t_i)$.

We utilize an *s-t min-cut* algorithm to ensure maximal input reductions as per Theorem 4.2. However, the range computation of the resulting cut may in cases be prohibitively expensive. It therefore may be desired to choose a cut with larger cardinality, weakening reduction potential in favor of computational efficiency – though iterative abstractions may be performed to ultimately converge upon the min-cut with lesser resources. In [MKK$^+$02] it is proposed to reparameterize a *group* $U$ of a candidate cut $V_{\mathcal{C}}$ to eliminate PRIMARY INPUTS $I_U$ which are in the combinational fanin of $U$ but not $V_{\mathcal{C}} \setminus U$. This reduction may be accomplished in our framework by selecting a cut of $V_{\mathcal{C}} = U \cup (I \setminus I_U)$, noting that any PRIMARY INPUTS in $V_{\mathcal{C}}$ will merely be replaced by other PRIMARY INPUTS, hence may effectively be treated as non-quantifiable variables when computing the range (similarly to REGISTERS in $N_{\overline{\mathcal{C}}}$). We have found that an efficient way to select suboptimal cuts for incremental abstraction is to compute min-cuts over increasing subsets of the desired cut, enabling the earlier abstractions to simplify later abstractions by iteratively decreasing $|I|$.

1. Begin with an initial abstraction $\mathcal{A}$ of $N$ such that $T \subseteq \overline{\mathcal{C}}$.
2. Attempt to prove or falsify each target in $\mathcal{A}$.
3. If the target is proven unreachable, this result is valid for $N$; return this result.
4. If a trace is obtained asserting the target in $\mathcal{A}$, search for a corresponding trace in $N$. If one is found, return this result.
5. Otherwise, the trace over $\mathcal{A}$ is spurious. Identify a refinement of $\mathcal{A}$ – i.e., a set of gates to move from $\mathcal{C}$ to $\overline{\mathcal{C}}$ – to eliminate the spurious trace. Repeat Step 2 with the refinement.

Figure 4.5: Localization refinement algorithm

## 4.3 Min-Cut Based Localization

**Definition 4.2.** A *localization* $\mathcal{A}$ of $N$ is a netlist obtained by computing a cut of $N$ such that $T \subseteq \overline{\mathcal{C}}$, and by replacing $V_{\mathcal{C}}$ by a set of PRIMARY INPUTS $V'_{\mathcal{C}}$ of netlist $N'_{\mathcal{C}}$, resulting in $\mathcal{A} = N'_{\mathcal{C}} \,\|\, N_{\overline{\mathcal{C}}}$. This is referred to as *injecting cut-points to* $V_{\mathcal{C}}$.

Localization differs from the structural reparameterization of Section 4.2 in that the localized netlist only trace-contains the original netlist. Thus verifying the localized netlist in place of original netlist is sound but incomplete. Because the overapproximation through localization may result in a spurious assertion of a target, *refinement* is often used to tighten the overapproximation by increasing the size of $\overline{\mathcal{C}}$, e.g., using the algorithm of Figure 4.5. For larger netlists, the localization may contain many thousands of PRIMARY INPUTS when using traditional approaches of selecting $V_{\mathcal{C}}$ to comprise only REGISTERS and PRIMARY INPUTS (e.g., [WHL+01, GGYA03]), or of refining individual gates. This large input count tends to render the BDD-based reachability analysis which is commonly used for the proof analysis in Step 2 infeasible. In [WHL+01, Wan03], this problem is addressed by further overapproximating the localization by computing an *s-t min-cut* between its PRIMARY INPUTS and sequentially-driven gates (i.e., gates which have a REGISTER in their combinational fanin), and injecting cut-points to the resulting

1. Select a set of gates to add to the refinement $\mathcal{A}'$ of $\mathcal{A}$ using an arbitrary algorithm. Let $\langle \mathcal{C}', \overline{\mathcal{C}}' \rangle$ be the cut of $N$ corresponding to $\mathcal{A}'$.
2. Compute an *s-t min-cut* $\langle \mathcal{C}_1, \overline{\mathcal{C}}_1 \rangle$ over $N$, with all gates in $\overline{\mathcal{C}}'$ as $t$, and $I \cup (R \cap \mathcal{C}')$ as $s$.
3. Add $\overline{\mathcal{C}}_1$ to the refinement $\mathcal{A}'$.

Figure 4.6: Min-cut based abstraction refinement algorithm

cut gates to significantly reduce localized PRIMARY INPUT count. When a trace is obtained on the post-processed localization, an attempt is made to map that trace to the original localization. If the mapping fails, in [Wan03] various heuristics are proposed to select REGISTERs to add for the next localization refinement phase, instead of directly addressing the causal post-process cut-point injection.

The min-cut based localization refinement scheme we have developed to minimize PRIMARY INPUT growth is depicted in Figure 4.6. In Step 1, a new localization $\mathcal{A}'$ is created from $\mathcal{A}$ by adding a set of refinement gates, which may be selected using any of the numerous proposed refinement schemes (e.g., [Wan03, CGKS02]). For optimality, however, we have found that the refinement should be at the granularity of individual gates vs. entire next-state functions to avoid locking unnecessary complex logic into the localization. In Step 2, an *s-t min-cut* $\langle \mathcal{C}_1, \overline{\mathcal{C}}_1 \rangle$ is computed over $N$. In Step 3, the gates of $\overline{\mathcal{C}}_1$ are added to $\mathcal{A}'$ to ensure that $\mathcal{A}'$ has as few PRIMARY INPUTs as possible while containing the original refinement of Step 1. Note that the newly-added gates are all combinational because all REGISTERs not already in $\mathcal{A}'$ are seeded into $s$, hence cannot be in the set ($\overline{\mathcal{C}}_1$) added to $\mathcal{A}'$.

Unlike the approach of [WHL$^+$01, Wan03], which *eliminates* gates from the logic deemed necessary by the refinement process hence is prone to introducing spurious counterexamples, our min-cut based localization *adds* combinational logic to the refinement to avoid this risk while ensuring minimal PRIMARY INPUT count. While the overapproximate nature of localization may nonetheless result in spurious counterexamples, our approach avoids the secondary overapproxima-

tion of theirs which is done without refinement analysis to heuristically justify its validity. Our more general approach also avoids adding unnecessary REGISTERs during refinement, since it has the flexibility to select which combinational logic to include. In our experience, many refinements may be addressed solely by altering the placement of the cut within the combinational logic. Additionally, our approach is often able to yield a localization with *lesser* PRIMARY INPUT count due to its ability to safely inject cut-points at gates which are sequentially-driven by REGISTERs included in the localization, which their REGISTER-based localization does not support and their combinational cut-point insertion disallows to minimize its introduction of spurious counterexamples. Finally, our approach enables localization to simplify complex initial value cones, as the inclusion of REGISTER $r$ does not imply the inclusion of its initial value cone. Only the subset of that cone deemed necessary to prevent spurious counterexamples will be added during refinement. This initial-value refinement capability has not been addressed by prior research, despite its utility – e.g., when coupled with techniques which lock reachability data into initial values such as retiming [KB01].

In a transformation-based verification framework [KB01, Bau02], one could attempt to reduce the PRIMARY INPUT count of an arbitrarily-localized netlist by using the structural reparameterization of Section 4.2 instead of using a min-cut based localization refinement scheme, or of overapproximately injecting cut-points to a combinational min-cut thereof as proposed in [WHL$^+$01]. As per Theorem 4.2, this synergistic strategy is theoretically able to reduce PRIMARY INPUT count to within a factor of two of REGISTER count. This bound is only possible due to the ability of reparameterization to abstract sequentially-driven logic. In contrast, the min-cut approach of [WHL$^+$01] is taken with $t$ being the set of all sequentially-driven gates, which is often much larger than the set of REGISTERs – hence PRIMARY INPUT count may remain arbitrarily larger than REGISTER count with their approach. Reparameterization is thus a superior PRIMARY INPUT-elimination strategy compared to the cut-point insertion of [WHL$^+$01], and has the additional benefit of retaining soundness and completeness. Nevertheless, the dramatic PRIMARY INPUT growth which may occur during traditional localization approaches often

entails exorbitant resources for reparameterization to overcome on large netlists. We have therefore found that an PRIMARY INPUT-minimizing localization scheme such as ours is necessary to safely minimize PRIMARY INPUT growth *during* localization, to in turn enable the optimal PRIMARY INPUT elimination of reparameterization with minimal resources.

## 4.4 Transformation Synergies

In a transformation-based verification (TBV) framework [KB01, Bau02], various algorithms are encapsulated as *engines* which each receive a netlist, perform some processing on that netlist, then transmit a new, simpler netlist to a child engine. If a verification result (e.g., a proof or counterexample) is obtained by a given engine from a child engine, that engine must map that result to one consistent with the netlist it received before propagating that result to its parent – or suppress it if no such mapping is possible. Synergistic transformation sequences often yield dramatic iterative reductions – possibly several orders of magnitude compared to a single application of the individual techniques [MBP+04]. In this section we detail some of the synergies enabled and exploited by our techniques.

Theorem 4.2 illustrates that all REGISTER-reducing transformations (e.g., retiming [KB01], redundancy removal [KPKG02, MBPK05], localization [WHL+01], and structural target enlargement [BKA02]) synergistically enable greater PRIMARY INPUT reductions through structural reparameterization. For example, retiming may find a minimal-cardinality REGISTER placement to eliminate reparameterization bottlenecks caused by their arbitrary initial placement. Localization injects cut-points to the netlist, which when reparameterized enable reductions even at *deep* gates which previously had no PRIMARY INPUTs in their combinational fanin. Redundancy removal and rewriting may enable *s-t min-cut* algorithms to identify smaller-cardinality netlist cuts.

In addition to its PRIMARY INPUT reductions, structural reparameterization reduces REGISTER correlation as per Figure 4.3b. As with redundancy removal, this often enables subsequent localization to yield greater reductions, since the heuristic

abstraction algorithms are less likely to identify unnecessary REGISTERs as being required to prevent spurious counterexamples. We have found iterative localization and reparameterization strategies to be critical to yield adequate simplifications to enable a proof *or* a counterexample result on many complex industrial verification problems. The concept of iterative localization strategies was also proposed in [GGYA03], leveraging the heuristics inherent in the SAT algorithms used for the abstraction to identify different subsets of the netlist as being necessary across the nested localizations, in turn enabling iterative reductions. Our TBV approach enables significantly greater reduction potential, since it not only allows the use of differing abstraction heuristics across nested localizations, but also allows arbitrary transformations to iteratively simplify the netlist between localizations to *algorithmically – not merely heuristically* – enable greater localization reductions. In cases, the result enabled through our iterative reductions was a *spurious* localization counterexample which could be effectively used by the causal prior localization engine for refinement. This illustrates the utility of our synergistic transformation framework for the generation of complex counterexamples for abstraction refinement, enabling a more general refinement paradigm than that of prior work, e.g., [WHL$^+$01, Wan03, GGYA03].

Retiming is limited in its reduction potential due to its inability to alter the REGISTER count of any directed cycle in the netlist graph, and its inability to remove all REGISTERs along *critical paths* of differing REGISTER count between pairs of gates [LS91]. Both reparameterization and localization are capable of eliminating such paths, enabling greater REGISTER reductions through retiming. This is illustrated in Figure 4.3b, where reparameterization eliminates the directed cycles comprising $r_1$ and $r_2$, enabling a subsequent retiming to eliminate those REGISTERs in Figure 4.3c. Retiming has the drawback of increasing PRIMARY INPUT count due to the symbolic simulation used to calculate retimed initial values [KB01]. Both reparameterization and our min-cut based localization are capable of offsetting this PRIMARY INPUT growth, enabling retiming to be more aggressively applied without risking a proof-fatal PRIMARY INPUT growth, as we have otherwise witnessed in practice.

| **S4863 [vE98]** | **Initial** | **COM** | **RET** | **COM** | **CUT** | | **Initial** | **COM** | **CUT** | **RET** | | | | **Resources** |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| REGISTERS | 101 | 101 | 37 | 37 | 21 | | 101 | 101 | 34 | 0 | | | | **1 sec** |
| PRI. INPUTS | 49 | 49 | 190 | 190 | 37 | | 49 | 49 | 21 | 21 | | | | **34 MB** |
| **S6669 [vE98]** | **Initial** | **COM** | **RET** | **COM** | **CUT** | | **Initial** | **COM** | **CUT** | **RET** | | | | |
| REGISTERS | 303 | 186 | 49 | 49 | 0 | | 303 | 186 | 138 | 0 | | | | **1 sec** |
| PRI. INPUTS | 80 | 61 | 106 | 81 | 40 | | 80 | 61 | 40 | 40 | | | | **35 MB** |
| **SMM** | **Initial** | **COM** | **LOC** | **CUT** | **LOC** | **CUT** | **LOC** | **CUT** | | | | | | |
| REGISTERS | 36359 | 33044 | 760 | 758 | 464 | 167 | 130 | 129 | | | | | | **229 sec** |
| PRI. INPUTS | 261 | 71 | 2054 | 666 | 366 | 109 | 135 | 60 | | | | | | **291 MB** |
| **MMU** | **Initial** | **COM** | **LOC** | **CUT** | **LOC** | **CUT** | **RET** | **COM** | **CUT** | | | | | |
| REGISTERS | 124297 | 67117 | 698 | 661 | 499 | 499 | 133 | 131 | 125 | | | | | **1038 sec** |
| PRI. INPUTS | 1377 | 162 | 1883 | 809 | 472 | 337 | 1004 | 287 | 54 | | | | | **386 MB** |
| **RING** | **Initial** | **COM** | **LOC** | **CUT** | **RET** | **COM** | **CUT** | **LOC** | **CUT** | **LOC** | **CUT** | **LOC** | **CUT** | |
| REGISTERS | 20692 | 19557 | 266 | 262 | 106 | 106 | 106 | 65 | 65 | 49 | 48 | 47 | 35 | **745 sec** |
| PRI. INPUTS | 2507 | 2507 | 568 | 280 | 726 | 587 | 480 | 452 | 376 | 330 | 263 | 259 | 64 | **240 MB** |
| **BYPASS** | **Initial** | **COM** | **LOC** | **CUT** | **LOC** | **CUT** | **LOC** | **CUT** | **LOC** | **CUT** | **LOC** | **CUT** | | |
| REGISTERS | 11621 | 11587 | 311 | 306 | 265 | 265 | 216 | 212 | 164 | 154 | 127 | 124 | | **240 sec** |
| PRI. INPUTS | 432 | 410 | 501 | 350 | 333 | 254 | 248 | 216 | 203 | 156 | 154 | 123 | | **175 MB** |

Table 4.1: Synergistic transformation experiments

## 4.5 Experimental Results

In this section we provide experimental results illustrating the reduction potential of the techniques presented in this paper. All experiments were run on a 2GHz Pentium 4, using the IBM internal transformation-based verification tool *SixthSense*. We use **COM, RET, CUT** and **LOC** engines in our experiments; each performs a cone of influence reduction.

We present several sets of experiments in Table 4.1 to illustrate the power of and synergy between these engines. The first column indicates the name of the benchmark and the size metric being tracked in the corresponding row. The second reflects the size of the original netlist; phase abstraction [BHSA03] was used to preprocess the industrial examples. The successive columns indicate the size of the problem *after* the corresponding transformation engine (indicated in the row labeled with the benchmark name) was run.

The first two examples in Table 4.1 are sequential equivalence checking proof obligations of SIS-optimized ISCAS89 benchmarks from [vE98]. The first presented flow demonstrates how **CUT** offsets the increase in PRIMARY INPUT count caused by **RET**, and also the REGISTER reduction potential of **CUT** itself. The second flow additionally illustrates how reparameterization enhances the

REGISTER-reduction ability of **RET**, enabling retiming to eliminate *all* REGISTERs from both benchmarks. **CUT** was able to eliminate significant REGISTER correlation – and thereby critical paths – in these benchmarks due to logic of the form $(i_1 \not\equiv r_1)$ and $i_2 \vee (i_3 \wedge r_2)$ as illustrated in Figure 4.3.

The remaining four examples are difficult industrial invariant checking problems. SMM and MMU are two different memory management units. RING validates the prioritization scheme of a network interface unit. BYPASS is an instruction decoding and dispatch unit. These results illustrate the synergistic power of iterative reparameterization and localization strategies, coupled with retiming, to yield dramatic incremental netlist reductions. The resulting abstracted netlists were easily discharged with reachability analysis, though otherwise were too complex to solve with reachability or induction. In SMM, the first **LOC** reduces REGISTER count by a factor of 43, though increases PRIMARY INPUT count by a factor of 29 to 2054. Without our min-cut based localization, this PRIMARY INPUT growth is even more pronounced. Refining entire next-state functions as per [WHL+01] yields 29221 PRIMARY INPUTs; their combinational cut-point injection may only eliminate 54 of these, as most of the logic is sequentially driven. **CUT** could eliminate 28514 of these given substantial resources. If refining individual gates, we obtain 2755 PRIMARY INPUTs. In practice, we often witness an even more pronounced PRIMARY INPUT growth through gate-based refinement vs min-cut based refinement e.g., 3109 vs. 1883 PRIMARY INPUTs for MMU. In MMU, **LOC** and **CUT** enable a powerful **RET** reduction with PRIMARY INPUT growth which is readily contained by a subsequent **CUT**. RING is a difficult example which **LOC** and **CUT** alone were unable to adequately reduce to enable reachability. **RET** brought REGISTER count down to an adequate level, though increased PRIMARY INPUT count substantially due to complex retimed initial values. A single **CUT** was unable to contain that input growth with reasonable resources, though the ability to safely overapproximate the initial value cones with **LOC** iteratively and synergistically enabled **CUT** to eliminate all but a single PRIMARY INPUT per initial value cone.

Table 4.2 illustrates the utility of structural reparameterization prior to unfolding. Column 2 and 3 illustrate the REGISTER and PRIMARY INPUT count of

| **Benchmark** | $\|R\|$ | $\|I\|$ Orig. | $\|I'\|$ Reparam. | $\|R\| \leq \|I\|$ Unfold Depth | $\|R'\| \leq \|I'\|$ Unfold Depth | $\|I\|$ Unfold Depth 25 | $\|I'\|$ Unfold Depth 25 | $\|I\|$ Unfold Depth 100 | $\|I'\|$ Unfold Depth 100 |
|---|---|---|---|---|---|---|---|---|---|
| **LMQ** | 345 | 189 | 135 (29%) | 6 | 8 | 3884 | 2735 (30%) | 17309 | 12111 (30%) |
| **DA_FPU** | 6348 | 534 | 240 (57%) | 24 | 39 | 7038 | 3120 (56%) | 47088 | 21120 (55%) |
| **SQMW** | 13583 | 1271 | 421 (67%) | 23 | 47 | 16356 | 4538 (72%) | 111681 | 36113 (68%) |

Table 4.2: INPUT counts with and without reparameterization prior to unfolding

the corresponding redundancy-removed [KPKG02] netlists. Column 4 provides the PRIMARY INPUT count of the reparameterized netlist; the numbers in parentheses illustrate percent reductions. Columns 5 and 6 illustrate the unfolding depth at which PRIMARY INPUT count exceeds REGISTER count with and without reparameterization. This is the unfolding depth at which one may wish to use reparameterization within the symbolic simulator to *guarantee* a reduction in variable count [CCK04]. Note that this depth is significantly greater for the abstracted than the original netlist. Practically, a bug may be exposed by the symbolic simulator *between* these depths, hence our approach may preclude the need for costly reparameterization on the unfolded instance. More generally, the *simplify once, unfold many* optimization enabled by our abstraction reduces the amount of costly reparameterization necessary over greater unfolding depths, and enables shallower depths to be reached more efficiently due to lesser variable count. Another noteworthy point is that REGISTER count is significantly greater than PRIMARY INPUT count in these netlists (as is common with industrial designs). Reparameterization within symbolic simulators operates on parametric variables for the REGISTERs, and on the *unfolded* PRIMARY INPUTs which become comparable in cardinality to the REGISTERs. In contrast, our structural reparameterization operates solely upon parametric variables for the *cut gates* (bounded in cardinality by the abstracted PRIMARY INPUT count, in turn bounded by the original PRIMARY INPUT count as per the proof of Theorem 4.2), and on the *original* PRIMARY INPUTs: a set of significantly lesser cardinality, implying significantly lesser resource requirements.

Note also that we did not perform more aggressive transformations such as localization and retiming on the examples of Table 4.2. As illustrated by Table 4.1, doing such is clearly a beneficial strategy in our synergistic framework. However, the aim of this table is to demonstrate how our structural reparameterization alone

benefits symbolic simulation. The final columns of this table show PRIMARY INPUT count with and without reparameterization for unfolding depths of 25 and 100.

## 4.6 Related Work

Several techniques have been proposed to reduce the number of PRIMARY INPUTs of a netlist for specific verification algorithms. For example, the approach of enhancing symbolic simulation through altering the parametric representation of subsets of the input space via manual case splitting strategies was proposed in [JG94, AJS99]. The approach of automatically reparameterizing unfolded variables during symbolic simulation to offset their increase over unfolding depth has also been explored, e.g., in [BO02, MKK+02, CCK04]. Various approaches for reducing variable count in symbolic reachability analysis have been proposed, e.g., through early quantification of PRIMARY INPUTs from the transition relation [MHF98], enhanced by partitioning [JKS02] or overapproximation [CCK+02].

Overall, our structural reparameterization technique is complementary to this prior work: by transforming the sequential netlist prior to unfolding, we enable a *simplify once, unfold many* optimization to bounded analysis reducing the amount of costly reparameterization needed over unfolding depth. Nonetheless, PRIMARY INPUT growth over unfolding depth is inevitable; while our technique reduces this growth, a reparameterizing symbolic simulator may nonetheless be beneficial for analysis of the abstracted netlist.

Our structural reparameterization approach is most similar to [MKK+02], which computes a cut of a logic cone, then parametrically replaces that cut by a simpler representation which preserves trace-equivalence. Unlike [MKK+02], which seeks to improve the efficiency of BDD-based combinational analysis hence retains all computations as BDDs, ours converts the reparameterized representation to gates. We are the first to propose the use of reparameterization as a structural reduction for sequential netlists, enabling its benefits to arbitrary verification and falsification algorithms, in addition to enabling dramatic iterative reductions with synergistic transformations as will be discussed in Section 4.4. Our approach also enables an efficient trace lifting procedure, unlike the approach of [MKK+02].

# Chapter 5

# Netlist Simplification in the Presence of Constraints

## 5.1 Overview

Sequential equivalence checking as well as property checking often require the specification of environmental assumptions to prevent reporting of uninteresting failures due to illegal input scenarios. Consider the case where the execute stage of an instruction pipeline is optimized based on the fact that the execute stage will not receive an instruction with illegal opcodes from the decode stage. One may wish to verify through sequential equivalence checking that the optimizations have not changed the functionality of the execute stage. This would require modeling environmental assumptions at the decode stage interface which guarantee the absence of illegal instructions.

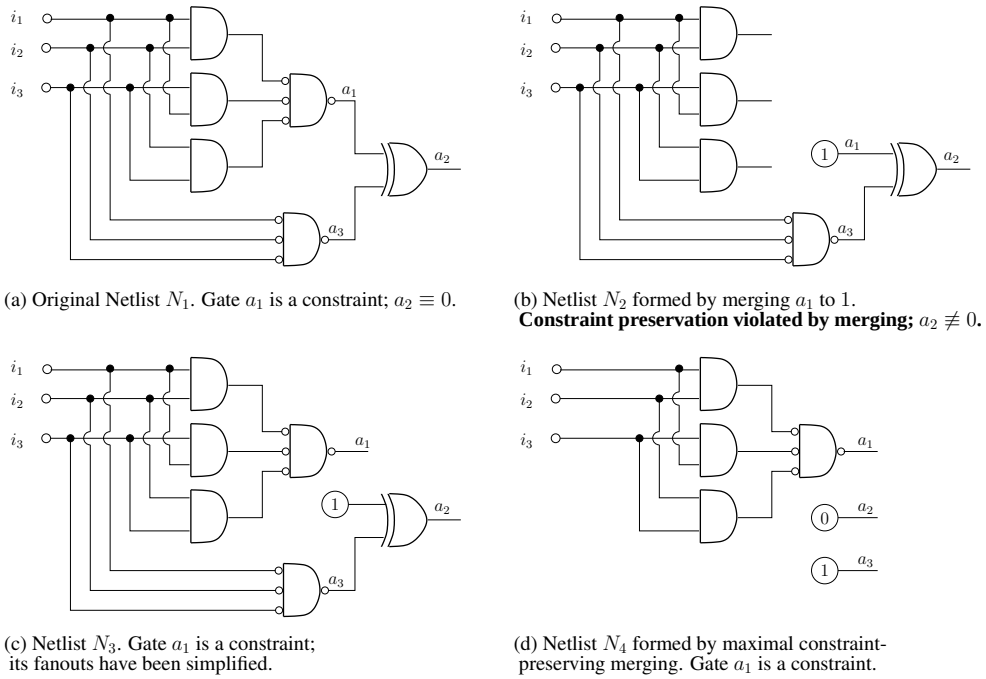
Specification of environmental assumptions is even more essential in property checking. Consider the environment for an instruction buffer that models the behavior of the instruction fetch unit and load-store unit. The assumptions the environment needs to satisfy would include requiring that instructions are not transferred into the buffer if doing so would overflow the buffer, and that *flush* operations initiated by the load-store unit due to operand lookup exceptions must target instruction tags of previously-dispatched instructions within the buffer. Most environments require a substantial number of assumptions, many of which involve

temporal handshaking with the outputs of the design.

There are two fundamental approaches to modeling environmental assumptions. First, one may utilize an imperative generator-style paradigm, where (possibly sequential) *filter* logic is used to convert nondeterministic data streams into legal input sequences. This filter logic is in turn composed with the design under verification [ZK03]. Second, one may utilize a declarative constraint-based approach, wherein illegal scenarios are enumerated using specific language constructs, and the verification toolset must ensure that these scenarios are not violated in any reported counterexample [Pix99].

Constraint-based testbenches have several advantages over generator-based approaches. In addition to the practical observation that declarative-style specification is often simpler than imperative [YPA05], the *checker-assumption duality* paradigm enabled by the use of constraints allows an assumption on the PRIMARY INPUTs of one design component to be directly reused as a checker on the outputs of an adjacent design component. This duality enables the guarantee of compositional correctness by cross-validating assumptions across adjacent components [NT00]. Constraints may also be used to implement case-splitting strategies to decompose complex verification tasks for computational efficiency [JG94, JWPB05]. Due to their benefits, constraints have gained wide-spread acceptance, and most verification languages provide constructs to specify constraints – e.g., through the *assume* keyword of PSL [Acca] and the *constraint* keyword of SystemVerilog [Accb].

Given their pervasiveness, it is important for verification algorithms to leverage constraints to enhance the overall verification process. However, it is even more critical to preserve constraint semantics during this process. In this chapter, we study the applicability of sequential redundancy identification and redundancy removal in the presence of constraints. We address the optimal simplification of netlists with constraints through redundancy removal. In particular, we provide the theoretical foundation with corresponding efficient implementation to extend scalable *assume-then-prove* sequential redundancy identification frameworks [vE98, BC00, HCC+00, MBPK05] to optimally leverage constraints.




(a) Original Netlist $N_1$. Gate $a_1$ is a constraint; $a_2 \equiv 0$.

(b) Netlist $N_2$ formed by merging $a_1$ to 1. **Constraint preservation violated by merging;** $a_2 \not\equiv 0$**.**

(c) Netlist $N_3$. Gate $a_1$ is a constraint; its fanouts have been simplified.

(d) Netlist $N_4$ formed by maximal constraint-preserving merging. Gate $a_1$ is a constraint.

Figure 5.1: Combinational constraint example

## 5.2 Constraint Challenges to Redundancy Removal

Consider sequential redundancy removal, wherein gates which are functionally equivalent in all reachable states may be merged to reduce design size. Because constraints restrict the set of reachable states, they may enhance reduction potential by enabling a pair of gates to be merged which are equivalent within the constrained reachable state space, but may not be equivalent otherwise. Viewed another way, the constraints imply a *don't care* condition which may be exploited when attempting to merge gates, similar to the exploitation of *observability don't cares* for enhanced reduction [ZKKSV06].

However, such merging risks weakening the evaluation of the constraints, resulting in spurious property violations. To illustrate constraint weakening through merging, consider the combinational netlist illustrated in Figure 5.1. In the original netlist $N_1$, gate $a_2$ could evaluate to 1 (e.g., if $i_1 = 1$ and $i_2 = i_3 = 0$) or 0 (e.g.,

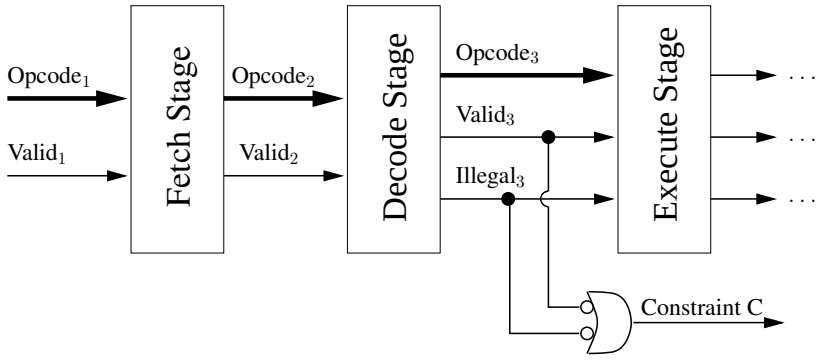



Figure 5.2: Constraint weakening through merging

if $i_1 = i_2 = i_3 = 0$). However, labeling gate $a_1$ as a constraint would force at least two of $i_1, i_2, i_3$ to evaluate to 1, in turn forcing gate $a_3$ to evaluate to 1 and $a_2$ to evaluate to 0. For optimality one would like to simplify the netlist using redundancy removal. In [KPKG02], a structural conjunctive decomposition of the constraint is proposed, traversing each constraint gate fanin-wise through AND gates and stopping at inversion points and other gate types, merging each of these terminal gates to constant ONE. Applying this algorithm to netlist $N_1$, gate $a_1$ will be merged to constant ONE. However, this merging *fails to preserve constraint semantics* as gate $a_2$ in the resulting netlist $N_2$ could evaluate to 1 (if $i_1 = i_2 = i_3 = 0$).

Constraint weakening through merging is not restricted to combinational netlists. To illustrate how constraints may be weakened through merging in a sequential netlist, consider the example in Figure 5.2 which includes the Fetch-Decode-Execute component of a processor. Assume that the testbench used to verify this component must prevent the Execute stage from encountering *valid* instructions with *illegal* opcodes. This may easily be achieved by constraining the output of the Decode stage to be *invalid* if its opcode is *illegal*. Next assume that the logic in the Fetch and Decode stages ensures that if the instruction in the Decode stage is invalid, its opcode is illegal. Coupled with the constraint, which enforces (Valid$_3$, Illegal$_3$) $\neq$ (1,1), this *satisfiability don't care* condition that (Valid$_3$, Illegal$_3$) $\neq$ (0,0) ensures that instructions in the Decode stage are invalid if and only if they

are illegal, within all reachable states. This fact may be leveraged by a redundancy removal algorithm, which may simplify fanout logic in the Execute stage accordingly. However, this fact will also entail redundancies in the fanin of the constraint – e.g., that the Valid and Illegal gates are antivalent – hence the redundancy removal framework may wish to perform the corresponding merges. While the simplification entailed by merging is often advantageous in reducing subsequent verification resources, constraint-enabled merges within the fanin of constraints may result in the constraints losing their ability to prevent invalid counterexamples. In this particular case, such merging could syntactically simplify the constraint gate to constant one, precluding its ability to prevent valid yet illegal instructions from propagating into the Execute stage.

Constraints pose several challenges to sequential redundancy identification frameworks described in Chapter 3. We refer to redundancy identification as *sound* if the identified gates are truly equivalent in all reachable states, and *complete* if all functionally equivalent gates are identified. Sequential redundancy identification frameworks based on "assume-then-prove" paradigm operate as per Figure 3.1. The speculative merging performed in Step 3 is necessary for scalability, since sequential redundancy identification is a PSPACE-complete problem [JB06]. This step requires additional consideration in the presence of constraints, since if the redundancy candidates are not truly equivalent, the speculative merge may alter constraint evaluation resulting in an unsound and incomplete procedure. We address this problem in Section 5.3. When this algorithm terminates, it will reflect *all* gates which are equivalent across all reachable states. However, in the presence of constraints, additional analysis is necessary before they may be merged to avoid violating netlist semantics. We address this topic in Sections 5.4 and 5.5.

## 5.3 Optimal Redundancy Identification under Constraints

Key to the scalability of sequential redundancy identification is the process of speculative reduction described in Chapter 3.2 and illustrated in Figure 3.2. Speculative reduction enables leveraging potentially equivalent gates to simplify the PSPACE-complete problem of sequential redundancy identification. Key to the optimality of redundancy identification in the presence of constraints is the ability to leverage those constraints to prune reachable states, thereby exposing equivalences which otherwise may not be identifiable. However, speculative merging in the presence of constraints raises two concerns:

1. Even if the speculative merges are for truly equivalent gates, the merges may weaken the constraints as per the examples of Figure 5.1 and 5.2. This may entail incompleteness of the redundancy identification process, since unreachable states may become reachable under the speculative merges.

2. At the time of speculative reduction, the validity of the merges as reflecting true redundancy has not been demonstrated. Thus, the speculative merges risk arbitrarily altering constraint evaluation, which may weaken *or strengthen* their evaluation, causing the redundancy identification framework to become unsound since reachable states may become unreachable.

In the absence of constraints, the correctness of using speculative reduction for sound and complete redundancy identification using assume-then-prove frameworks depicted in Figure 3.1 has been proven in Chapter 3.2.1. However, as noted above, in the presence of constraints, speculative reduction runs the risk of causing an unsound and incomplete redundancy identification. The following theorem establishes the correctness of a slightly modified form of speculative reduction in netlists with constraints, which ensures that miters for inequivalent redundancy candidates will remain assertable, and others will remain unassertable.

**Theorem 5.1.** Given a netlist $N$, consider netlist $N'$ formed by adding to $N$ a replication of the combinational fanin of each constraint, and re-labeling the replicated counterpart of each constraint gate as the constraint. Consider netlist $N''$, derived by speculative reduction of $N'$, though restricting merging to the original gates of $N$. Using $N''$ as the basis of sequential redundancy identification is sound and complete in identifying redundant gates in $N$.

*Proof.* Consider any trace $p$ over $N$, and the corresponding trace $p''$ obtained by simulating the input sequence of $p$ on $N''$. To demonstrate soundness and completeness of redundancy identification, we prove a stronger condition: that either $p$ illustrates no mismatches within any equivalence class *and* no miters are asserted in $p''$ (completeness), or that a nonempty subset of earliest-occurring mismatches illustrated in $p$ has a corresponding subset of earliest-occurring miter assertions in $p''$ (soundness). We prove this condition by induction on this earliest time-frame.

*Base Case: Time 0.* As discussed in Chapter 3, to ensure the soundness of speculative reduction, all equivalence classes must be validated as accurate at time 0. Thus no mismatches may be illustrated in $p$. Additionally, if $p$ is valid at time 0, the speculative merging cannot alter any valuations in $p$ vs. $p''$ at time 0 hence there can be no miter assertions at time 0. Furthermore, since there are no speculative merges in the combinational fanin of the constraints, their evaluation and hence their constraining power over REGISTERS and PRIMARY INPUTS cannot be altered, and thus $p$ is valid at time 0 if and only if $p''$ is valid at time 0

*Inductive Case: Time $i+1$.* By the induction hypothesis, assume that no mismatches nor miter assertions occurred at times $0, \ldots, i$. We prove that the theorem holds at time $i+1$. If no mismatches in $p$ nor miter assertions in $p''$ manifest at time $i+1$, our proof obligation is trivially satisfied. Otherwise, first consider the case that there is a nonempty set of gates $A$ which differ from their representatives at time $i+1$ in $p$. Consider the maximal subset $B \subseteq A$ such that no element in $b \in B$ has any element of $A \setminus b$ in the combinational fanin of $b$ or $R(b)$. Clearly $B$ is non-empty: gate $b$ cannot directly be speculatively merged, nor transitively through a sequence of speculative merges, onto a representative which lies in its combinational fanout without introducing a combinational cycle. Note that any speculative

merging in the fanin of $b$ or $R(b)$ cannot alter their valuation at time $i+1$, because that merging did not alter the valuation of any gate at times $0, \ldots, i$ by the induction hypothesis, nor at time $i+1$ due to the construction of $B$. Thus, the miters correlating to every gate in $B$ must assert at time $i+1$ in $p''$. This same argument can be used to demonstrate that a nonempty set $B$ of miter assertions at time $i+1$ in $p''$ must have a corresponding set of mismatches in $p$.

We finally must demonstrate that $p$ is valid at time $i+1$ if and only if $p''$ is valid at time $i+1$. Note that the state of the REGISTERs in the combinational fanin of the constraints must be identical across $p$ vs. $p''$ at time $i+1$, since their next-state functions evaluated the same at times $0, \ldots, i$ by the induction hypothesis. Since there are no speculative merges in the combinational fanin of the constraints, any mismatches at time $i+1$ cannot affect their *valuation* at that time-frame, nor their *evaluation* hence their constraining power over REGISTERs and PRIMARY INPUTs cannot be altered. Thus $p$ is valid at time $i+1$ if and only if $p''$ is valid at time $i+1$.

We thus conclude that either no mismatches occur in $p$ nor miter assertions in $p''$ at time $i+1$, or a nonempty set of speculatively merged gates $B$ will illustrate miter assertions in $p''$ and mismatches in $p$ at time $i+1$. □

## 5.4 Redundancy Removal under Constraints

Using the framework discussed in Section 5.3, we may compute the exact set of gates which are equivalent in the constrained reachable state space. In theory, this framework is capable of proving every unassertable target in a testbench, since they correlate to gates which are semantically equivalent to 0. However, either due to computational resource limitations which result in suboptimal redundancy identification, or due to targets which are truly assertable, some targets may remain unsolved after the redundancy identification process.

If any targets remain unsolved after the redundancy identification process, one generally wishes to leverage the identified redundancy to simplify the netlist so that a subsequent verification strategy may benefit from that simplification [MBPK05, Kue04, MBP+04]. Examples of known algorithmic synergies which benefit from

redundancy elimination include: faster and deeper exhaustive bounded search using SAT; greater reduction potential through transformations and abstractions such as combinational rewriting, retiming, and localization reduction; and enhanced inductiveness [MBPK05, MBP+04]. In this section, we discuss how to optimally leverage this identified redundancy.

It was demonstrated in [MBA05] that a merge of gate $g_1$ onto $g_2$ is guaranteed to preserve property checking as long as no constraint $c_1$ in the fanout of $g_1$ was used to constrain the state space during the proof of $g_1 \equiv g_2$. This implies that certain identified equivalent gates may be safely merged. However, this approach is suboptimal since certain merges which do not adhere to this criterion may nonetheless be performed while preserving property checking. As a trivial example, if one adds a constraint to a netlist which does not restrict its reachable states whatsoever, there is no reason that such a constraint preclude any merges regardless of fanout connectivity. As another example, if one adds a constraint to a netlist which has only unassertable targets, weakening those constraints through merging equivalent gates cannot cause spurious counterexamples. In practice, it is difficult to assess *which* of the identified redundancies is conditional upon *which* of the constraints (e.g., vs. holding due to satisfiability don't cares alone), without performing numerous independent redundancy identification analyzes with different subsets of constraints. Performing multiple redundancy identification analyzes is computationally expensive, thus motivating our technique to optimally leverage the redundancy identified using *all* constraints within a single run of the algorithm of Figure 3.1.

The following theorem establishes that we may safely perform a greater degree of merging than enabled by the results of [MBA05].

**Theorem 5.2.** Consider any set of gate pairs which have been proven equivalent across all reachable states in netlist $N$, and whose merged *from* gates (vs. merged *onto* gates) do not lie in the combinational fanin of any constraint. Performing the corresponding merges to yield netlist $N'$ will ensure that $N'$ is trace-equivalent to $N$.

*Proof.* Similarly to Theorem 5.1, since no gates within the combinational fanin

of constraints are merged onto others, their evaluation and hence their constraining power over REGISTERs and PRIMARY INPUTs cannot be altered. The initial states of $N$ and $N'$ are thus identical; any trace of length 1 in $N$ is valid in $N'$ and vice-versa. Furthermore, since the merges have been verified to reflect equivalence across the constrained reachable states, they cannot alter next-state function valuations. Thus, by simple inductive reasoning relative to any trace prefix of length $i$, any trace of length $i+1$ in $N'$ is also valid for $N$ and vice-versa. □

Theorem 5.2 enables us to perform a significantly greater degree of merging than enabled by the results of [MBA05], particularly for highly cyclic netlists where most of the netlist lies in the fanin of the constraints. However, we note that the additional merging enabled by this theorem is itself suboptimal, since the combinational fanin of the constraints may be arbitrarily large and, as per the examples preceding Theorem 5.2, there may still be verification-preserving merging possible therein. We now prove that general constraint-enhanced merging is sound but incomplete.

**Theorem 5.3.** Consider any set of gate pairs which have been proven equivalent across all reachable states in netlist $N$. Performing the corresponding merges to yield netlist $N'$ will ensure that $N'$ trace-contains $N$.

*Proof.* The fact that the gate pairs have been proven equivalent across all reachable states ensures that, within all trace prefixes which do not violate constraints in $N$, these merges do not alter the valuation of any gate whatsoever in $N'$. Thus, every constraint-satisfying trace prefix in $N$ is also valid in $N'$. However, the fact that merging may be performed in the combinational fanin of the constraints means that their evaluation relative to REGISTERs and PRIMARY INPUTs may be altered. In particular, for extensions to valid trace prefixes in $N$ – i.e., for time-frames $i+j$ (for non-negative $j$) where some constraint in $N$ is violated at time $i$ – the merges may alter the evaluation of gates in their fanout. In doing so, these merges may cause constraints in their fanout to evaluate to 1 in $N'$ vs. to 0 in $N$. Thus, constraint-satisfying trace prefixes in $N'$ may violate constraints in $N$. Netlist $N'$ thus trace-contains $N$, but generally may not be trace-equivalent to $N$. □

**Given:** Netlist $N'$ formed from $N$ by merging equivalent gate pairs $G_f \mapsto G_t$,
and traces $p'_1, \ldots, p'_i$ over $N'$

1. Simulate each $p'_i$ on $N$ to obtain trace $p_i$
2. If $p_i$ asserts any target $t$ in $N$, report that result as a valid counter-example and eliminate $t$ from $T$, the targets of $N$
3. If $T$ is empty, exit
4. Identify the set of merged nodes $D_1 \subseteq G_f$ of $N'$ which differ in valuation across any $p_i$ and $p'_i$ within the prefix of $p'_i$, ignoring constraint violations in $p_i$
5. Construct netlist $N''$ from $N$, performing conditional merges for the differing subset of $G_f$ enumerated in set $D_1$
6. Cast a SAT problem conjuncting over each $p'_i$, checking for the lack of an assertion of targets in $N''$ under the input sequence of $p'_i$
7. Define causal merge set $D_2$ as those whose selector is assigned to 0 from the SAT solution
8. Form refined netlist $N'''$ from $N$ by merging $G_f \setminus D_2$

Figure 5.3: Trace refinement algorithm

Theorem 5.3 implies that, if a target $t'$ is proven as unassertable in $N'$, the corresponding target $t$ must be unassertable in $N$ – though counterexamples from $N'$ may be invalid on $N$. This theorem motivates an abstraction-refinement framework conceptually similar to [CGJ+00], which retains those merges which do not entail spurious counterexamples for efficiency of a subsequent verification strategy, while discarding the others.

### 5.4.1 Abstraction-Refinement Framework

**Definition 5.1.** A *conditional merge* from gate $g_1$ onto gate $g_2$ consists of replacing $g_1$ by a multiplexor selected by a newly-created nondeterministic constant[1] gate $i_{g_1,g_2}$. If the selector evaluates to 1, the value of $g_2$ is driven at the output of the multiplexor. Otherwise, the original value of $g_1$ is driven at the output of the multiplexor.

We present an algorithm in Figure 5.3 for identifying a set of merges (hereafter referred to as *causal merges*) to discard in response to a set of spurious coun-

[1] A nondeterministic constant may be represented in a netlist by a REGISTER whose next-state function is itself (hence it never toggles), and whose initial value is a PRIMARY INPUT.

terexamples. The abstracted netlist $N'$ is formed by merging each $g_f \in G_f$ onto the corresponding $g_t \in G_t$ as per the surjective mapping $G_f \mapsto G_t$. While this algorithm only requires a single trace to refine upon, it is generalized to operate upon an arbitrary set of traces to enable optimal refinement as will be discussed in Section 5.5. Step 1 of this algorithm maps each trace $p_i'$ obtained from $N'$ to trace $p_i$ over $N$, from which we may assess the behavior of $N$ under the input sequence demonstrated in $p_i'$. Step 2 checks if any resulting trace constitutes a valid assertion of any target in $N$. If so, the resulting trace is reported as a valid counterexample and the corresponding target is eliminated from the set of unsolved targets $T$. If $T$ becomes empty, then the verification problem associated with the netlist has been solved hence the algorithm exits in Step 3. Otherwise, the algorithm proceeds to identify a set of merges which were responsible for the spurious counterexamples. Step 4 discerns an overapproximation $D_1$ of the set of causal merges, identifying those merged gates whose valuations differ between any $p_i$ and $p_i'$ during the prefix of $p_i'$. Ideally, this trace is minimally assigned so as to illustrate the assertion of a target in $N'$. The set $D_1$ may next be minimized in Steps 5-7 by casting a SAT problem which seeks to avoid any target assertions under the input sequence of each $p_i'$ within netlist $N''$ formed from $N$ by performing *conditional merges* for the potentially causal merges. Step 8 constructs a refined netlist, eliminating those merges which were determined to cause the spurious counterexamples in $N'$.

**Theorem 5.4.** Given trace $p_i'$ which is a counterexample to $t_i'$ of netlist $N'$, the algorithm of Figure 5.3 will either yield a valid counterexample $p_i$ to target $t_i$ of $N$, or produce a refined netlist $N'''$ which will not exhibit a spurious assertion against the input stimuli of $p_i'$.

*Proof.* This theorem is trivially true if the algorithm produces a counterexample on $N$. Otherwise, we note that:

- When constructing $D_1$, gate inequivalence is checked for all time-frames in the prefix of $p_i'$, which will illustrate the spurious assertion of $t_i'$. This check ignores constraint-based prefixing within $p_i$ to enable detecting *which* merges

**Given:** Netlist $N$; Initialize $i = 1$
1. Compute equivalence classes in $N$ using a variant of the algorithm of Figure 3.1 as per Theorem 5.1, yielding desired merges $G_f \mapsto G_t$
2. Form $N'$ by performing the subset of merges $G'_f \mapsto G_t$ which adhere to Theorem 5.2
3. Form $N_1$ from $N'$ by performing the remaining merges $G''_f \mapsto G_t$
4. Use an arbitrary set of algorithms to attempt to prove or falsify the targets in $N_i$
5. If any targets were proven unassertable on $N_i$, report the corresponding targets unassertable for $N$
6. If a nonempty set of counterexamples $P_i$ were obtained on $N_i$, use Figure 5.3 on netlist $N^*$ vs. $N$ and trace set $P^*$ to obtain a valid set of counterexamples and / or a refined netlist $N_{i+1}$, else exit
7. Report any valid counterexamples that were obtained for $N$
8. If unsolved targets remain, increment $i$ and goto Step 4, else exit

Figure 5.4: Abstraction-refinement framework. $N^* = N_i$ and $P^* = P_i$ in Step 4 implies *standard abstraction-refinement*; $N^* = N_1$ and $P^* = \bigcup_{j \in \{1,\ldots,i\}} P_j$ in Step 4 implies *optimal abstraction-refinement.*

in $N'$ weakened the constraints in $N$. Thus $D_1$ will include all merged gates whose behavior was altered in $p'_i$ vs. $p_i$.

- The SAT problem is satisfiable, since if the selector of each conditional merge construct is set to 0, the SAT solution will be equivalent to $p_i$ which has been demonstrated not to assert $t_i$ in $N$.

- Set $D_2$ by construction enumerates the subset of merges which, if eliminated, will prevent the assertion of $t'''$, the counterpart of $t'$ in $N'''$.

□

As per Theorem 5.4, the refinement process of Figure 5.3 is adequate to ensure that the resulting netlist will not exhibit a spurious assertion against any counterexamples used as the basis of the refinement. This result allows us to develop an abstraction-refinement framework which is guaranteed to converge as presented in Figure 5.4.

**Theorem 5.5.** The algorithm of Figure 5.4 (run in *standard abstraction-refinement* mode) will converge upon a correct verification result for the original netlist $N$.

*Proof.* We consider the individual steps of this algorithm.

- Step 1 computes the equivalence classes of gates, which are correct as per Theorem 5.1. We may select an arbitrary set of desired merges consistent with these equivalence classes, reflected by surjective mapping $G_f \mapsto G_t$.
- Step 2 performs those subset of merges (denoted as $G'_f \mapsto G_t$) which are guaranteed to preserve trace-equivalence as per Theorem 5.2. Thus, if verification were performed directly upon $N'$, all results would be correct with no need for refinement.
- Step 3 performs the remaining merges, denoted as $G''_f \mapsto G_t$, where $G''_f = G_f \setminus G'_f$, to yield netlist $N_1$.
- Because $N_i$ trace-contains $N$ as per Theorem 5.3, any target proven unassertable in $N_i$ implies a corresponding unassertable target in $N$, hence any results reported in Step 5 are correct.
- If counterexamples were obtained for $N_i$, they may be valid for $N$ or they may be spurious. The algorithm of Figure 5.3 is used to differentiate these cases in Step 6. The result of this algorithm is a set of valid counterexamples for $N$ and / or a refined netlist $N_{i+1}$.
- By Theorem 5.4, any counterexamples reported in Step 7 will be valid.
- By Theorem 5.4, if any targets remain unsolved, refined netlist $N_{i+1}$ will not exhibit a spurious assertion against traces $P_i$. Convergence of the refinement loop is guaranteed noting that netlists are finite as per Definition 2.7, hence $|G''_f|$ is finite and *each* refinement iteratively eliminates one or more elements from $G''_f$.

□

## 5.5 Optimality of Reductions

Theorem 5.4 ensures the correctness and convergence of the overall abstraction-refinement procedure. However, there are several points to consider regarding the *optimality* of the resulting refined netlist, which may have a significant impact on the resources required to compute counterexamples, as well as to prove targets unassertable.

1. While the SAT solution obtained from Step 6 of the algorithm of Figure 5.3 identifies an adequate set of causal merges for refinement, it does not directly attempt to obtain a solution with a *minimal* set of causal merges, as would be necessary for optimality of the refined netlist.

2. Compatibility issues with don't-care enabled merging entail that two sets of merges may be independently but not jointly valid or vice-versa [MB05, ZKKSV06]. The selection of causal merges may thus entail cumulative sub-optimalities across refinement iterations, even if each individual iteration is optimal.

Regarding the first issue: for optimal reductions, one would wish to eliminate as few merges as possible during each refinement. A precise solution to this problem may be attained by solving a max-sat problem [LS05] in Step 6 of the algorithm of Figure 5.3, constructed from the original SAT instance augmented with an additional clause for each conditional merge construct representing the selection of the merged value. For enhanced runtime, we have found that a near-optimal initial bound to the max-sat solution may be obtained using a standard SAT solver augmented with a decision procedure which heuristically assigns 1 to PRIMARY INPUTS before assigning 0. Recall that, due to enforcing the input sequence of $p'_i$, PRIMARY INPUTS correlating to the resulting SAT instance will occur within conditional merge instances.

Regarding the second issue: to circumvent the risk that local choices at each refinement iteration will entail global suboptimality across multiple iterations, it

is necessary to re-compute refinement iterations relative to the initial maximally-merged abstraction $N_1$. This process is illustrated by the *optimal abstraction-refinement* mode of the algorithm of Figure 5.4, where at a particular refinement iteration, *all* prior counterexamples $\bigcup_{j\in\{1,\ldots,i\}} P_j$ are used to refine relative to $N_1$, instead of merely using the final set $P_i$ obtained upon the prior refined netlist $N_i$. The following theorems demonstrate the correctness and optimality of this flavor of the abstraction-refinement process.

**Theorem 5.6.** Assuming that a max-sat procedure is used in the refinement algorithm of Figure 5.3, the algorithm of Figure 5.4 (run in *optimal abstraction-refinement* mode) will converge upon a correct verification result for the original netlist $N$.

*Proof.* The correctness of verification results follows from the proof of Theorem 5.5. To guarantee convergence, we note that we cannot have two identical refined netlists $N_i = N_j$ for $i > j$. This follows from the observation that $N_i$ is formed by refining against *all* prior traces, including $P_j$ which comprises a spurious counterexample against $N_j$. By Theorem 5.4, we therefore cannot have $N_i = N_j$. Additionally, since netlists and hence $G''_f$ are finite, there are a finite number of possible distinct refined netlists. □

**Theorem 5.7.** Assuming that a max-sat procedure is used in the refinement algorithm of Figure 5.3, the algorithm of Figure 5.4 (run in *optimal abstraction-refinement* mode) will at every iteration yield an optimal refined netlist which retains as many of the desired merges as possible while not exhibiting a spurious assertion against any counterexample obtained prior to that iteration.

*Proof.* This proof follows trivially by Theorem 5.6 and by the construction of the max-sat formulation. □

One additional point regarding optimality: the validity of don't-care enabled merges is generally neither symmetric nor transitive [ZKKSV06]. Thus, the selection of desired merges from the computed equivalence classes in Step 1 of the algorithm of Figure 5.4 may impact the size of the refined netlist. Clearly optimality

would be achieved if the algorithm of Figure 5.4 were run upon every permutation of desired merges consistent with the computed equivalence classes, and selecting the result which retains the most merges. However, doing so would be computationally intractable, and likely run contrary to the use of the abstraction-refinement framework to reduce overall verification resources. To partially offset this computational expense, we note that in reaction to a spurious counterexample, while the *causal* merges must be eliminated, we may attempt to introduce *alternate* merges from within the equivalence classes in their place. Such a framework may be used to generate a refinement retaining an optimal number of merges, provided that the procedure exhaustively attempt alternate merges before outright discarding any set of causal merges. Convergence of such an alternate-merge introduction framework may be guaranteed simply by ensuring that no refinement $N_{i+j}$ repeat the same set of merges reflected in an earlier refinement $N_i$. While computationally superior to the naïve approach of exhaustive enumeration, since alternative merges need only be explored *on demand*, this approach is also prone to be computationally intractable in practice due to requiring the exploration of all cross-products of permissible merges for the identified causal merges. We thus introduce an efficient technique to generate a nearly-optimal set of desired merges from the equivalence classes in Section 5.5.1.

### 5.5.1 Incremental Elimination of Constraint-Weakening Merges

SAT-based analysis is often used to search for counterexamples, iteratively checking for failures at time 0, 1, 2, ... until some computational resource limitation is exceeded. For efficiency, it is desirable to leverage *incremental SAT* [Hoo93] in this process, first creating a SAT instance to check for a failure at time 0, then unfolding an additional time-frame onto the existing SAT instance to check for a failure at time 1, etc. Such incrementality enables the reuse of learned clauses from earlier time-frames to speed up the SAT solution for later time-frames [Sht01].

In an abstraction-refinement framework, incrementality is more difficult to achieve such that the same SAT instance may be utilized to find counterexamples

*across* refinements. However, if utilizing the standard abstraction-refinement mode of the algorithm of Figure 5.4, and utilizing the *conditional merge* construct instead of outright performing the desired merges, incrementality may be achieved by merely constraining the causal conditional merge selectors to 0. Such constraints effectively eliminate the causal merges within the SAT instance and allow additional counterexamples to be identified therein. Practically, we have found that devoting a small amount of computational resources to such an incremental SAT-based procedure, and using the resulting counterexamples to jump-start the optimal abstraction-refinement process, tends to substantially reduce overall resources to arrive at a refined abstraction which is not prone to spurious counterexamples.

Furthermore, one mechanism that we have found to quickly converge upon a nearly-optimal set of compatible merges from within the equivalence classes (Step 1 of the algorithm of Figure 5.4) is to utilize the preprocessing mechanism discussed in the prior paragraph with a variant of the conditional merge which enables the selection among *all* gates within an equivalence class. As spurious counterexamples are obtained, we may disable the causal merges and preserve selection among the rest. The desired merges may then be chosen from those which remain at the termination of this preprocessing step, heuristically helping to ensure near-optimal compatibility.

## 5.6 Experimental Results

In this section, we provide experiments to illustrate the redundancy removal enhancements enabled by our techniques. All experiments were run on a 2.1GHz POWER5 processor, using the IBM internal transformation-based verification tool *SixthSense* [MBP+04]. We applied our techniques on a variety of designs, including a subset of the IBM FV Benchmarks [IBM] and four difficult industrial testbenches. FXU is a testbench used to verify the control path of a fixed-point unit. FPU is a floating-point unit. SCNTL is a testbench used to verify the control of an instruction-dispatch unit. IBUFF is an instruction buffer. Each of these examples has constraints which entail dead-end states.

| Benchmark | Design Info | | Constraint-Safe Merging [MBA05] | | | Constraint-Enhanced Merging | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | Gates | Targets | Gates Merged | Unsolved Targets | Resources (s; MB) | Gates Merged | Refined Merges | Improvement | Unsolved Targets | Resources (s; MB) |
| FXU | 32903 | 8 | 2218 | 0 | 450; 146 | 2482 | 64 | 9.0% | 0 | 400; 195 |
| FPU | 115037 | 1 | 2022 | 1 | 5465; 690 | 4928 | 0 | 143.7% | 0 | 1140; 384 |
| SCNTL | 51504 | 551 | 6638 | 24 | 342; 133 | 6962 | 17 | 4.5% | 0 | 1103; 383 |
| IBUFF | 19230 | 303 | 222 | 14 | 77; 91 | 831 | 20 | 260.8% | 0 | 144; 160 |
| IBM_FV_11 | 4799 | 1 | 228 | 0 | 16; 64 | 747 | 4 | 225.9% | 0 | 37; 69 |
| IBM_FV_24 | 13391 | 1 | 313 | 0 | 70; 119 | 793 | 7 | 151.1% | 0 | 83; 137 |

Table 5.1: Sequential redundancy removal results

The enhanced redundancy removal of our approach is illustrated in Table 5.1. The first 3 columns indicate the name of the benchmark, the size of the original netlist and the number of targets in the original netlist. We first compare the redundancy removal possible using prior techniques [MBA05] with that of our approach (Columns 4 vs. 7), and then illustrate the impact of this additional reduction on the verification process by using $k$-induction [ES03] to attempt to verify the targets on the optimized designs (Columns 5 vs. 10). The resources reported in Columns 6 vs. 11 refer to the combined process of redundancy removal and induction. The induction process was limited to 30 seconds and $k \leq 10$ to avoid significant skew of runtime for cases where targets were left unsolved.

As illustrated in Table 5.1, our approach identifies significantly more redundancy resulting in an increased number of non-refined merges, often more than $2\times$ (indicated by a number greater than $100\%$ in Column 9). The average increase in non-refined merges across all of these designs is $132.5\%$. Our techniques also enable the solution of more targets, some of which we were unable to find a successful proof strategy for otherwise. We report the number of merges which were refined during the algorithm of Figure 5.4 in Column 8 (which is a subset of Column 7), which is only a small percentage of the additional merges enabled by our techniques. Our approach in cases entails moderate additional run-time due to the larger set of equivalence candidates during the redundancy identification process, though in other cases, such as the FPU, this results in significantly lesser runtime.

## 5.7 Related Work

The work of [MBA05] discusses redundancy removal in the presence of constraints, allowing gate merges in the fanin of a constraint provided that the proof of the corresponding gate equivalence does not require the state-pruning power of that constraint. Our approach lifts this restriction, enabling a greater degree of simplification. It is noteworthy that the other constraint-preserving netlist transformations proposed in [MBA05] may synergistically enhance, and be enhanced by, our constraint-based redundancy removal.

In [LC06], the authors propose to enhance an inductive SAT solver by performing *all* merges enabled by the constraints representing the induction hypothesis. To avoid the weakening of constraints entailed by such merging, they add additional constraints over the fanin of the merged gates within the SAT solver. However, unlike our approach, they do not address constraint-enhanced reduction of sequential netlists; theirs is effectively a run-time optimization to the inductive SAT solver, which our approach may complementarily use if relying upon induction in Step 4 of the algorithm of Figure 3.1.

There are similarities between our approach and those that optimize relative to other types of don't cares. However, ours is among the first to address a framework for efficient exploitation of don't cares across *sequential* logic boundaries. Additionally, our techniques efficiently allow concurrent global identification and reduction of all redundancy candidates, unlike ODC-based techniques which often probe for merge candidates one by one due to compatibility concerns (e.g., [ZKKSV06]), and for efficiency often trade optimality for computational efficiency using local analysis (e.g., [ZKKSV06, MB05]). Overall, there are numerous complementary distinctions between our constraint-based optimization and frameworks for exploiting other flavors of don't cares, and we feel that there is promise in exploring the use of our methods to extend other don't-care based simplification methods to the sequential domain.

# Chapter 6

# Exploiting Constraints in Transformation-Based Verification

## 6.1 Overview

In Chapter 5, we studied how a scalable *assume-then-prove* sequential redundancy identification framework [vE98, BC00, HCC+00, MBPK05] can be extended to optimally leverage constraints. However, to ensure the scalability and applicability sequential redundancy identification frameworks, we also need to address the applicability of various transformation and verification engines in a TBV framework in the presence of constraints. In particular, we need to study how transformation and verification algorithms can optimally leverage constraints to enhance verification process and at the same time avoid violating constraint semantics.

To illustrate the challenges in applying transformation algorithms in the presence of constraints, consider the netlist depicted in Figure 6.1. Constraint *c* disallows precisely the input sequences that can evaluate *t* to 1. If j $>$ i, then *t* can evaluate to 1 as the constraint precludes such paths only at a later time-step. If on the other hand j $\leq$ i, constraint *c* prevents *t* from ever evaluating to 1. This demonstrates that temporal abstractions like retiming [KB01], which may effectively alter the values of $i$ and $j$, must take precautions to ensure that constraint semantics are preserved through their transformations.

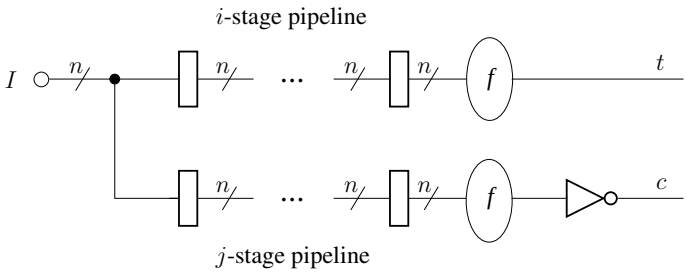



Figure 6.1: Sequential constraint example

Additionally, we study constraint-preserving testcase generation for simulation. This is a widely researched topic, e.g., [YSP+99, YAAP03]; however, the prior solutions do not address preservation of *dead-end constraints* which entail states for which there is no legal input stimulus. Dead-end constraints tend to reduce the efficiency of explicit-state analysis, as well as semi-formal search; when a dead-end state is reached, the only recourse is to backtrack to an earlier state. Though dead-end constraints are considered *user errors* in certain methodologies [YSP+99], they are specifiable in a variety of languages, and in cases are powerful constructs for modeling verification tasks and case-splitting strategies [JG94].

In this chapter, we discuss how various existing automated transformation algorithms may be optimally applied in a property-preserving manner to designs with constraints. We also introduce an efficient input stimulus generation algorithm that is robust against dead-end states as well as fully-automated techniques for eliminating, introducing, and simplifying constraints in a property-preserving manner.

## 6.2 Constraint-Preserving Simulation

Constraint preserving simulation is critical not only for the efficiency of sequential redundancy identification frameworks, but also plays an important role in enabling successful application of verification algorithms such as semi-formal search. In a sequential redundancy identification framework, random simulation along with

semi-formal analysis plays a vital role in computing initial redundancy candidates. By comparing valuations of the gates within simulation-generated traces, many inequivalences may be identified with approximately linear runtime, which often substantially reduces the number of unsuccessful proof attempts in the redundancy identification process [vE98, Kue04].

Constraint-preserving input stimuli generation has been widely researched by the simulation community, e.g., in [YSP+99, YAAP03]. However, the vast majority of this work is only applicable to netlists without dead-end states. While some propose that testbench authors should avoid the use of constraints which entail dead-end states [YSP+99], such constraints may readily be expressed in various specification languages. In practice, we have often seen such constraints in use due to the manual effort involved in attempting to map all constraints directly to PRIMARY INPUTs; e.g., to convert the single-predicate constraint that illegal Decode-stage instructions imply invalidity in Figure 5.2 into a complex constraint over instructions entering the Fetch stage. When a dead-end state is encountered during simulation, the simulation process must halt and begin anew from a shallower state. While the overhead of checkpointing and restoring states for simulation is somewhat undesirable, a more significant problem is that dead-end states may preclude the ability of random simulation to reach deeper states in the design.

The inability to reach deeper states in the presence of dead-end states is a major drawback for both redundancy identification and semi-formal search. In a sequential redundancy identification framework, this may entail highly inaccurate initial equivalence classes for gates which may only be differentiated by deep traces. Highly inaccurate initial set of equivalence classes can cause major slowdowns in the application of resource intensive algorithms to prove redundancy candidates. The main aim of semi-formal search is to expose design flaws which are too deep to reach with purely formal search. In semi-formal search, it is the job of simulation to take the design to a state closer to the assertion of a target and thereby enhance the ability of formal algorithms to expose the fail. Dead-end states can prevent simulation from reaching deeper states and hence reduce efficiency of semi-formal search.

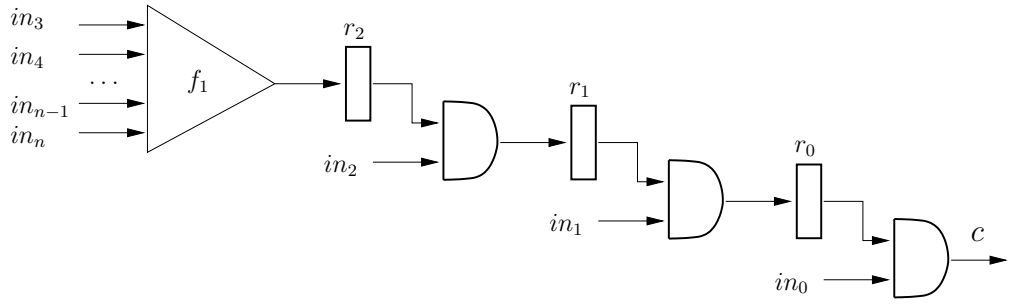



Figure 6.2: Sequential constraint example

```
for (i = 0; i < |C|; i++) {
   K (c_i) = sliding_window_depth (c_i);
}
for (cycle = 0; cycle < num_sim_cycles; cycle++) {
   S_current = State of Netlist N at time cycle
   simulation_patterns = satisfiability_check (N, S_current, C, K);
   Perform Random Simulation of Netlist N using simulation_patterns;
   if dead-end state reached, break;
}
```

Figure 6.3: Simulation Using Sliding Window Algorithm

To illustrate the challenge of stimulus generation in the presence of sequentially-driven constraints, consider the example of Figure 6.2, where gate $c$ is a constraint and $r_0, \ldots, r_2$ are REGISTERS. The stimulus generator must assign values to PRIMARY INPUTS $in_0, \ldots, in_n$ which prevent $c$ from evaluating to 0 for the desired duration of the simulation run. While the stimulus generator could assign $0$ to $in_i$ (for $i \in \{0, 1, 2\}$) at any time-frame $j$, doing so would result in violating $c$ at time $j + i$. The stimulus generator thus must perform assignments at time $j$ which preclude constraint violations at later time-frames.

Our solution for stimulus generation is based upon $k$-step satisfiability solving. At each time-frame $j$, we cast a satisfiability check from the current state of the netlist to identify a set of PRIMARY INPUT valuations at time $j$ which may be

extended to satisfy each constraint for times $j$ through $j+k$ using a *sliding window* approach. The algorithm to perform simulation using sliding window algorithm is illustrated in Figure 6.3. For scalability, it is important to keep $k$ as small as possible because satisfiability checking is an NP-complete problem [Coo71]. Our solution, as follows, thus performs design-specific analysis to determine a minimal value of $k$ which avoids dead-end states.

**Definition 6.1.** Given a simple directed path in $\langle V, E \rangle$, the *delay* of that path is the number of gates of type REGISTER. The *minimum input delay* of a gate $g$ is defined as the minimum delay along any simple path from any PRIMARY INPUT to $g$. The *maximum input delay* of a gate $g$ is the minimum delay relative to the deepest PRIMARY INPUT: the maximum value among any PRIMARY INPUT for the minimum delay along any simple path from that PRIMARY INPUT to $g$.

Experimentally, we have found that setting $k$ to the *maximum input delay* of a corresponding constraint gate is adequate to prevent simulation from encountering dead-end states. The use of maximum input depth for $k$ is intuitive; it represents the minimum delay at which some *furthest* PRIMARY INPUT may affect the evaluation of a given constraint. Depending upon the nature of the design, a PRIMARY INPUT may affect the constraint much later than the input delay, e.g., if the design has a counter. However, we have not encountered any designs for which this depth is inadequate, given a large set of industrial examples.

A related concern is whether this $k$ is larger than necessary, since the performance of the satisfiability solver is highly sensitive to this value. For example, if the AND gates in Figure 6.2 are converted to OR gates, the *minimum input delay* of 0 is adequate to avoid dead-end states by assigning $in_0$ to 1. In practice, the computation of minimum adequate window depth is prohibitively expensive, requiring a sequence of quantified Boolean formulas checking whether for every state, there exists an input sequence of a particular depth which satisfies the constraint. Our practical solution to this problem is to break the overall simulation run into a number of phases. We use a $\log_2$ search procedure within a minimum and maximum range (initialized as the *minimum* and *maximum input delay*, respectively) to identify a minimal yet adequate sliding window depth to enable the desired simulation

```
for (i = 0; i < |C|; i++) {
   if (initial_phase) {
      min_range (c_i) = minimum input delay (c_i);
      max_range (c_i) = maximum input delay (c_i);
    }
   K (c_i) = min_range (c_i);
}
Sim: Simulate using sliding window algorithm described in Figure 6.3
if (dead-end state reached) {
   for (i = 0; i < |C|; i++) {
      if (c_i evaluates to 0 during simulation) {
            if (K (c_i) == max_range (c_i)) {
                  min_range (c_i) = max_range (c_i) + 1;
                  max_range (c_i) = 2×max_range (c_i);
                  K (c_i) = min_range (c_i);
             } else {
                  min_range (c_i) = K (c_i) + 1;
                  K (c_i) = ⌈(min_range (c_i) + max_range (c_i)) / 2⌉;
             }
       }
   }
   Goto Sim:
} else {
   max_range (c_i) = K (c_i); Goto next phase:
}
```

Figure 6.4: Algorithm for choosing sliding window depth in each phase

run length as illustrated in Figure 6.4. At each phase, we generate stimuli using the minimum range value as the window depth. If that fails (i.e., encounters a dead-end state), we update the minimum range to the unsuccessful value plus one and repeat the phase using a median depth between the minimum and maximum range. If successful, we update the maximum range to the successful value and move on to the next phase. Since the *maximum input delay* may be exponentially lesser than the adequate value, if the maximum range is found inadequate at any phase, we double it for the next iteration. This range analysis is performed on a per-constraint basis for optimality; it is straightforward to identify *which* constraints were satisfied vs.

| Benchmark | Design Info | SimGen [YSP+99] | | Sliding Window | | | SAT-only | |
|---|---|---|---|---|---|---|---|---|
| | Gates | Valid Time-Steps | Time (s) | Input Depth: Min; Max; Algo | Valid Time-Steps | Time (s) | Valid Time-Steps | Time (s) |
| FXU | 32903 | 1 | 0.13 | 1; 2; 1 | 1000 | 2.6 | 165 | 1800 |
| FPU | 115037 | 5 | 0.39 | 5; 14; 10 | 1000 | 228 | 902 | 1800 |
| SCNTL | 51504 | 53 | 2.94 | 7; 14; 8 | 1000 | 87 | 72 | 1800 |
| IBUFF | 19230 | 57 | 1.05 | 5; 13; 6 | 1000 | 7.4 | 134 | 1800 |
| IBM_FV_11 | 4799 | 2 | 0.17 | 4; 8: 6 | 1000 | 2.6 | 337 | 1800 |
| IBM_FV_24 | 13391 | 2 | 0.59 | 4; 19; 4 | 1000 | 3.2 | 252 | 1800 |

Table 6.1: Constraint-preserving simulation results

violated within each phase.

To enable highest state coverage during simulation-based analysis, it is often desirable to avoid frequent repetition of PRIMARY INPUT valuations, instead attempting to generate a good distribution across the constraint-satisfying valuations [KK07]. We use a circuit SAT solver [KPKG02] which performs minimal assignments in its solutions; PRIMARY INPUTs that are unassigned in the SAT solution may be randomized. CNF SAT solvers may also be manipulated to obtain minimally-assigned solutions, e.g., as in [RS04]. To provide further randomness to the simulation process, e.g., to prevent the SAT solver from frequently yielding the same valuation to a given set of PRIMARY INPUTs, one may add clauses to force the SAT solver to generate a solution which differs in its selection of PRIMARY INPUT assignments. If the result is unsatisfiable, then a previously-generated result may be randomly selected.

The experiments of Table 6.1 illustrate the power of our sliding window algorithm for constraint-preserving random simulation. The benchmarks include a subset of the IBM FV Benchmarks [IBM] and four difficult industrial testbenches. FXU is a testbench used to verify the control path of a fixed-point unit. FPU is a floating-point unit. SCNTL is a testbench used to verify the control of an instruction-dispatch unit. IBUFF is an instruction buffer. Each of these examples has constraints which entail dead-end states.

The first two columns indicate the name of the benchmark and the size of the original netlist. We compare the results of SimGen [YSP+99] (Columns 3-4) which performs purely combinational constraint solving; our sliding window approach (Columns 5-7); and pure formal analysis using SAT to solve the constraints

for the entire duration of the simulation run, using random stimulus generation for unassigned PRIMARY INPUTS (Columns 8-9). Our goal is to simulate the designs without encountering dead-end states for $1000$ time-steps, within a time-limit of $1800$ seconds. While fast, SimGen [YSP+99] results in constraint violations within $57$ time-steps for each design, and often substantially lesser. The SAT-only approach times-out for every design, often completing substantially lesser than 1000 time-steps. In contrast, our sliding window approach is able to complete the desired simulation for every design. The sliding window depth used varies across the designs; in Column 5, we report the minimum and maximum input depth, followed by the depth algorithmically converged upon by our $\log_2$ range analysis.

### 6.2.1 Related Work

Constraint-satisfying stimulus generation has been extensively studied, though the vast majority of this work does not address dead-end states, e.g., [YSP+99, YAAP03]. The approach of [CDH+05] is one of the few that addresses dead-end states by using a BDD-based framework to manipulate a synthesized constraint automaton to avoid dead-end states before they are reached. While demonstrated to be effective, this approach is only applicable if the constraints are specified using temporal logic to reason solely about PRIMARY INPUTS. If arbitrary design signals are referenced by the constraints, as in our practical experience they frequently are, their approach resorts to underapproximation, precluding the exploration of arbitrary reachable states. Our techniques do not suffer this limitation.

## 6.3 Retiming

Retiming is a synthesis optimization technique capable of reducing the number of REGISTERS of a netlist by relocating them across combinational gates [LS91].

**Definition 6.2.** A *retiming* of netlist $N$ is a gate labeling $r : V \mapsto \mathbb{Z}$, where $r(v)$ is the *lag* of gate $v$, denoting the number of REGISTERS that are moved backward

through $v$. A *normalized retiming* $r'$ may be obtained from an arbitrary retiming $r$, and is defined as $r' = r - \max_{v \in V} r(v)$.

In [KB01], normalized retiming is proposed for enhanced invariant checking. The retimed netlist $\tilde{N}$ has two components: **(1)** a sequential *recurrence structure* $\tilde{N}'$ which has a unique representative for each combinational gate in the original netlist $N$, and whose REGISTERs are placed according to Definition 6.2, and **(2)** a combinational *retiming stump* $\tilde{N}''$ obtained through unfolding, representing retimed initial values as well as the functions of combinational gates for prefix time-steps that were effectively discarded from the recurrence structure. It is demonstrated in [KB01] that each gate $\tilde{u}'$ within $\tilde{N}'$ is trace-equivalent to the corresponding $u$ within $N$, modulo a temporal skew of $-r(u)$ time-steps. Furthermore, there will be $-r(u)$ correspondents to this $u$ within $\tilde{N}''$, each being trace-equivalent to $u$ for one time-step during this temporal skew. Property checking of target $t$ is thus performed in two stages: a bounded check of the time-frames of $t$ occurring within the unfolded retiming stump, and a fixed-point check of $\tilde{t}'$ in the recurrence structure. If a trace is obtained over $\tilde{N}'$, it may be mapped to a corresponding trace in $N$ by reversing the $\langle$*gate, time*$\rangle$ relation inherent in the retiming.

**Theorem 6.1.** Consider a normalized retiming where every target and constraint gate is lagged by the same value $-i$. Property checking will be preserved provided that:

1. the $i$-step bounded analysis of the retiming stump enforces all constraints across all time-frames, and

2. every retimed constraint gate, as well as every unfolded time-frame of a constraint referenced in a retimed initial value in $\tilde{N}'$, is treated as a constraint when verifying the recurrence structure.

*Proof.* Correctness of (1) follows trivially by the construction of the bounded analysis. Correctness of (2) follows from the observation that: **(a)** every gate lagged by $-i$ time-steps (including all targets and constraints) is trace-equivalent to the corresponding original gate modulo a skew of $i$ time-steps, and **(b)** the trace pruning

caused by constraint violations within the retiming stump is propagated into the recurrence structure by re-application of the unfolded constraint gates referenced in the retimed initial values. □

The min-area retiming problem may be cast as a minimum-cost flow problem [LS91, HMB07]. One may efficiently model the restriction of Theorem 6.1 by *renaming* the target and constraint gates to a single vertex in the retiming graph, which inherits all fanin and fanout edges of the original gates. This modeling forces the retiming algorithm to yield an optimal solution under the equivalent-lag restriction. While this restriction may clearly impact the optimality of the solution, it is generally necessary for property preservation.

## 6.4 Structural Target Enlargement

Target enlargement [BKA02] is a technique to render a target $t'$ which may be asserted at a shallower depth from the initial states of a netlist, and with a higher probability, than the original target $t$. Target enlargement uses preimage computation to calculate the set of states which may assert target $t$ within $k$ time-steps. A transition-function vs. a transition-relation based preimage approach may be used for greater scalability. Inductive simplification may be performed upon the $k$-th preimage to eliminate states which assert $t$ in fewer than $k$ time-steps. The resulting set of states may be synthesized as the enlarged target $t'$. If $t'$ is unreachable, then $t$ must also be unreachable. If $t'$ is asserted in trace $p'$, a corresponding trace $p$ asserting $t$ may be obtained by casting a $k$-step bounded search from the state asserting $t'$ in $p'$ which is satisfiable by construction, and concatenating the result onto $p'$ to form $p$. The modification of traditional target enlargement necessary in the presence of constraints is depicted in Figure 6.5.

**Theorem 6.2.** The target enlargement algorithm of Figure 6.5 preserves property checking.

*Proof.* The constraint-preserving bounded analysis used during the target enlargement process will generate a valid trace, or guarantee that the target cannot be as-

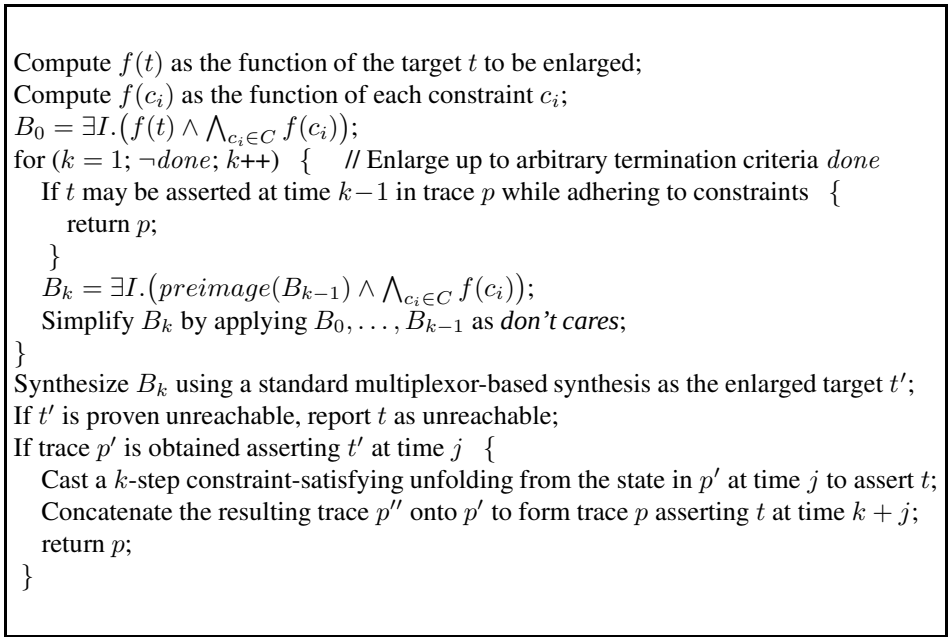



Figure 6.5: Target enlargement algorithm

serted at times $0, \ldots, k-1$, by construction. To ensure that the set of enlarged target states may reach the original target along a trace which does not violate constraints, the constraint functions are conjuncted onto each preimage prior to input quantification. The correctness of *target unreachable* results, as well as the trace lifting process, relies upon the fact that there exists an $k$-step extension of any trace asserting $t'$ which asserts $t$ as established in [BKA02], here extended to support constraints. □

There is a noteworthy relation between retiming a target $t$ by $-k$ and performing a $k$-step target enlargement of $t$; namely, both approaches yield an abstracted target which may be asserted $k$ time-steps shallower than the corresponding original target. Recall that with retiming, we retimed the constraints in lock-step with the targets. With target enlargement, however, we retain the constraints intact. There is one fundamental reason for this distinction: target enlargement yields sets of states which only preserve the *assertability* of targets, whereas retiming more

tightly preserves trace equivalence modulo a time skew. This relative weakness of property preservation with target enlargement is due to its input quantification and preimage accumulation via the don't cares. If preimages were performed to *enlarge* the constraints, there is a general risk that a trace asserting the enlarged target while preserving the enlarged constraints may not be extendable to a trace asserting the original target, due to possible conflicts among the PRIMARY INPUT valuations between the constraint and target cones in the original netlist. For example, a constraint could evaluate to 0 whenever a PRIMARY INPUT $i_1$ evaluates to 1, and a target could be assertable only several time-steps after $i_1$ evaluates to 1. If we enlarged the constraint and target by one time-step, we would lose the unreachability of the target under the constraint because we would quantify away the effect of $i_1$ upon the constraint.

## 6.5 Structural Reparameterization

Reparameterization techniques, as described in Chapter 4, operate by identifying a cut of a netlist graph $V_C$, enumerating the valuations sensitizable to that cut (its *range*), then synthesizing the range relation and replacing the fanin-side of the cut by this new logic. In order to guarantee property preservation, one must generally guarantee that target and constraint gates lie on the cut or its fanout. Given parametric variables $p^i$ for each cut gate $V_C^i$, the range is computable as $\exists I. \bigwedge_{i=1}^{|V_C|} \left(p^i \equiv f(V_C^i)\right)$. If any cut gate is a constraint, its parametric variable may be forced to evaluate to 1 in the range to ensure that the synthesized replacement logic inherently reflects the constrained PRIMARY INPUT behavior. This cut gate will then become a constant ONE in the abstracted netlist, effectively being discarded.

While adequate for combinationally driven constraints and a subset of sequentially driven constraints, this straight-forward approach does not address the preservation of dead-end states. A postprocessing approach is thus necessary to identify those abstracted constraints which have dead-end states, and to re-apply the dead-end states as constraints in the abstracted netlist. Given a constraint gate

$c_i$ that is used to constrain the range, the dead-end states represented by the constraint can be computed by evaluating $\exists I.f(c_i)$. If not a tautology, the result of the existential quantification represents dead-end states for which no PRIMARY INPUT valuations are possible, hence a straight-forward multiplexor-based synthesis of the result may be used to create a logic cone to be tagged as a constraint in the abstracted netlist.

To illustrate the importance of re-applying dead-end constraints during reparameterization, consider a constraint of the form $i_1 \wedge r_1$ for PRIMARY INPUT $i_1$ and REGISTER $r_1$. If this constraint is used to restrict the range of a cut, its replacement gate will become a constant ONE hence the constraint will be effectively discarded in the abstracted netlist. The desired byproduct of this restriction is that $i_1$ will be forced to evaluate to 1 in the function of all cut gates. However, the undesired byproduct is that the abstracted netlist will no longer disallow $r_1$ from evaluating to 0 without the reapplication of the dead-end constraint $\exists i_1.(i_1 \wedge r_1)$ or simply $r_1$. Because this re-application will ensure accurate trace-prefixing in the abstracted netlist, the range may be simplified by applying the dead-end state set as *don't cares* prior to its synthesis as noted in [YAAP03].

If there are multiple constraints in the netlist, computing of dead-end constraints individually with respect to each constraint gate $c_i$ used to constrain the range may give wrong results. For example, consider two constraints $c_1 = i_1 \vee r_1$ and $c_2 = \neg i_i \vee r_1$. If we individually compute dead-end states with respect to each constraint by computing $\exists i_1.(i_1 \vee r_1)$ and $\exists i_1.(\neg i_1 \vee r_1)$, it would appear that there are no dead-end constraints. However, both constraints $c_1$ and $c_2$ need to be satisfied simultaneously when evaluating the netlist and the only way to satisfy both the constraints is by forcing $r_1$ to evaluate to 1. This implies that we need to add $r_1$ as a dead-end constraint to the netlist after reparameterization. Hence, rather than computing the dead-end constraints with respect to each constraint, one must compute dead-end constraint with respect to the conjunction of all the constraints that are used to constrain the range as follows $\exists I. \bigwedge_{i=1}^{|C|} f(c_i)$.

**Theorem 6.3.** Structural reparameterization is property-preserving, provided that any constraints used to restrict the computed range are re-applied as simplified

dead-end constraints in the abstracted netlist.

*Proof.* The correctness of reparameterization without dead-end constraints follows from prior work, e.g., [BM05]. Note that reparameterization may replace any constraints by constant ONE in the abstracted netlist. Without the re-application of the dead-end states as a constraint, the abstracted netlist will thus be prone to allowing target assertions beyond the dead-end states. The re-application of the dead-end states as a constraint closes this semantic gap, enabling property-preservation. □

## 6.6 Phase Abstraction

*Phase abstraction* [BHSA03] is a technique for transforming a *latch-based* netlist to a REGISTER-based one. A latch is a gate with two inputs (*data* and *clock*), which acts as a buffer when its *clock* is active and holds its last-sampled *data* value (or initial value) otherwise. Topologically, a $k$-phase netlist may be $k$-colored such that latches of color $i$ may only combinationally fan out to latches of color $\big((i+1) \bmod k\big)$; a combinational gate acquires the color of the latches in its combinational fanin. A modulo-$k$ counter is used to clock the latches of color $(j \bmod k)$ at time $j$. As such, the initial values of only the $(k–1)$ colored latches propagate into other latches. Phase abstraction converts one color of latches into REGISTERS, and the others into buffers, thereby reducing state element count and temporally *folding* traces modulo-$k$, which otherwise stutter.

Phase abstraction may not preserve property checking for netlists with constraints as illustrated by the following example. Assume that we have a 2-phase netlist with a target gate of color 1, and a constraint gate of color 0 which is unconditionally violated one time-step after the target evaluates to 1. Without phase abstraction, the target may be unassertable since the constraint prefixes the trace only on the time-step after the target evaluates to 1. However, if we eliminate the color-0 latches via phase abstraction, the constraint becomes violated concurrently with the target's evaluation to 1, hence the target becomes unassertable. Nonetheless, there are certain conditions under which phase abstraction preserves property checking as per the following theorem.

**Theorem 6.4.** If each constraint and target gate is of the same color, phase abstraction preserves property checking.

*Proof.* The correctness of phase abstraction without constraints has been established in prior work, e.g., [BHSA03]. Because every constraint and target gate are of the same color $i$, they update concurrently at times $j$ for which $\big((j \bmod k) = i\big)$. Phase abstraction will merely eliminate the stuttering at intermediate time-steps, but not temporally skew the updating of the constraints relative to the targets. Therefore, the trace prefixing of the constraints remains property-preserving under phase abstraction. □

Automatic approaches of attempting to establish the criteria of Theorem 6.4, e.g., via *padding* pipelined latch stages to the constraints to align them with the color of the targets, are not guaranteed to preserve property checking. The problem is that such approaches unconditionally delay the trace prefixing of the constraints, hence even a contradictory constraint which can never be satisfied at time zero – which thus renders all targets unassertable – may become contradictory only at some future time-step in the range $1, \ldots, (k-2)$. After phase abstraction, this delay will be either zero or one time-step; in the latter case, we have opened a hole during which phase abstracted targets may be asserted, even if they are truly unassertable in the original netlist. Nonetheless, in most practical cases, one may methodologically specify their desired verification problem in a way that adheres to the criteria of Theorem 6.4.

## 6.7 $C$-Slow Abstraction

$C$-slow abstraction [BTA$^+$00] is a state folding technique which is related to phase abstraction, though is directly applicable to REGISTER-based netlists. A $c$-slow [LS91] netlist has REGISTERs which may be $c$-colored such that REGISTERs of color $i$ may only combinationally fan out to REGISTERs of color $\big((i+1) \bmod c\big)$; a combinational gate acquires the color of the REGISTERs in its combinational fanin. Unlike $k$-phase netlists, the REGISTERs in a $c$-slow netlist update every time-step hence

generally never stutter. Additionally, the initial value of every REGISTER may propagate to other REGISTERs. $C$-slow abstraction operates by transforming all but a single color of REGISTERs into buffers, thereby reducing REGISTER count and temporally *folding* traces modulo-$c$. To account for the initial values which would otherwise be lost by this transformation, an unfolding approach is used to inject the retained REGISTERs into all states reachable in time-frames $0, \ldots, (c{-}1)$.

As with phase abstraction, if the target and constraint gates are of differing colors, this abstraction risks converting some assertable targets to unassertable due to its temporal collapsing of REGISTER stages. Additionally, even the criteria of requiring all target and constraint gates to be of the same color as with Theorem 6.4 is not guaranteed to preserve property checking with $c$-slow abstraction. The problem is due to the fact that $c$-slow netlists do not stutter $\bmod c$. Instead, each time-step of the abstracted netlist correlates to $c$ time-steps of the original netlist, with time-steps $i, c+i, 2{\cdot}c+i, \ldots$ being evaluated for each $i < c$ *in parallel* due to the initial value accumulation. Reasoning across $\bmod c$ time-frames is intrinsically impossible with $c$-slow abstraction; thus, in the abstracted netlist, there is generally no way to detect if a constraint was effectively violated at time $a{\cdot}c+i$ in the original netlist when evaluating a target at time $(a+1){\cdot}c+j$ for $i \neq j$. Even with an equivalent-color restriction, $c$-slow abstraction thus risks becoming *overapproximate* in the presence of constraints. Nonetheless, methodologically, constraints which are not amenable to this state-folding process are of little practical utility in $c$-slow netlists. Therefore, in most cases one may readily map an abstracted counterexample trace to one consistent with the original netlist, e.g., using satisfiability analysis to ensure constraint preservation during intermediate timesteps.

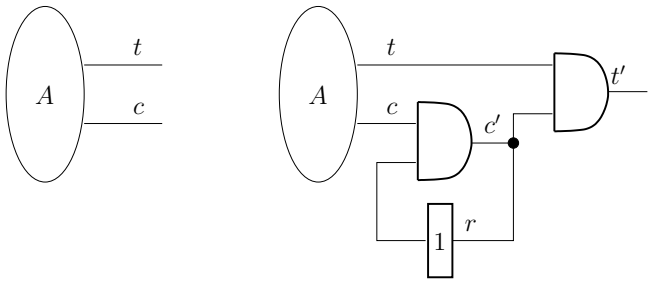
## 6.8 Approximating Transformations

**Overapproximating Transformations.** Various techniques have been developed for attempting to reduce the size of a netlist by overapproximating its behavior. Any target proven unreachable after overapproximation is guaranteed to be unreachable before overapproximation. However, if a target is asserted in the overapproximated

netlist, this may not imply that the corresponding target is assertable in the original netlist. Localization [CGKS02, MA04] is a common overapproximation technique which replaces a set of cut gates of the netlist by PRIMARY INPUTs. The abstracted cut can obviously simulate the behavior of the original cut, though the converse may not be possible.

Overapproximating transformations are directly applicable in the presence of constraints. Overapproximating a constraint cone only weakens its constraining power. For example, while the cone of target $t$ and constraint $c$ may overlap, after localizing the constraint cone it may only comprise localized PRIMARY INPUTs which do not appear within the target cone, thereby losing all of its constraining power on the target. Such constraint weakening is merely a form of overapproximation, which must already be addressed by the overall overapproximate framework. Both counterexample-based [CGKS02] and proof-based [MA04] localization schemes are applicable to netlists with constraints, as they will both attempt to yield a minimally-sized localized netlist such that the retained portion of the constraint and target cones will guarantee unreachability of the targets.

**Underapproximating Transformations.** Various techniques have been developed to reduce the size of a netlist while underapproximating its behavior. For example, unfolding only preserves a time-bounded slice of the netlist's behavior; case splitting (e.g., by merging PRIMARY INPUTs to constants) may restrict the set of traces of a netlist. Underapproximating transformations may safely be applied to a netlist with constraints, as underapproximating a constraint cone only strengthens its power. For example, if a constraint is of the form $i_1 \vee i_2$, underapproximating by merging $i_1$ to constant ZERO will force $i_2$ to constant ONE in the underapproximated netlist even though a target may be asserted in the original netlist only while assigning $i_2$ to a 0. However, this restriction – which may cause unreachable results for targets which were assertable without the underapproximation – must already be addressed by the overall underapproximate framework. Assertions of targets on the underapproximated netlist still imply valid assertions on the original netlist even in the presence of constraints. Extensions to underapproximate frameworks to




a) Netlist $N$; gate $c$ is a constraint, gate $t$ is a target.

b) Netlist $N'$ with no constraints; gate $t'$ is a target.

Eliminate_constraints (Netlist $N$)
$c = 1 \wedge \bigwedge_{v_i \in C} v_i$;
$r$ = **create_register**;
$Z(r) = 1$;
$c' = c \wedge r$;
next_state($r$) = $c'$;
foreach $t \in T$
$t' = t \wedge c'$;
replace each $t \in T$ by corresponding $t'$;
remove all constraints: $C = \emptyset$;

c) Constraint elimination algorithm

Figure 6.6: Property-preserving constraint elimination

enable completeness – e.g., diameter bounding approaches for *complete* unfolding, and complete case splitting strategies – are directly applicable in the presence of constraints.

## 6.9 Constraint Elimination

Given the challenges that they pose to various algorithms, one may wish to eliminate constraints in a property-preserving manner. In Figure 6.6c, we introduce a general constraint elimination algorithm.

**Theorem 6.5.** The constraint elimination algorithm of Figure 6.6c is a *property-preserving* transformation.

*Proof.* Consider any trace $p$ that asserts target $t$ in netlist $N$ at time $i$. Note that netlist $N'$ has the same set of gates as $N$ in addition to gates $c'$, $r$, and $t'$. Consider the trace $p'$ of $N'$ where all common gates have the same valuations over time as in $p$, and gates $c'$, $r$ and $t'$ are evaluated as per Definition 2.10. Because $t$ is asserted at time $i$, $\forall j \leq i.\big(p(c,j) = 1\big)$, and thus by construction of $c'$, $\forall j \leq i.\big(p'(c',j) = 1\big)$. Because $t' = t \wedge c'$, we also have $\forall j \leq i.\big(p(t,j) = p'(t',j)\big)$. It may similarly be proven that for any trace $p'$ that asserts target $t'$ at time $i$, there exists an equivalent trace $p$ that asserts target $t$ at time $i$. □

Performing the constraint elimination transformation in Figure 6.6 enables arbitrary verification and transformation algorithms to be applied to the resulting netlist without risking the violation of constraint semantics. However, this approach could result in significant performance degradation for both types of algorithms:

- Transformation algorithms (particularly redundancy removal) lose their ability to leverage the constraints for optimal simplification of the netlist.
- Falsification algorithms may waste resources analyzing uninteresting states, i.e., from which no target may subsequently be asserted due to $c'$ evaluating to 0.
- The tactical utility of the constraints for case-splitting strategies is lost.

## 6.10 Constraint Introduction

It follows from the discussion of sequential redundancy identification in Chapter 5 that reduction potential may be increased by constraints. It may therefore be desirable to derive constraints that may be introduced into the netlist while preserving property checking, at least temporarily to enhance a particular algorithm.

**Theorem 6.6.** Consider netlist $N$ with gate $g$. If no target in $T$ may be asserted along any trace after gate $g$ evaluates to 0, then $g$ may be labeled as a constraint while preserving property checking.

*Proof.* If gate $g$ is labeled as a constraint, by Definition 2.10, we will only reason about the prefix *length* of traces wherein gate $g$ always evaluates to 1. Since no target in $T$ may be asserted along any trace after gate $g$ evaluates to 0, by Definition 2.20, netlist $N'$ formed from $N$ by labeling gate $g$ as a constraint is property-preserving trace equivalent to $N$. □

Taking the example netlist of Figure 6.6b, any of the gates $c$, $c'$, and $r$ may be labeled as a constraint provided that we may establish the corresponding condition of Theorem 6.6, effectively reversing the transformation of Figure 6.6c. While

this proof may in cases be as difficult as property checking itself, we propose an efficient heuristic algorithm for deriving such constraint candidate gates as follows. Similar to the approach of [BK04], we may localize each of the targets, and use a preimage fixed-point computation to underapproximate the number of time-steps needed to assert that target from a given set of states. Any state not reached during this fixed-point may never reach that target. The intersection of such state sets across all targets represents the conditions from which no target may subsequently be asserted. While the approach of [BK04] proposes only to use this set to steer semi-formal analysis away from useless states, we propose to synthesize the resulting conditions as a constraint in the netlist to enhance reduction potential.

Note that these constraints are in a sense *redundant* because no target asserts may occur after they evaluate to 0 anyway. Therefore, instead of forcing all algorithms to adhere to these constraints which may have an associated overhead, we may treat these as *verification don't cares* so that algorithms may choose to either use these constraints to restrict evaluation of the netlist, or to ignore them. Note that certain verification algorithms, e.g., SAT-based search, may inherently *learn* such conditions and direct their resources accordingly. Ours is a more general paradigm which enables leveraging this information for arbitrary algorithms, particularly to enhance reduction potential.

## 6.11 Constraint Simplification

In this section, we discuss a general approach to simplify constraints. We also discuss an efficient implementation of this paradigm which attempts to replace a constraint with its preimage, heuristically trying to reduce the size of the constraint cone and enable the elimination of that constraint through reparameterization.

We define $prop\_p(t, c)$ as the target gate resulting from applying the constraint elimination algorithm of Figure 6.6c specifically to target $t$ and gate $c$.

**Theorem 6.7.** Consider a netlist $N$ with constraint $c_1$ and target set $T$. Let gate $c_2$ be any arbitrary gate in the netlist. If $\forall t \in T. \big(prop\_p(t, c_1) \equiv prop\_p(t, c_2)\big)$ without

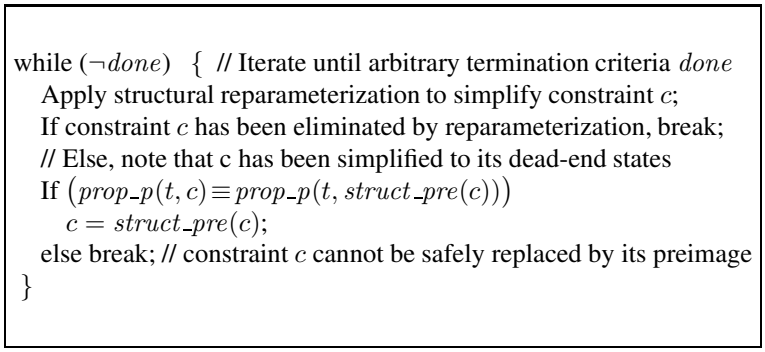



Figure 6.7: Heuristic constraint simplification algorithm

the trace-prefixing of constraint $c_1$, then converting $N$ into $N'$ by labeling $c_2$ as a constraint instead of $c_1$ is a property-preserving transformation.

*Proof.* Since $\forall t \in T.\big(prop\_p(t, c_1) \equiv prop\_p(t, c_2)\big)$ without the trace-prefixing entailed by constraint $c_1$, this proof follows directly from Definition 2.20 and Theorem 6.5. □

Theorem 6.7 illustrates that in certain cases, we may modify the constraint gates in a netlist while preserving property checking. Practically, we wish to exploit this theorem to shrink the size of the constraint cones and thereby effectively strengthen their reduction potential. Note that the structural reparameterization algorithm in Section 6.5 is able to eliminate constraints which have no dead-end states. This is in a sense an optimal transformation, as the constraining power of the constraints are thereafter reflected in the netlist structure itself and effectively filters the input stimulus applied to the netlist. Given these motivations, we present a heuristic constraint simplification algorithm.

**Definition 6.3.** The *structural preimage* of a gate $u$ which has no PRIMARY INPUTS in its combinational fanin, $struct\_pre(u)$, is a logic cone obtained by replacing each REGISTER gate $v \in R$ in the combinational fanin of gate $u$ with its corresponding next-state function.

The algorithm of Figure 6.7 attempts to iteratively simplify, and ultimately

eliminate, the constraints in a property-preserving manner. At each iteration, reparameterization is used to replace the current constraint by its dead-end states. Note that this step will eliminate the constraint if it entails no dead-end states. Otherwise, we attempt to simplify the resulting sequential constraint by replacing it with its structural preimage, using Theorem 6.7 to validate that this replacement preserves property checking. If this check fails (either through refutation or excessive resource requirements), then the algorithm terminates. Otherwise, the algorithm iterates with the resulting simplified constraint.

To illustrate how this algorithm works in practice, consider its application on constraint $c$ in the netlist of Figure 6.1. If $j \leq i$, constraint $c$ can be iteratively replaced by its preimage until it becomes combinational, at which point reparameterization will outright eliminate it. If $j > i$, constraint $c$ can be simplified by shrinking $j$ to $i+1$, at which point the check based upon Theorem 6.7 fails causing the iterations to terminate.

Practically, the equality check of Figure 6.7 tends to be computationally expensive. However, this check can be simplified as per the following theorem.

**Definition 6.4.** The *structural initialization* of a gate $u$ which has no PRIMARY INPUTS in its combinational fanin, $struct\_init(u)$, is a logic cone obtained by replacing each REGISTER gate $v \in R$ in the combinational fanin of gate $u$ with its corresponding initial value function. The initial value constraint of $u$ is defined as $init\_cons(u) = init\_r \vee struct\_init(u)$, where $init\_r$ is a REGISTER whose initial value is ZERO and next-state function is ONE.

**Theorem 6.8.** Consider a netlist $N$ with constraint $c_1$. If $\forall t \in T.\big(prop\_p(t, c_1) \implies prop\_p(t, struct\_pre(c_1))\big)$ in $N$ with the trace-prefixing entailed by constraint $c_1$, then converting $N$ into $N'$ by labeling $struct\_pre(c_1)$ and $init\_cons(c_1)$ as constraints instead of $c_1$ is a property-preserving transformation.

*Proof.* **(1)** The implication proof in $N$ means that within the prefix of any trace, either the two gates evaluate to the same value, or $prop\_p(t, c_1)$ evaluates to 0 and $prop\_p\big(t, struct\_pre(c_1)\big)$ to 1. The latter condition cannot happen since within any prefix, constraint $c_1$ must evaluate to 1, which implies that $t$ cannot evaluate to 1

and $prop\_p(t, c_1)$ to 0 concurrently. The implication proof thus ensures that if $t$ is asserted within any prefix at time $i$, then $struct\_pre(c_1)$ must evaluate to 1 at times 0 to $i$.

**(2)** Since $N$ and $N'$ have the same set of gates, they also have the same set of traces; only the constraint sets differ. The trace prefixing of $N'$ is stricter than that of $N$ as follows. **(a)** All traces prefixed at time 0 because of constraint $c_1$ in netlist $N$ are also prefixed at time 0 because of constraint $init\_cons(c_1)$ in $N'$. **(b)** All traces prefixed at time $i+1$ because of constraint $c_1$ in netlist $N$ are prefixed at time $i$ because of constraint $struct\_pre(c_1)$ in $N'$.

**(3)** For property-preservation, we must only ensure that target $t$ cannot be asserted during time-steps that were prefixed in $N'$ but not $N$. During such time-steps, $c_1$ evaluates to 1, and $struct\_pre(c_1)$ to 0, hence $prop\_p\big(t, struct\_pre(c_1)\big)$ must evaluate to 0. The proof of this implication check thus requires $prop\_p(t, c_1)$ to evaluate to 0 at such time-steps, ensuring that $t$ evaluates to 0.

□

Practically, we have found that the trace-prefixing of $c_1$ substantially reduces the complexity of the proof obligation of Theorem 6.8 vs. Theorem 6.7, e.g., by enabling low cost inductive proofs. This check tends to be significantly easier than the property check itself, as it merely attempts to validate that the modified constraint does not *alter* the assertability of the target along any trace, independently of whether the target is assertable or not. Additionally note that $init\_r$ can readily be eliminated using retiming.

# Chapter 7

# Conclusion

In this chapter, we summarize the contributions of this dissertation, and discuss future research directions. In this dissertation, we have made significant advances in the area of sequential redundancy identification and this has enabled scalable and efficient sequential equivalence checking across a wide variety of design modifications, and enhanced proofs we well as falsification in property checking. Our contributions are three-fold.

**Sequential Equivalence Checking**

We introduced a novel and flexible redundancy identification framework based on speculative-reduction in Chapter 3. This framework has enabled SEC solutions to be highly scalable, scaling up to designs with 10,000s of state elements and beyond. We also proposed the idea of using a flexible and synergistic set of transformation and verification algorithms to help identify redundancy in Chapter 3. This has enabled SEC solutions to be applicable across arbitrary design transformations. The development of highly scalable and widely applicable SEC solutions based on the techniques presented in this dissertation has resulted in SEC becoming a standard part of design methodologies at IBM. By becoming the main verification framework for validating micro-architectural optimizations for design closure at IBM, SEC has eliminated costly functional verification regression process and enabled

shorter design cycles. SEC has also opened the door for automated sequential synthesis through its ability to validate the sequential design transformations. Though such automated synthesis transformations have been developed decades ago, until the development of automated scalable SEC solutions, they have seen little use. Sequential synthesis is unavoidable if one wants to design high performance circuits at a higher level of abstraction. Design at a higher level of abstraction is an inevitable direction of digital design as it enables faster and more efficient design of complex systems and significantly reduces functional verification complexity [Spi04].

### Constraint-based Verification

To enable the application of SEC on designs with constraints, we have developed the theoretical framework with corresponding efficient implementation to enable the optimal sequential redundancy identification for designs with constraints in Chapter 5. We propose new algorithms for constraint-preserving testcase generation for simulation in the presence of dead-end states in Chapter 6. We also discuss how various automated netlist transformations may be optimally applied while preserving constraint semantics, including dead-end states.

### Transformation-based Verification

Transformation-based verification (TBV) is the key in enabling SEC across arbitrary design transformations. We enhance the scalability of TBV by proposing two new structural abstraction techniques in Chapter 4. We study the synergy that these netlist transformations have with each other, and also with other transformations such as retiming and redundancy removal. We also propose new strategies for using these transformations to enable scalable sequential redundancy identification.

While the focus of this dissertation is upon the benefits that the enhanced sequential redundancy identification may bring to a verification framework such as SEC or property checking, we note that the potential application domain of our work is significantly broader.

The netlist reductions enabled by our solution may reduce resources in val-

idation frameworks such as simulation and hardware emulation. They may also be used to enhance technology-independent logic synthesis by using the enhanced sequential redundancy removal techniques as a synthesis transformation.

### Future Work

There are numerous future work directions to enhance the results reported herein. It has been shown that observability don't cares (ODC) can significantly enhance combinational redundancy identification [ZKKSV06]. Use of ODCs to enhance sequential redundancy identification is still in the nascent stage and has great promise.

Clock gating is a well known concept for power saving and consists of shutting off the clock to certain areas of the chip during known periods of inactivity. Clock gating verification can be setup as a sequential equivalence checking problem where we equivalence check the design with clock gating enabled vs. clock gating disabled at time frames where these designs are equivalent to produce matching outputs. Though the two designs are structurally similar, this is a very difficult SEC problem since the gates across the two designs are equivalent only when clock is active (care condition). The use of ODCs to enhance this SEC check is an obvious solution. It furthermore would be interesting to explore the automated use of uninterpreted functions to transform the designs being equivalence checked.

As discussed in Chapter 3, transformation algorithms have the ability to trivialize certain SEC problems. For example, if retiming is the only sequential transformation that has been used during logic synthesis, a retiming engine as part of a transformation-based verification framework maybe able to transform the PSPACE-complete problem of SEC to a NP-complete problem of checking satisfiability [MBPK05]. Development of new netlist transformations that are related to the sequential synthesis transformations applied on the designs could be the key to scalable validation of sequential synthesis.

With further improvements to the scalability and applicability of sequential redundancy identification algorithms, we believe that extensive use of sequential synthesis and thereby design and verification at higher levels of abstraction will soon become a practical reality.

# Bibliography


[ABBSV00] Adnan Aziz, Felice Balarin, Robert Brayton, and Alberto Sangiovanni-Vincentelli. Sequential synthesis using S1S. *IEEET-CAD: IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 19:1149–1162, 2000.

[Acca] Accelera. *PSL Language Reference Manual*. http://www.eda.org/vfv.

[Accb] Accelera. *SystemVerilog Language Reference Manual*. http://www.systemverilog.org/.

[ADMS02] Demos Anastasakis, Robert Damiano, Hi-Keung Tony Ma, and Ted Stanion. A practical and efficient method for compare-point matching. In *Design Automation Conference*, pages 305–310, June 2002.

[AJS99] Mark D. Aagaard, Robert B. Jones, and Carl-Johans H. Seger. Formal verification using parametric representations of Boolean constraints. In *Design Automation Conference*, pages 402–407, June 1999.

[AMP04] Demos Anastasakis, Lisa McIlwain, and Slawomir Pilarski. Efficient equivalence checking with partitions and hierarchical cut-points. In *Design Automation Conference*, pages 539–542, June 2004.

[Arm66] D. B. Armstrong. On finding a nearly minimal set of fault detection tests for combinational logic nets. In *IEEE Transactions on Electronic Computers*, volume EC-15, pages 66–73, 1966.

[AS04] Mohammad Awedh and Fabio Somenzi. Increasing the robustness of bounded model checking by computing lower bounds on the reachable states. In *Formal Methods in Computer-Aided Design*, pages 230–244, November 2004.

[Bau02] Jason Baumgartner. *Automatic Structural Abstraction Techniques for Enhanced Verification*. PhD thesis, University of Texas, Dec. 2002.

[BC00] Per Bjesse and Koen Claessen. SAT-based verification without state space traversal. In *Formal Methods in Computer-Aided Design*, pages 372–389, November 2000.

[BCCZ99] Armin Biere, Alessandro Cimatti, Edmund M. Clarke, and Yunshan Zhu. Symbolic model checking without BDDs. In *Tools and Algorithms for Construction and Analysis of Systems*, pages 193–207, March 1999.

[BHSA03] Jason Baumgartner, Tamir Heyman, Vigyan Singhal, and Adnan Aziz. An abstraction algorithm for the verification of level-sensitive latch-based netlists. *Formal Methods in System Design*, pages 39–65, (23) 2003.

[BK01] Jason Baumgartner and Andreas Kuehlmann. Min-area retiming on flexible circuit structures. In *Int'l Conference on Computer-Aided Design*, pages 176–182, Nov. 2001.

[BK04] Per Bjesse and James Kukula. Using counter example guided abstraction refinement to find complex bugs. In *Design Automation and Test in Europe*, pages 156–161, 2004.

[BK05] Per Bjesse and Jim Kukula. Automatic generalized phase abstraction for formal verification. In *International Conference on Computer-Aided Design*, pages 1076–1082, November 2005.

[BKA02] Jason Baumgartner, Andreas Kuehlmann, and Jacob Abraham. Property checking via structural analysis. In *Computer-Aided Verification*, pages 151–165, July 2002.

[BM05] Jason Baumgartner and Hari Mony. Maximal input reduction of sequential netlists via synergistic reparameterization and localization strategies. In *Correct Hardware Design and Verification Methods*, pages 222–237, Oct. 2005.

[BM07] Robert Brayton and Alan Mishchenko. Scalable sequential verification. Technical report, ERL Technical Report, EECS Department, UC Berkeley, 2007.

[BO02] Valeria Bertacco and Kunle Olukotun. Efficient state representation for symbolic simulation. In *Design Automation Conference*, pages 99–104, June 2002.

[Bra93] Daniel Brand. Verification of large synthesized designs. In *International Conference on Computer-Aided Design*, pages 534–537, 1993.

[Bry86] Randal E. Bryant. Graph-based algorithms for Boolean function manipulation. *IEEE Transactions on Computers*, C-35(8):677–691, August 1986.

[BTA+00] Jason Baumgartner, Anson Tripp, Adnan Aziz, Vigyan Singhal, and Flemming Andersen. An abstraction algorithm for the verification of generalized C-slow designs. In *Computer-Aided Verification*, pages 5–19, July 2000.

[CBM89] O. Coudert, C. Berthet, and J. C. Madre. Verification of synchronous sequential machines based on symbolic execution. In *Workshop on Automatic Verification Methods for Finite State Systems*, June 1989.

[CCK+02] Pankaj Chauhan, Ed Clarke, James Kukula, Samir Sapra, Helmut Veith, and Dong Wang. Automated abstraction refinement for model

checking large state spaces using SAT based conflict analysis. In *Formal Methods in Computer-Aided Design*, pages 33–51, Nov. 2002.

[CCK04] Pankaj Chauhan, Edmund M. Clarke, and Daniel Kroening. A SAT-based algorithm for reparameterization in symbolic simulation. In *Design Automation Conference*, pages 524–529, June 2004.

[CDH+05] Eduard Cerny, Ashvin Dsouza, Kevin Harer, Pei-Hsin Ho, and Hi-Keung Tony Ma. Supporting sequential assumptions in hybrid verification. In *Asia and South Pacific Design Automation Conference*, pages 1035–1038, January 2005.

[CGJ+00] Edmund M. Clarke, Orna Grumberg, Somesh Jha, Yuan Lu, and Helmut Veith. Counterexample-guided abstraction refinement. In *Computer Aided Verification*, pages 154–169, July 2000.

[CGKS02] Edmund Clarke, Anubhav Gupta, James Kukula, and Ofer Strichman. SAT based abstraction-refinement using ILP and machine learning techniques. In *Computer-Aided Verification*, pages 265–279, July 2002.

[Coo71] S. A. Cook. The complexity of theorem-proving procedures. In *ACM Symposium on the Theory of Computing*, pages 151–158, May 1971.

[CP] Hyunwoo Cho and Carl Pixley. Apparatus and method for deriving correspondences between storage elements of a first circuit model and storage elements of a second circuit model. In *U.S. Patent 5,638,381*.

[ES03] Niklas Eén and Niklas Sörennson. Temporal induction by incremental SAT solving. In *Workshop on Bounded Model Checking*, 2003.

[FF56] L. R. Ford and D. R. Fulkerson. Maximal flow through a network. *Canadian Journal of Mathematics*, 8:399–404, 1956.

[FV99] Kathy Fisler and Moshe Vardi. Bisimulation and model checking. In *Correct Hardware Design and Verification Methods*, pages 338–341, September 1999.

[GGYA03] Aarti Gupta, Malay Ganai, Zijiang Yang, and Pranav Ashar. Iterative abstraction using SAT-based BMC with proof analysis. In *International Conference on Computer-Aided Design*, pages 416–423, November 2003.

[GL94] Orna Grumberg and David E. Long. Model checking and modular verification. *ACM Transactions on Programming Languages and System*, 16(3):843–871, 1994.

[GPB00] E. Goldberg, M. R. Prasad, and R. K. Brayton. Using SAT in combinational equivalence checking. In *International Workshop on Logic & Synthesis*, May 2000.

[HCC+00] Shi-Yu Huang, Kwang-Ting Cheng, Kuang-Chien Chen, Chung-Yang Huang, and F. Brewer. AQUILA: An equivalence checking system for large sequential designs. *IEEE Transactions on Computers*, 49(5):443–464, May 2000.

[HMB07] A. Hurst, A. Mishchenko, and R. K. Brayton. Fast minimum-register retiming via binary maximum-flow. In *Formal Methods in Computer-Aided Design*, pages 181–187, November 2007.

[Hoo93] John N. Hooker. Solving the incremental satisfiability problem. *Journal of Logic Programming*, 15(1-2):177–186, 1993.

[IBM] IBM Formal Verification Benchmark Library. http://www.haifa.il.ibm.com/projects/verification/RB_Homepage/fvbenchmarks.html.

[Int04] International Technology Roadmap for Semiconductors. http://public.itrs.net, 2004.

[JB06] Jie-Hong Jiang and Robert Brayton. Retiming and resynthesis: A complexity perspective. In *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, volume 25, pages 2674–2686, December 2006.

[JG94] P. Jain and Ganesh Gopalakrishnan. Efficient symbolic simulation-based verification using the parametric form of Boolean expressions. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 13(8):1005–1015, August 1994.

[JKS02] HoonSang Jin, Andreas Kuehlmann, and Fabio Somenzi. Fine-grain conjunction scheduling for symbolic reachability analysis. In *Tools and Algos. Construction and Analysis of Systems*, pages 312–326, April 2002.

[JWPB05] Christian Jacobi, Kai Weber, Viresh Paruthi, and Jason Baumgartner. Automatic formal verification of fused-multiply-add FPUs. In *Design, Automation and Test in Europe*, pages 1298–1303, March 2005.

[KB01] Andreas Kuehlmann and Jason Baumgartner. Transformation-based verification using generalized retiming. In *Computer-Aided Verification*, pages 104–117, July 2001.

[KH03] Zurab Khasidashvili and Ziyad Hanna. SAT-based methods for sequential hardware equivalence verification without synchronization. In *International Conference on Computer-Aided Design*, pages 1170 – 1175, 2003.

[KK07] Nathan Kitchen and Andreas Kuehlmann. Stimulus generation for constrained random simulation. In *International Conference on Computer-Aided Design*, pages 258–265, November 2007.

[Koh78] Zvi Kohavi. *Switching and Finite Automata Theory*. McGraw-Hill, 1978.

[KPKG02] Andreas Kuehlmann, Viresh Paruthi, Florian Krohm, and Malay Ganai. Robust Boolean reasoning for equivalence checking and functional property verification. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 21(12):1377–1394, December. 2002.

[KS00] James H. Kukula and Thomas R. Shiple. Building circuits from relations. In *Computer-Aided Verification*, pages 131–143, July 2000.

[Kue04] Andreas Kuehlmann. Dynamic transition relation simplification for bounded property checking. In *International Conference on Computer-Aided Design*, pages 50–57, November 2004.

[LC06] Feng Lu and K.-T. Cheng. IChecker: An efficient checker for inductive invariants. In *High Level Design, Validation, and Test Workshop*, pages 176–180, November 2006.

[LS91] C. Leiserson and J. Saxe. Retiming synchronous circuitry. *Algorithmica*, 6:5–35, 1991.

[LS05] Mark H. Liffton and Karem A. Sakallah. On finding all minimally unsatisfiable subformulas. In *Theory and Applications of Satisfiability Testing*, pages 173–186, June 2005.

[MA04] Kenneth L. McMillan and Nina Amla. Automatic abstraction without counterexamples. In *Tools and Algorithms for Construction and Analysis of Systems*, pages 2–17, April 2004.

[MB05] A. Mishchenko and R. Brayton. SAT-based complete don't-care computation for network optimization. In *Design, Automation and Test in Europe*, pages 412–417, March 2005.

[MBA05] Hari Mony, Jason Baumgartner, and Adnan Aziz. Exploiting constraints in transformation-based verification. In *Correct Hardware Design and Verification Methods*, pages 269–284, October 2005.

[MBP+04] Hari Mony, Jason Baumgartner, Viresh Paruthi, Robert Kanzelman, and Andreas Kuehlmann. Scalable automated verification via expert-system guided transformations. In *Formal Methods in Computer-Aided Design*, pages 159–173, November 2004.

[MBP+06] Hari Mony, Jason Baumgartner, Viresh Paruthi, Robert Kanzelman, and Geert Janssen. Scalable sequential equivalence checking across arbitrary design transformations. In *International Conference on Computer Design*, pages 159–173, November 2006.

[MBPK05] Hari Mony, Jason Baumgartner, Viresh Paruthi, and Robert Kanzelman. Exploiting suspected redundancy without proving it. In *Design Automation Conference*, pages 463–466, June 2005.

[MCB06] Alan Mishchenko, S. Chatterjee, and Robert Brayton. DAG-aware AIG rewriting: A fresh look at combinational logic synthesis. In *Design Automation Conference*, pages 532–535, July 2006.

[MCBE06] A. Mishchenko, S. Chatterjee, R. K. Brayton, and N. Een. Improvements to combinational equivalence checking. In *IEEE/ACM International Conference on Computer-Aided Design*, pages 836–843, 2006.

[MCBJ08] Alan Mishchenko, Michael Case, Robert Brayton, and Stephen Jang. Scalable and scalably-verifiable sequential synthesis. In *International Workshop on Logic and Synthesis*, 2008.

[McM03] K. L. McMillan. Interpolation and SAT-based model checking. In *Computer-Aided Verification*, pages 1–13, 2003.

[MHB98] Gurmeet Singh Manku, Ramin Hojati, and Robert K. Brayton. Structural symmetry and model checking. In *Computer-Aided Verification*, pages 159–171, July 1998.

[MHF98] Sela Mador-Haim and Limor Fix. Input elimination and abstraction

in model checking. In *Formal Methods in Computer-Aided Design*, pages 304–320, November 1998.

[MKK+02] In-Ho Moon, Hee Hwan Kwak, James Kukula, Thomas Shiple, and Carl Pixley. Simplifying circuits for formal verification using parametric representation. In *Formal Methods in Computer-Aided Design*, pages 52–69, Nov. 2002.

[MMZ+01] Matthew W. Moskewicz, Conor F. Madigan, Ying Zhao, Linto Zhang, and Sharad Malik. Chaff: Engineering an efficient SAT solver. In *Design Automation Conference*, pages 530–535, June 2001.

[Moo56] Edward F. Moore. Gedanken-experiments on sequential machines. In *Automata Studies*, volume 34, pages 129–153. Princeton University Press, 1956.

[MS03] Maher Mneimneh and Karem Sakallah. REVERSE: Efficient sequential verification for retiming. In *International Workshop on Logic and Synthesis*, 2003.

[NPMJ03] Kelvin Ng, Mukul R Prasad, Rajarshi Mukherjee, and Jawahar Jain. Solving the latch mapping problem in an industrial setting. In *Design Automation Conference*, pages 442–447, June 2003.

[NT00] Kedar Namjoshi and Richard Trefler. On the completeness of compositional reasoning. In *Computer Aided Verification*, pages 139–153, July 2000.

[Pix92] Carl Pixley. A theory and implementation of sequential hardware equivalence. In *IEEE Transactions on Computer-Aided Design*, pages 1469–1494, December 1992.

[Pix99] Carl Pixley. Integrating model checking into the semiconductor design flow. In *Electronic Systems Technology & Design*, pages 67–74, 1999.

[PJW05] Viresh Paruthi, Christian Jacobi, and Kai Weber. Efficient symbolic simulation via dynamic scheduling, don't caring, and case splitting. In *Correct Hardware Design and Verification Methods*, pages 114–128, Oct. 2005.

[RH07] Jie-Hong R. and Jiang Wei-Lun Hung. Inductive equivalence checking under retiming and resynthesis. In *IEEE/ACM International Conference on Computer-Aided Design*, pages 326–333, 2007.

[RS04] Kavita Ravi and Fabio Somenenzi. Minimal satisfying assignments for bounded model checking. In *Tools and Algorithms for Construction and Analysis of Systems*, pages 31–45, April 2004.

[Sht01] Ofer Shtrichman. Pruning techniques for the SAT-based bounded model checking problem. In *Correct Hardware Design and Verification Methods*, pages 58–70, September 2001.

[SK97] Dominik Stoffel and Wolfgang Kunz. Record and play: A structural fixed point iteration for sequential circuit verification. In *Int'l Conference on Computer-Aided Design*, pages 394–399, Nov. 1997.

[SPAB01] Vigyan Singhal, Carl Pixley, Adnan Aziz, and Robert Brayton. Theory of safe replacements for sequential circuits. In *IEEE Transactions on Computer-Aided Design*, pages 248–265, February 2001.

[Spi04] G. S. Spirakis. Designing for 65nm and beyond. In *Keynote Address at Design Automation and Test in Europe (DATE)*, 2004.

[SSS00] Mary Sheeran, Satnam Singh, and Gunnar Stalmarck. Checking safety properties using induction and a SAT-solver. In *Formal Methods in Computer-Aided Design*, pages 108–125, Nov. 2000.

[vE98] C. A. J. van Eijk. Sequential equivalence checking without state space traversal. In *Design, Automation and Test in Europe*, pages 618–623, February 1998.

[vE00] C. A. J. van Eijk. Sequential equivalence checking based on structural similarities. *IEEE Transactions on Computer-Aided Design*, 19(7):814–819, July 2000.

[Wan03] Dong Wang. *SAT based Abstraction Refinement for Hardware Verification*. PhD thesis, Carnegie Mellon University, May 2003.

[WHL+01] Dong Wang, Pei-Hsin Ho, Jiang Long, James H. Kukula, Yunshan Zhu, Hi-Keung Tony Ma, and Robert F. Damiano. Formal property verification by abstraction refinement with formal, simulation and hybrid engines. In *Design Automation Conference*, pages 35–40, June 2001.

[YAAP03] J. Yuan, K. Albin, A. Aziz, and C. Pixley. Constraint synthesis for environment modeling in functional verification. In *Design Automation Conference*, pages 296–299, June 2003.

[YPA05] Jun Yuan, Carl Pixley, and Adnan Aziz. *Constraint-Based Verification*. Springer Verlag, 2005.

[YSP+99] J. Yuan, K. Shultz, C. Pixley, H. Miller, and A. Aziz. Modeling design constraints and biasing in simulation using BDDs. In *International Conference on Computer-Aided Design*, pages 584–590, Nov. 1999.

[ZK03] Yunshan Zhu and James Kukula. Generator-based verification. In *International Conference on Computer-Aided Design*, pages 146–153, November 2003.

[ZKKSV06] Qi Zhu, Nathan Kitchen, Andreas Kuehlmann, and Alberto Sangiovanni-Vincentelli. SAT sweeping with local observability don't-cares. In *IEEE/ACM Design Automation Conference*, pages 229–234, July 2006.

# Vita

Hari Mony was born to Subramony Hariharaiyer and Jayalakshmi Mony, in Kollam, India in 1977. He received his Bachelor of Technology degree in Electrical Engineering from the Indian Institute of Technology, Kharagpur, in May 1999 and Master of Science degree in Computer Engineering from the University of Texas at Austin in May 2001. From July 2001, he has been engaged in the research, development, and support of SixthSense (tool for sequential equivalence checking and semi-formal verification) at IBM Systems & Technology Group. His research interests lie in automatic formal verification of large and complex industrial designs.

Permanent Address: 16516 Castletroy Dr
Austin, TX 78717

This dissertation was typeset with LaTeX $2_\varepsilon$[1] by the author.

[1] LaTeX $2_\varepsilon$ is an extension of LaTeX. LaTeX is a collection of macros for TeX. TeX is a trademark of the American Mathematical Society. The macros used in formatting this dissertation were written by Dinesh Das, Department of Computer Sciences, The University of Texas at Austin, and extended by Bert Kay, James A. Bednar, and Ayman El-Khashab.