

FP6-IST-507219

PROSYD:

Property-Based System Design

Instrument: Specific Targeted Research Project

Thematic Priority: Information Society Technologies

Property-Based Error Localization (Deliverable 2.2/2)

Due date of deliverable: April 30, 2005

Actual submission date: April 29, 2005

Start date of project: January 1, 2004

Duration: Three years

Organisation name of lead contractor for this deliverable: TU Graz

Revision 1.0

Project co-funded by the European Commission within the Sixth Framework Programme (2002-2006)		
Dissemination Level		
PU	Public	<input checked="" type="checkbox"/>
PP	Restricted to other programme participants (including the Commission Services)	<input type="checkbox"/>
RE	Restricted to a group specified by the consortium (including the Commission Services)	<input type="checkbox"/>
CO	Confidential, only for members of the consortium (including the Commission Services)	<input type="checkbox"/>

Notices

For information, contact Roderick Bloem rbloem@ist.tugraz.at.

This document is intended to fulfil the contractual obligations of the PROSYD project concerning deliverable 2.2/2 described in contract number 507219.

The information in this document is provided "as is", and no guarantee or warranty is given that the information is fit for any particular purpose. The user thereof uses the information at its sole risk and liability.

© Copyright PROSYD 2005. All rights reserved.

Table of Revisions

Version	Date	Description and reason	By	Affected sections
0.1	March 15, 2005	Creation	Staber	All
0.2	April 8, 2005	Finished correction approach	Bloem, Staber	Games, Solving Games
0.3	April 21, 2005	Version sent to partners	Bloem, Staber	All
1.0	April 29, 2005	Public version release	Bloem, Staber	All

Authors

Roderick Bloem
Stefan Staber

Executive Summary

Property verification tools, be they static or dynamic, yield a trace as evidence that a design violates a property. Getting from a trace to the cause of the failure can be very time consuming. The subject of this work is to automate the localization of a fault.

We present two methods. The **correction approach** combines fault localization with correction. If a property is violated, the method finds the possible locations for the fault and suggests corrections. We state the localization and correction problem as a game that is won if there is a correction that is valid for all possible inputs. This is a very powerful fully automatic method. For invariants, the most frequently used class of properties, the method is complete. The approach is based on a heuristic, therefore for complex general PSL properties we cannot guarantee that a correction is found, but in most cases we are successful. Finding a correction may be hard for complex circuits.

For large and complex circuits we introduce our **localization approach**. This method does not find corrections, but is based on consistency of the observations. It does not guarantee that all locations marked as suspicions can actually be used to correct the design, but it reduces the amount of the design to be inspected to a fraction and is computationally much easier. We provide an example where the number of components to be considered faulty is reduced to 7% of all components.

Purpose

The purpose of this document is to describe the work done on fault localization for PSL properties.

Intended Audience

This guide is intended for researchers working on verification tools using PSL or a similar specification language. It is assumed that readers are familiar with the notions and terms related to PSL and have a basic understanding of model checking.

Background

There are currently no methods for localization of faults detected with temporal properties. In fact, the work on sequential circuits is very sparse, as we discuss in the introduction. Most of the work is centered around a very limited class of faults that occur, for instance, when an optimizer makes a mistake. Existing work also assumes that a reference model is available. We do not have this requirement: the property takes the place of the reference model.

Contents

Table of Revisions	iii
Authors	iii
Executive Summary	iii
Purpose	iii
Intended Audience.....	iii
Background	iv
Contents	v
Table of Figures	vi
List of Tables	vii
Glossary	viii
1 Introduction and Background.....	1
Related Work.....	2
2 Correction Approach	5
Games for Localization and Correction.....	5
Solving Games.....	8
Strategies	8
Localization and Correction	9
3 Localization Approach.....	11
Model Based Diagnosis	11
Model Based Diagnosis for Sequential Systems and Temporal Properties..	14
4 Comparison of Strengths	19
Example - Sequential Multiplier.....	19
Correction approach	20
Localization approach.....	20
5 Conclusion	23
6 References.....	25

Table of Figures

Figure 1 - Simple circuit.....	5
Figure 2 - Unrolling of circuit in Figure 1	5
Figure 3 - Faulty system	6
Figure 4 - Corrected system	6
Figure 5 - Game to localize and correct the fault.....	6
Figure 6 - Request-Acknowledge circuit.....	14
Figure 7 - Unrolling and Extension of circuit in Figure 6	15
Figure 8 - Circuit with gate G2 as diagnosis.....	16
Figure 9 - Sequential Multiplier.....	20

List of Tables

Table 1 - Function for the system choice.....	7
Table 2 - Comparison of strengths.....	19

Glossary

Abductive based diagnosis

In abductive based diagnosis it is known in which ways a component can fail. Using these knowledge abductive based diagnosis tries to find a component of the model and a corresponding fault that explains the discrepancy between observed and desired behavior.

BDD

Binary Decision Diagram. A data structure for storing sets upon which many model checkers are based.

Combinatorial circuit

A logic circuit whose output is a function of only the present input.

Consistency based diagnosis

Fault diagnosis based on the principle of solving contradictions.

Constraints

A restriction on the behavior of a component.

Error

The difference between a computed, observed or measured value or condition and the true, specified, or theoretically correct value or condition. For example, a difference of 30 meters between a computed result and the correct result.

Failure

An incorrect result. For example, a computed result of 12 when the correct result is 10.

Fault

An incorrect step, process, or data definition. For example, an incorrect instruction in a computer program

Infinite Game

A finite state machine on which two players, the protagonist and the antagonist, determine the run, by each determining part of the input. The game comes with a winning condition and the task of the protagonist is to make sure that the run satisfies this condition.

Invariant

An invariant is a special case of a safety property. It sets a constraint on the state of the program that must hold in all states. For instance, “a and b are never 1 simultaneously.”

Liveness property

A liveness property states that something good should happen. For instance, “signal acknowledge is eventually one”.

Mistake

A human action that produces an incorrect result. For example, an incorrect action on the part of the programmer or operator.

Model Based Diagnosis

A formal method for fault localization. We distinguish between abductive based and consistency based diagnosis.

PSL

Property Specification Language, the language for specification of designs upon which PROSYD is based.

Safety Property

A safety property states that something bad should not happen. For instance, “a is never 1 in two consecutive clock ticks.”

SAT-solver

An algorithm for solving instances of the Boolean satisfiability problem.

Sequential circuit

A logic circuit whose output is a function of the present input and the previous state of the circuit.

Unrolling

By unrolling a sequential circuit for a finite amount of time frames a equivalent combinational circuit is provided.

1 Introduction and Background

Verification tools usually provide a trace if the design does not fulfill a property. Even if a failure trace is available, it may be hard to find the fault contained in the system. Researchers have taken different approaches to alleviate this problem. One approach is to make the traces themselves easier to understand. In the setting of model checking, Jin et al. [12] introduce an approach that identifies points of choice in the failure trace that cause the error and Ravi and Somenzi [20] propose a method to remove irrelevant variables from a counterexample derived using bounded model checking. Similarly, in the setting of software testing, Zeller and Hildebrandt [29] consider the problem of simplifying the input that causes failure.

A second approach to help the user understand a failure (which is not necessarily the same as locating the fault) is to consider several similar program traces, some of which show failure and some success [28, 10, 2, 22, 9]. The similarities between failure traces and their differences with the successful traces give an indication of the parts of the program that are likely involved in the failure.

In this report we present two formal approaches for fault localization and discuss their strengths and limitations.

In the **correction approach**, we take the view that a component may be responsible for a fault if it can be replaced by an alternative that makes the system correct. Thus fault localization and correction are closely connected, and we present an approach that combines the two. We assume a finite-state sequential system, which can be hardware or finite-state software. We furthermore assume that a (partial) specification is given in property specification language (PSL), and we endeavor to find and correct a fault in such a way that the new system satisfies its specifications for all possible inputs. Our fault model is quite general: we assume that any component can be replaced by an arbitrary function in terms of the inputs and the state of the system.

Prosyd Deliverable D2.2/1 (Property-based logic synthesis for rapid design prototyping) provides the background for our approach. The deliverable is due 1st September 2005. A preliminary description of part of that deliverable is available in Jobstmann et al. [13]. In that work, a method for the correction of a set of suspect components is presented. A restriction in the work is that a suspicion of the location of the fault has to be given by the user. We solve that restriction by integrating fault localization and correction.

We consider the fault localization and correction problem as an infinite game in which the system is the protagonist and the environment the antagonist. The winning condition for the protagonist is the satisfaction of the specification. The system first chooses which component is incorrect and then, at every clock cycle, which value to use as the output of the component. If for any input sequence, the system can choose outputs of the component such that the system satisfies the specification, the game is won. If the corresponding strategy is memoryless (the

output of the component depends only on the state of the system and its inputs), it prescribes a replacement behavior for the component that makes the system correct. The method is complete for invariants, and in practice works well for general PSL properties, even though it is not complete.

The **localization approach** is based on the theory of model based diagnosis, which provides a general, logic-based approach to fault localization. In contrary to the correction approach we perform fault localization but we do not provide a fix for the fault candidates. Again we assume a finite-state sequential system. A (partial) specification is given as an PSL-safety property.

In the diagnosis setting, a sequential system is unrolled with length of a given counterexample. By doing this an equivalent combinational representation of the sequential system is provided. In order to provide diagnoses, a reference model which states the correct behavior of the system is needed. In the usual setting, such a reference model is given by one test case, which is derived from the counterexample. The derivation of the test case is usually done by a verification engineer and must be performed separately for different counterexamples.

In our approach we propose to unroll the given PSL-safety property by applying the expansion rules in order to obtain a reference model automatically. We combine the unrolled PSL property and the unrolled sequential system and obtain a new combinational system. On this system we perform model based diagnosis in order to obtain fault candidates.

The method is efficient and is able to handle large and complex designs. It does not guarantee that all found fault candidates can be used to correct the design, but it provides a significant support for the verification engineer.

Related Work

Much work has been done in correcting combinational circuits. Typically, a correct version of the circuit is assumed to be available. (For instance, because optimization has introduced a bug.) These approaches are also applicable to sequential circuits, as long as the state space is not re-encoded. The work of Madre et al. [16] and Liaw et al. [15] discusses formal methods of fault localization and correction based on Boolean equations. The fault model of Madre et al. [16] is the same one we use for sequential circuits: any gate can be replaced by an arbitrary function. Chung, Wang, and Hajj [3] improve these methods by pruning the set of possible faults. They consider only a set of *simple*, frequently occurring design faults. In Tomita et al. [25] an approach is presented that may fix multiple faults of limited type by generating special patterns.

Work on sequential diagnosis and correction is more sparse. In the sequential setting, we assume that it is not known whether the state is correct at every clock tick, either because the reference model has a different encoding of the state space, or because the specification is given in a logic rather than as a circuit. Wahba and

Borrione [27] discuss a method of finding single errors of limited type (forgotten or extraneous inverter, and/or gate switched, etc.) in a sequential circuit. The specification is assumed to be another sequential circuit. Their algorithm finds the fault using a given set of test patterns. It iterates over the time frames, in each step removing from suspicion those gates that would, if changed, make a correct output incorrect or leave an incorrect output incorrect.

The report is structured as follows. In Section 2, we discuss our correction approach. We start with a motivating example and give necessary definitions for games. We show how the game can be solved and we prove the correctness and completeness of the game approach. In Section 3, we discuss our localization approach. We start with an overview of model based diagnosis and we present our new localization approach. We compare the two approaches in Section 4. In Section 5 we conclude.

2 Correction Approach

Games for Localization and Correction

We start with a simple example to explain the basic idea of the correction approach. Additionally, we introduce some formalisms necessary for the proof of correctness which we provide.

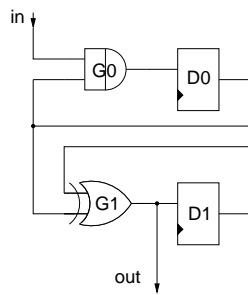


Figure 1: Simple circuit

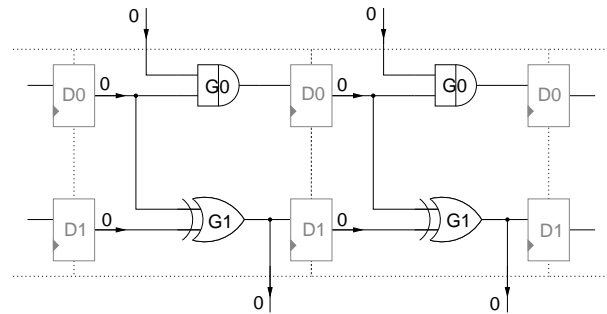


Figure 2: Unrolling of circuit in Figure 1

Figure 1 shows a simple sequential circuit. Suppose the initial state of the circuit is $(0,0)$ and the PSL specification is $\{(\neg \text{out}; \text{out})^*\}$. Figure 2 shows the unrolling of the circuit corresponding to a counterexample of length 2.

In order to identify faulty components, we need to decide what the components of the system are. In this report, the components that we use for circuits are gates or sets of closely related gates such as full adders or registers. For finite-state programs, our set of components consists of all expressions and the left-hand side of each assignment. Given a set of components, our approach searches for faulty components and corresponding replacement functions. The range of the replacement function depends on the component model, the domain is determined by the states and inputs. Note that the formulation of our approach is independent of the chosen set of components. We show how to search for faulty components and correct replacements by means of sequential circuits, where the specification F is the set of runs that satisfies some PSL formula φ . Our approach can handle multiple faults, but for simplicity we use a single fault to explain it. Thus, a correction is a replacement of one gate by an arbitrary Boolean function in terms of the primary inputs and the current state.

A circuit corresponds to a *finite state machine (FSM)* $M = (S, s_0, I, \delta)$, where S is a finite set of states, $s_0 \in S$ is the initial state, I is a finite set of inputs, and

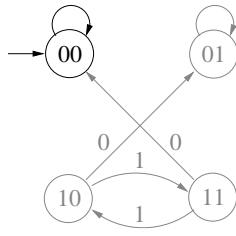


Figure 3: Faulty system

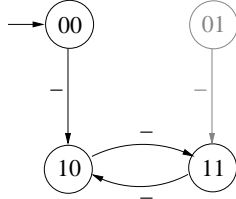


Figure 4: Corrected system

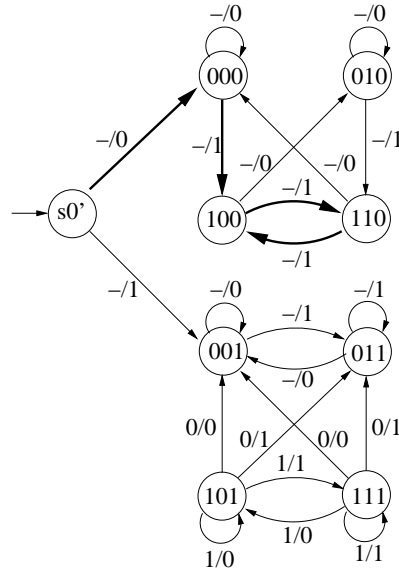


Figure 5: Game to localize and correct the fault

$\delta : S \times I \rightarrow S$ is the transition function. For example, if we are given the circuit in Figure 1 and we want it to fulfill the specification $\{(\neg \text{out}; \text{out}^*)\}$, we obtain the FSM shown in Figure 3.

We extend the FSM to a game between the system and the environment. A *game* G is a tuple $(S, s_0, I, C, \delta, F)$, where S is a finite set of states, $s_0 \in S$ is the initial state, I and C are finite sets of environment inputs and system choices, $\delta : S \times I \times C \rightarrow S$ is the complete transition function, and $F \subseteq S^\omega$ is the winning condition which corresponds to the specification. The winning condition is defined over sequences of states. To simplify matters, we translate the given specification in a corresponding set of sequences. In our example $\{(\neg \text{out}; \text{out}^*)\}$ corresponds to all sequences in which D_1 is 0 in the first two time frames and alternates between 1 and 0 afterwards.

Suppose we are given a circuit and the gates in the circuit are numbered by $0 \dots n$. We extend the corresponding FSM $M = (S, s_0, I, \delta)$ to a game by the following two steps

1. We extend the state space to $(S \times \{0 \dots n\}) \cup s'_0$. Intuitively, if the system is in state (s, d) , we suspect gate d to be incorrect. s'_0 is a new initial state. From this state, the system can choose which gate is suspect.
2. We extend the transition relation to reflect that the system can choose the output of the suspect gate.

If gate d is suspect, it is removed from the combinational logic of our circuit, and we obtain new combinational logic with one more input (and some new outputs, which we ignore). Let the function computed by this new circuit be given by $\delta_d : S \times I \times \{0, 1\} \rightarrow S$, where the third argument represents the new input.

Table 1: Function for the system choice

State			Input	Choice
$S \times \{0, 1\}$			I	C'
D_0	D_1	d	i	c
0	0	0	0	1
0	0	0	1	1
0	1	0	0	-
0	1	0	1	-
1	0	0	0	1
1	0	0	1	1
1	1	0	0	1
1	1	0	1	1

We construct the game $G = (S', s'_0, I, C', \delta', F')$, where

$$\begin{aligned}
 S' &= (S \times \{0, \dots, n\}) \cup s'_0, \\
 C' &= \{0 \dots n\}, \\
 \delta'(s'_0, i, c) &= (s_0, c), \\
 \delta'((s, d), i, c) &= (\delta_d(s, i, c \bmod 2), d), \\
 F' &= \{s'_0, (s_0, d_0), (s_1, d_1), \dots \mid s_0, s_1, \dots \in F\}.
 \end{aligned}$$

Note that the full range of the system choice ($\{0 \dots n\}$) is only used in the new initial state s'_0 to choose the suspect gate. Afterwards, we only need two values to decide the correct output of the gate (0 and 1), so we use the modulo operator.

For our simple example, we obtain the game shown in Figure 5. In the initial state the system chooses which of the gates (G_0 or G_1) is faulty. The upper part of the game in Figure 5 corresponds to an arbitrary function for gate G_0 , the lower one represents a replacement of gate G_1 . The edges are labeled with the values of environment input i and the system choice c separated by a slash, e.g., the transition from state 100 to 010 labeled with $-/0$ means that starting at $D_0 = 1$ and $D_1 = 0$ and assuming G_0 to be faulty, the system choice $C = 0$ forces the latches to be $D_0 = 0$ and $D_1 = 1$ in the next state regardless of the input.

Once we have constructed the game, we select system choices that restrict the game to those paths that fulfill the specification. In our example, first we choose a transition from s'_0 to either the upper or the lower part of the game. Choosing the transition from s'_0 to 000 means we try to fix the fault by replacing gate G_0 . In state 000 we select transitions that lead to paths that adhere to the given specification. In Figure 5 the bold arrows only allow paths with the sequence 001010... for D_1 as required by the specification. Taking only these transitions into account we get the function shown in Table 1 for the system choice c . For the 3rd and 4th Line in Table 1 we can choose arbitrary values for the system choice. This choice gives us freedom in picking the desired correction. Since we aim for corrections that yield to simple modified systems, we choose the simplest implementation, which sets $c = 1$ all the time. Using the corresponding transitions in the original model (Figure 3) yields the correct model shown in Figure 4.

Choosing the right transitions of the game corresponds to searching a memoryless winning strategy for the system that fulfills the winning condition F' . Formally,

given a game $G = (S, s_0, I, C, \delta, F)$, a *memoryless strategy* is a function $\sigma : S \times I \rightarrow 2^C$, which fixes a set of possible responses to an environment input. A *play* on G according to σ is a finite or infinite sequence $\pi = s_0 \xrightarrow{i_0 c_0} s_1 \xrightarrow{i_1 c_1} \dots$, such that $c_i \in \sigma(s_i, i_i)$, $s_{i+1} = \delta(s_i, i_i, c_i)$, and either the play is infinite, or $\exists n : \sigma(s_n, i_n) = \emptyset$, which means that the play is finite. A play is *winning* (for the system) if it is infinite and $s_0 s_1 \dots \in F$. A strategy σ is *winning* on G if all plays according to σ on G are winning. Depending on the winning condition we distinguish different types of games. The winning condition of an *PSL game* is the set of sequences satisfying an PSL formula φ . A *safety game* has the condition $F = \{q_0 q_1 \dots \mid \forall i : q_i \in A\}$ for some A . The type of the game for localizing and correction depends on the specification. In the next section, we explain how to obtain a winning strategy and we prove the correctness of our approach.

Note that we cannot find a memoryless winning strategy if we choose gate G_1 for replacement. Reconsider the game in Figure 5. If we choose gate G_1 in the initial state, our next state is 001. For this state we cannot find a memoryless strategy such that the specification is fulfilled. To get the desired 0 on the output we have to stay in state 001 in the first step. In the second step we have to go to state 011 to get 1 on the output. Without additional state we cannot decide in state 001 if we either stay in the state or if we must go to state 011.

In order to handle multiple faults we extend the game to select a set of suspect components in the initial state. In every following state the system chooses an output for each suspect component. Thus, the range of the replacement function consists of tuples of outputs, one output for each suspect component.

Solving Games

Our approach to solve games is based on finding a winning strategies for a game, as described in deliverable D2.2/1. We first summarize that approach and in subsection Localization and Correction we formalize our new approach and prove that a winning strategy corresponds to a valid correction and that for invariants a winning strategy exists iff a correction exists.

Strategies

For a set $A \subseteq S$ let

$$MXA = \{s \mid \forall i \in I \exists c \in C, s' \in A : (s, i, c, s') \in \delta\}$$

be the set of states from which, for any input, the system can force a visit to a state in A in one step. We define $MGA = \nu Z.A \cap MXZ$ to be the set of states from which the system can avoid leaving A . (The symbol ν denotes a greatest fixpoint,

see [4].) Note that the MX operation is similar to the preimage computation in symbolic model checking, apart from the quantification of the input variables. The MG operation mirrors EG.

If the specification is an invariant A , the set MGA is exactly the set of states from which the system can guarantee that A is always satisfied. If the initial state is in MGA , the game is won. The strategy for a safety game is then easily found. From any state, and for any input, select any system choice such that the next state is in MGA :

$$\sigma(q, i) = \{c \in C \mid \delta(q, i, c) \in A\}.$$

Note that the strategy is immaterial for nodes that are unreachable. The same holds for states that are not winning: they will never be visited.

For PSL specifications, the situation is more intricate.

A *finite-state strategy* determines the set of allowed system choices using a finite-state machine that has a memory of the past input and system choices. A finite-state strategy may, for example, alternately pick two different choices for one and the same system state and input.

We can compute a finite-state strategy for a game with winning condition φ by finding a strategy on the product of the game and a deterministic automaton for φ . A finite-state strategy corresponds to a correction in which the new FSM is the product automaton. Thus, it would add state that corresponds to the automaton for φ .

Finding a deterministic automaton for φ is hard in terms of implementation. Furthermore the automaton needs doubly exponential space in the size of the specification, and it is probably a bad idea to fix a simple fault by the addition of such a large amount of state. Therefore, deliverable D2.2/1 proposes a heuristic approach. The approach constructs a nondeterministic Büchi automaton from φ in the standard way [26], which causes only a singly exponential blowup. Then, to avoid adding state to the circuit, they present a heuristic to turn a finite-state strategy into a memoryless strategy. The heuristic works by finding choices that are common to all states of the finite-state strategy. These two heuristics imply that the method is not complete: if the property is not an invariant, a correction may not be found even if it exists. We take the view that this tradeoff is necessary for efficiency and simplicity of the correction.

The complexity of the approach is comparable to that of symbolic model checking of a property on the game that has $O(k \cdot \lg |\text{COMP}|)$ more Boolean state variables than the original system, where k is the number of faults assumed.

Localization and Correction

In this section we formalize our correction approach and prove that the approach is sound and for invariants complete.

If a winning positional strategy for the system exists, it determines (at least one) incorrect gate plus a replacement function. To see this, we need some definitions.

For a function $f : S \times I \rightarrow \{0, 1\}$, let $\delta[d/f]$ be the transition function obtained from δ by replacing gate d by combinational logic specified by f : $\delta[d/f](s, i) = \delta_d(s, i, f(s, i))$. Let $M[d/f]$ be the corresponding FSM. Let $\sigma : ((S \times \{0 \dots n\}) \cup s'_0) \times I \rightarrow 2^{\{0 \dots n\}}$ be a winning finite-state strategy. Note that $\sigma(s'_0, i)$ is independent of i , and let $D = \sigma(s'_0, i)$ for some i . Let \mathcal{F}_d be the set of all functions $f : S \times I \rightarrow \{0, 1\}$ such that $f(s, i) \in \{c \bmod 2 \mid c \in \sigma((s, d), i)\}$. We claim that D contains only correctable single-fault diagnoses and $\{\mathcal{F}_d\}_{d \in D}$ contains only valid corrections, and that for invariants there are no other single correctable diagnoses or corrections.

Theorem 1. *Let $d \in \{0 \dots n\}$ and let $f : S \times I \rightarrow \{0, 1\}$. We have that $d \in D$ and $f \in \mathcal{F}_d$ implies that $M[d/f]$ satisfies F . If F is an invariant, then $M[d/f]$ satisfies F implies $d \in D$ and $f \in \mathcal{F}_d$.*

Proof. Suppose $d \in D$ and $f \in \mathcal{F}_d$. Let $\pi = (s'_0, (s_0, d), (s_1, d), \dots)$ be the play of G for input sequence i'_0, i_0, i_1, \dots so that $(s_{j+1}, d) = \delta'((s_j, d), i_j, f(s_j, i_j))$. Since $f(s_j, i_j) \in \sigma((s_j, d), i_j) \pmod{2}$, π is a winning run and $s_0, s_1, \dots \in F$. Now note that $(s_{j+1}, d) = \delta'((s_j, d), i_j, f(s_j, i_j)) = (\delta_d(s, i_j, f(s_j, i_j)), d) = (\delta[d/f](s_j, i_j), d)$. Thus, s_0, s_1, \dots is the run of $M[d/f]$ for input sequence i_0, i_1, \dots , and this run is in F .

For the second part, suppose F is an invariant, and say $M[d/f]$ satisfies F . Then for any input sequence, the run of $M[d/f]$ is in F , and from this run we can construct a winning play as above. The play stays within the winning region, and by construction of the strategy for a safety game, all system choices that do not cause the play to leave the winning region are allowed by the strategy. Thus, the play is according to the winning strategy, so $d \in D$ and $f \in \mathcal{F}_d$. \square

As stated before, we can extend the theorem to PSL specifications only if we allow a correction to add extra state to the system. Therefore for general PSL properties we use heuristics that work very well in practice.

3 Localization Approach

Model Based Diagnosis

Model based diagnosis is a technique which provides a general, logic-based approach to fault localization and has been studied for several years. In this section, we summarize the approach and discuss its advantages and limitations.

Model based diagnosis originates with the localization of faults in physical systems. It assumes a model of the correct functioning of the system and an observation of the actual system that is in contradiction with the model. As output, it yields a set of components that may be to blame for the unexpected behavior.

Console et al. [5] show the applicability of model based diagnosis to fault localization in logic programs. In the setting of diagnosis of software, a model is derived from the source code of the program. It describes the actual, faulty behavior of the system. An oracle provides an example of correct behavior that is inconsistent with the actual behavior of the program. Using the model and the desired behavior, model based diagnosis yields a set of components that may have caused the fault. This approach has been extended to functional programs [23], hardware description languages [7], and object oriented programs [18].

Model based diagnosis comes in two flavors: abduction-based and consistency-based diagnosis [6]. Abduction-based diagnosis [19] assumes that the set of fault models is enumerated, i.e., it is known in which ways a component can fail. Using these fault models, it tries to find a component of the model and a corresponding fault that explains the observation.

In the model based diagnosis view, the approach of Wahba and Borriore [27] can be seen as an abductive approach: it works with a small set of given fault models and combines localization with correction.

Consistency-based diagnosis [14, 21] does not require the possible faults to be known, but rather tries to make the model consistent with the correct behavior by finding a component such that dropping any assumption on the behavior of the component causes the contradiction between the model and the correct behavior to disappear. In this setting, components are described as constraints, for example, an AND gate x with inputs i_1 and i_2 is described as

$$\neg ab_x \Rightarrow (out_x \Leftrightarrow i_1 \wedge i_2),$$

where ab means that x is abnormal and considered responsible for the failure. Note that nothing is stated about the behavior of the gate when abnormal is asserted. The task of consistency-based diagnosis is to find a minimal set Δ of components such that the assumption

$$\{ab_c \mid c \in \Delta\} \cup \{\neg ab_c \mid c \in \text{COMP} \setminus \Delta\}$$

is consistent with the oracle (where COMP is the set of components).

Consistency based reasoning has weaknesses when multiple instances of a component appear, for instance in the unrolling of a sequential circuit. (A similar observation is made in [24] for multiple test cases.) In diagnosis of sequential circuits, as in its combinational counterpart, the aim is to find a small set of components that explains the observations. A single incorrect trace is given and diagnosis is performed using the unrolling of the circuit as the model. A single faulty predicate is used for all occurrences of a given component. Hamscher and Davis [11] show that consistency-based diagnosis has a drawback in this setting: If dropping the constraints of a component removes any dependency between input and output, that component is a diagnosis. In sequential circuits, because of the replication of components, this is possible to hold for some components.

To illustrate the functioning and the limitation of model based diagnosis, we use the formalism introduced by Reiter [21] on our example. Reconsider the sequential circuit shown in Figure 1. Suppose the initial state of the circuit is $(0,0)$ and the specification is $\{(\neg \text{out}; \text{out})^*\}$. Figure 2 shows the unrolling of the circuit corresponding to a counterexample of length 2.

In model based diagnosis we define a *system* as a pair (SD, COMP) where SD is the system description and COMP is a finite set of components. The circuit in Figure 2 may be represented by a system with components $\text{COMP} = \{G0_0, G1_0, G0_1, G1_1\}$ and the following system description SD :

$$\begin{aligned} \neg ab(G0) &\Rightarrow [(g0_0 \Leftrightarrow in_0 \wedge D0_0) \wedge (g0_1 \Leftrightarrow in_1 \wedge g0_0)], \\ \neg ab(G1) &\Rightarrow [(g1_0 \Leftrightarrow D0_0 \oplus D1_0) \wedge (g1_1 \Leftrightarrow g0_0 \oplus g1_0)], \\ out_0 &\Leftrightarrow g1_0, \text{ and} \\ out_1 &\Leftrightarrow g1_1. \end{aligned}$$

Here, $g0_i$ and $g1_i$ correspond to the outputs of gates $G0$ and $G1$ respectively.

As observation we use the test case $\text{OBS} = \{in_0 = 0, in_1 = 0, out_0 = 0, out_1 = 1\}$. This test case satisfies the specification.

If we assume that no component is abnormal, we can derive following behavior:

$$\begin{aligned}
& (g0_0 \Leftrightarrow 0) \wedge (g0_1 \Leftrightarrow 0), \\
& (g1_0 \Leftrightarrow 0) \wedge (g1_1 \Leftrightarrow 0), \\
& \text{out}_0 \Leftrightarrow 0, \text{ and} \\
& \text{out}_1 \Leftrightarrow 0.
\end{aligned}$$

The observation OBS conflicts with the behavior derived from SD, because the observation demands that $\text{out}_1 = 1$.

Now we want to find a minimal set of components Δ such that

$$SD \cup OBS \cup \{ab_c \mid c \in \Delta\} \cup \{\neg ab_c \mid c \in \text{COMP} \setminus \Delta\}$$

is consistent.

Consider the AND gate. If we state that the gate is abnormal, the constraints for the gate output is removed. Now any gate output is possible. From the system description we derive following behavior:

$$\begin{aligned}
& (g1_0 \Leftrightarrow 0) \wedge (g1_1 \Leftrightarrow g0_0 \oplus g1_0), \\
& \text{out}_0 \Leftrightarrow 0, \text{ and} \\
& \text{out}_1 \Leftrightarrow g0_0 \oplus g1_0.
\end{aligned}$$

The assumption that the AND gate is faulty (abnormal), together with the assumption that the XOR gate is behaving correctly, is consistent with the system description and the observation. Therefore the AND gate is a diagnosis.

Now consider the XOR gate. If we state the XOR gate as abnormal we derive from the system description following behavior:

$$\begin{aligned}
& (g0_0 \Leftrightarrow 0) \wedge (g0_1 \Leftrightarrow 0), \\
& \text{out}_0 \Leftrightarrow g1_0, \text{ and} \\
& \text{out}_1 \Leftrightarrow g1_1.
\end{aligned}$$

This behavior is consistent with the observation. Therefore the XOR gate is also a diagnosis.

The conclusion that either gate can be the cause of the failure, however, is incorrect. There is no replacement for the XOR gate that corrects the circuit: for the output of the circuit to be correct for the given inputs, the output of the XOR gate needs to be 0 in the first and 1 in the second time frame. This is impossible because the inputs to the gate are necessarily 0 in both time frames. The circuit can be corrected, but the only single consistent replacement to fix the circuit for the given input sequence is to replace the AND gate by a gate whose output is 1 when both inputs are 0.

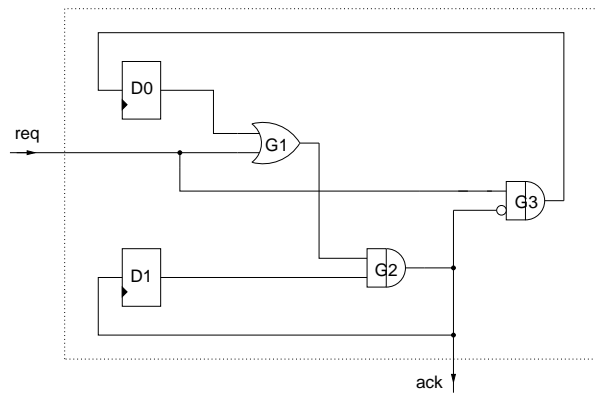


Figure 6: Request-Acknowledge circuit

Regardless of the presented drawback, the model based approach has achieved wide recognition due its advantages. Model based diagnosis provides a clear formal theory for diagnosis and has successfully been applied to various areas. It is able to handle large designs and reduces the set of components to be considered faulty to a fraction. Model based diagnosis is able to handle single or multiple faults in a system without alteration of the derived model. A single fault diagnosis represents one faulty component that explains the misbehavior of the system. Multiple fault diagnosis explains the faulty behavior by declaring more than one component as abnormal.

The complexity of model based diagnosis is in the worst case exponential in the number of components. A common way to obtain a good average performance is to locate only single fault diagnoses. This is a reasonable course of action because single faults are simpler explanation for the fault than multiple fault diagnoses and therefore more plausible.

Model Based Diagnosis for Sequential Systems and Temporal Properties

In this section, we describe the basic idea of our localization approach. We explain how we extend the model based diagnosis approach and how fault locations are obtained. We start with a simple example.

Figure 6 shows a simple arbiter with an request input and an acknowledge output. The circuit should satisfy the following properties: if a request is pending in a time frame it must be acknowledged in the same or in the next time frame; the acknowledge output must not be 1 in two consecutive time frames.

We state following PSL property for the desired behavior:

$$G((req \Rightarrow (ack \vee Xack)) \wedge (ack \Rightarrow \neg Xack)).$$

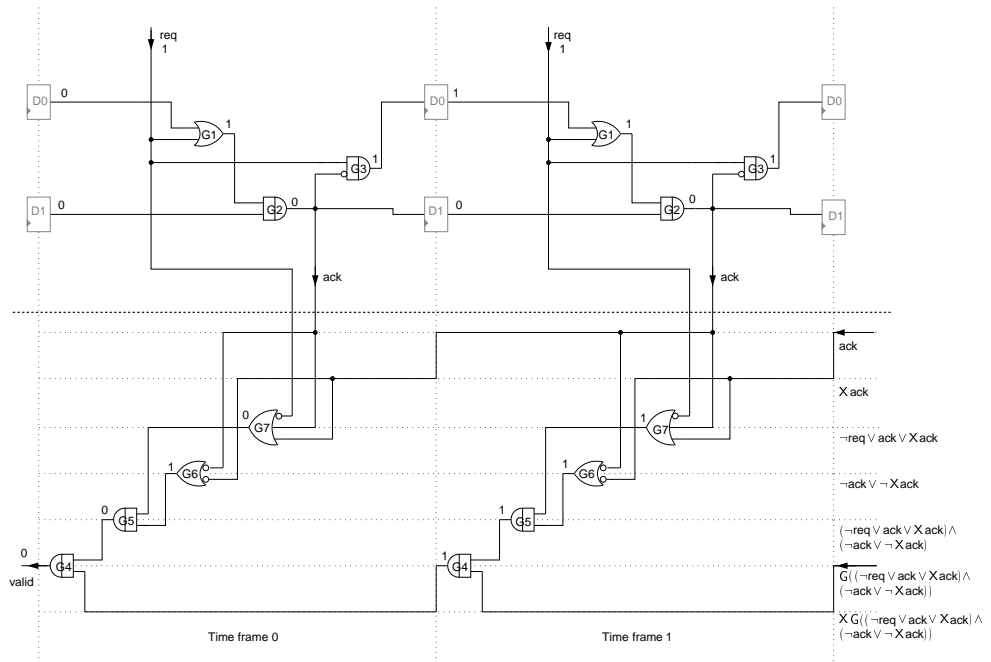


Figure 7: Unrolling and Extension of circuit in Figure 6

If we rewrite the implications we obtain following property:

$$G((\neg \text{req} \vee \text{ack} \vee X\text{ack}) \wedge (\neg \text{ack} \vee \neg X\text{ack})).$$

The circuit in Figure 6 does not fulfill that specification. Suppose that the initial state of both flip flops is 0. The shortest counterexample we obtain has length two: if we have requests in the first two time frames, output acknowledge is 0 in both frames.

In order to locate the fault the circuit is unrolled in the length of the counterexample. For computing the diagnosis we need a reference model which states the correct behavior of the circuit. Usually the user has to provide the reference model, i.e., a test case that was manually derived. We propose to derive a reference model from the given specification automatically. To this end we propose to extend the unrolled circuit with a combinational part that corresponds to the PSL specification. In order to do that, we use the *expansion rules* to *unroll* the formula [17, 8] (e.g., $F\varphi \equiv \varphi \vee XF\varphi$ and $G\varphi \equiv \varphi \wedge XG\varphi$). On the extended circuit model based diagnosis is applied and fault candidates are provided. We continue with our example in order to illustrate the extension of the circuit.

Figure 7 shows the unrolled circuit extended with the unrolled PSL specification. The gates below the vertical, dashed line correspond to the unrolled formula. Note that we obtain new inputs on the right side and a output valid on the left side of the figure. Value 0 on output valid indicates that we have a contradiction, i.e. the circuit does not fulfill the specification.

Every gate output in the lower part corresponds to a sub-formula of the specification. Note the labeling of the dashed horizontal lines. For example the output of gate G7 in time frame 0 corresponds to $(\neg \text{req}_0 \vee \text{ack}_0 \vee X\text{ack}_0)$. The signal $(X\text{ack}_0)$ is equivalent to ack_1 . Therefore we take as input for $(X\text{ack})$ in time

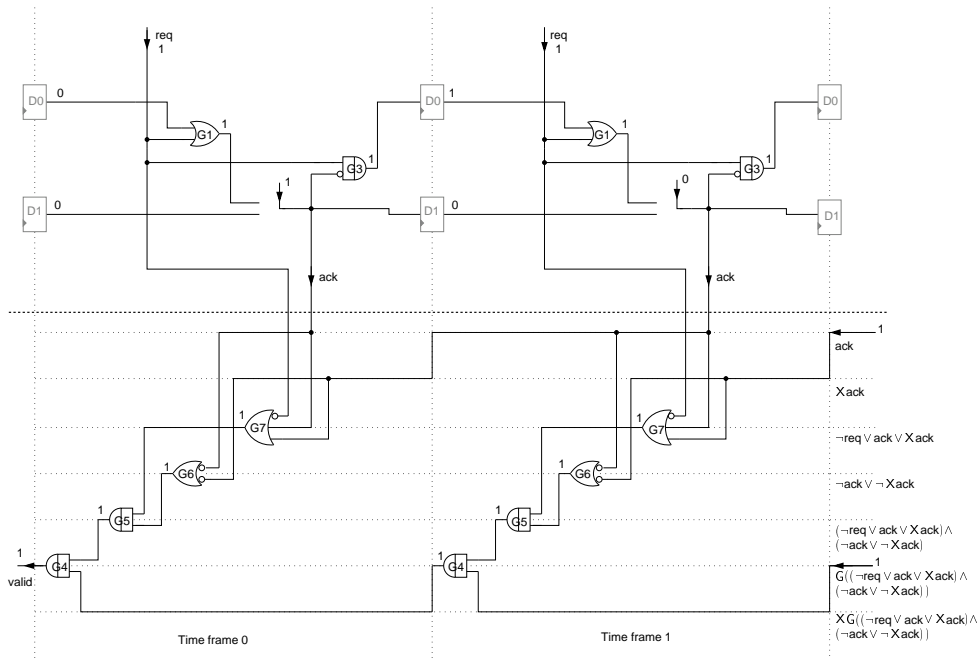


Figure 8: Circuit with gate G2 as diagnosis

frame 0 the signal ack from the next time frame. Due to this dependencies, the *direction* of the lower part is reversed: as inputs in a time frame we need outputs from a succeeding time frame.

Because we have a finite unrolling we obtain new inputs for those signals that are not available in a succeeding time frame. For example $Xack_1$ is equivalent to ack_2 . Time frame 2 is not available, and therefore the signal ack_2 is a new input. We want to find diagnoses for the length of the unrolling and therefore the succeeding time frames may not influence the diagnoses. We will soon see how values for the new inputs are chosen.

We introduce abnormal predicates for the components, as in the usual model based diagnosis approach. If a gate is declared as abnormal, the constraints for the gate output are removed in every time frame of the unrolling. If the gate is a valid diagnosis, the circuit is now consistent with the specification. We can confirm that by applying assignments to the inputs such that output valid of the unrolled circuit is 1. It is obvious that only gates of the original circuit can be declared as abnormal. Gates in the unrolled formula cannot be diagnoses.

In our example, AND gate G2 is a valid diagnosis. If we suspend the behavior of the gate we obtain a consistent circuit (see Figure 8).

The simple example indicates a limitation of our approach. For the unrolling of the circuit and the specification we need a counterexample without cycles. If a safety property fails, it is guaranteed that we obtain a counterexample without cycles. Consider the liveness property $F\phi$. A counterexample for that property is an infinite simple path where $G\neg\phi$ holds. Clearly we are not able to unroll an infinite path.

For properties that contain a safety and a liveness part, we are able to find diagnoses for the safety part. Consider the property $G\phi \wedge F\psi$. If the safety part $G\phi$ fails, we obtain a finite counterexample and we are able to unroll the circuit and the

property with length of the counterexample. We apply the expansion rules to the safety and the liveness part of the formula and obtain new inputs. Because we compute diagnosis for the safety part of the formula, we assume that the liveness part is satisfied by the circuit. A liveness property states that something good should happen in the future. The “future” in our circuit is represented by the new inputs obtained by the unrolling of the property. Therefore we can guarantee that the liveness part is satisfied, by applying the right values to the obtained inputs for the liveness property. In our example we set the obtained input for $F\psi$ to 1 and guarantee that the liveness part is satisfied.

For the calculation of the diagnoses we use a SAT-solver. Given a formula in Conjunctive Normal Form (CNF), a SAT-solver finds a variable assignment that satisfies the formula or proves that no variable assignment satisfies the CNF formula.

The CNF for the SAT-solver is composed of following subformulas:

- A formula representing the unrolled circuit and the input values corresponding to the counterexample. For the components an abnormal predicate is introduced.
- A formula representing the unrolled PSL property. The formula additionally states that the property must be satisfied.
- A formula that states that either every component is correct or one component is abnormal and the rest is correct.

To satisfy the CNF formula the SAT-solver has to find assignments for the abnormal predicates and the new inputs of the unrolled formula such that the circuit and the specification are consistent.

Instead of a SAT-solver we can use a pseudo-Boolean solver such as PBS [1]. Decision problems with pseudo-Boolean constraints are a special case of Integer Linear Programming (ILP) problems. PBS accepts a CNF formula extended with a pseudo-Boolean constraint. A pseudo-Boolean constraint has the form:

$$\left(\sum_{i=1}^n c_i \cdot b_i\right) \bowtie k$$

where $1 \leq i \leq n$, b_i is a Boolean variable, c_i is a rational constant, k is a rational constant, and \bowtie is one of $\{<, \leq, >, \geq, =\}$. For our purpose each b_i is one of the abnormal predicates, and each c_i is 1. If we remove the sub formula from the CNF which states that either every component is correct or one component is abnormal we can use the pseudo-Boolean constraint to solve a lot of different problems. For example we are able to encode the constraint that the number of abnormal components equals a constant k . In this way we can determine multiple fault diagnoses. In addition to the $\{<, \leq, >, \geq, =\}$ constraints, PBS can solve pseudo-Boolean optimization problems, i.e. minimizing or maximizing k . For our purpose we minimize k . The solution returned by PBS provides single fault diagnoses if exist, otherwise the minimum multiple fault diagnoses.

Table 2: Comparison of strengths

	Correction approach	Localization approach
Precision	suggested locations are correct and a correction is provided	located faults might be spurious
Specification	correct and complete for invariants, for general properties correct but not complete	applicable to safety properties and safety part of general properties
Complexity	finding strategies on infinite games	satisfiability of Boolean formulas

4 Comparison of Strengths

In this section we compare our two approaches and show their strengths and limitations.

Our correction approach is based on infinite games. The approach is able to locate and correct faults in finite state systems with a specification given in PSL. The approach is precise: a suggested correction is valid for all possible input sequences and therefore the approach is sound. If the specification is an invariant, the approach is complete: for a single point of failure, the fault is always found and corrected. For general properties we use heuristics that perform well in practice. The complexity of the correction approach corresponds to finding strategies on infinite games. Therefore the approach may fail on large systems.

Our localization approach is related to model based diagnosis. In contrary to the correction approach we do not take all possible input values into account. Using the unrolled specification, the approach is able to locate faults but does not provide corrections for the faults. Located faults might be spurious: consistency based diagnosis is not able to differentiate between correctable and non-correctable diagnoses. Nevertheless the localization approach is able to restrict the number of components to be considered faulty to a fraction. The approach is restricted to pure safety properties or to the safety part of a general PSL property, which are the most important properties. Due to the fact that diagnoses are calculated by applying a SAT solver we are able to handle large designs in a fast way.

Example - Sequential Multiplier

In this section we use a small example to compare our two approaches. The four-bit sequential multiplier shown in Figure 9 is introduced by Hamscher and Davis

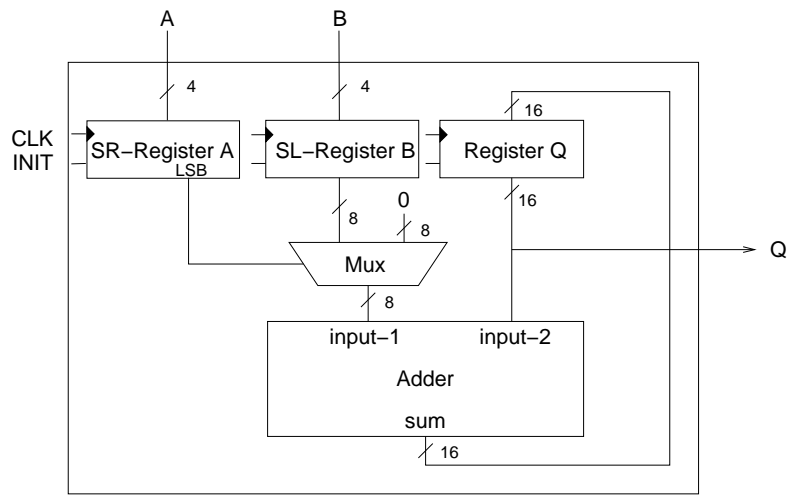


Figure 9: Sequential Multiplier

[11]. The multiplier has two input shift-registers, A and B, and a register Q which stores intermediate data. If input INIT is high, shift registers A and B are loaded with the inputs and Q is reset to zero. In every clock cycle register A is shifted right and register B is shifted left. The least significant bit (LSB) of register A is the control input for the multiplexer. If it is high, the multiplexer forwards the value of register B to the adder, which adds it to the intermediate result stored in register Q. After four clock cycles register Q holds the product $A * B$.

Suppose that the multiplier has a fault in the adder: The output of the single-bit full adder responsible for bit 0 always adds 1 to the correct output.

The components we use for fault localization are the eight full adders in the adder, the eight AND gates in the multiplexer, and the 8-bit registers A, B, and Q. As property we state that after four clock cycles the output must be equal to $A * B$.

Correction approach

The correction approach is able to find the faulty part in the adder and provides a correction for all possible inputs. It suggests to use a half adder for bit 0. This is simpler than the correction we expected and still correct: In the first time step, Q is 0 and in all subsequent steps, the LSB of B is 0 because B is shifted left. Thus, a carry never occurs.

This example shows the advantage of the correction approach. We take all possible inputs into account and therefore we find the fault and provide a correction for it.

Localization approach

For the localization approach we use one counterexample. If we load A and B with 6 and 9, respectively, the output is 58 instead of 54. With consistency-based

diagnosis we find six components out of 40 as possible fault candidates: bit two in the registers B and Q, the AND gate for bit two in the multiplexer and the full adders for the three least significant bits.

We can reduce the number of diagnoses by using multiple test cases and computing the intersection of the reported diagnoses. However, the full adder for bit one is a candidate in every test case. To see this, note that after four time slices the computed result is the correct value plus four. Regardless of the inputs, the carry bit of the full adder for bit 1 will have value 1 in at least one time step. If we change this value to 0, the calculated result of the multiplication is reduced by four and we obtain the correct result. This example shows once more that consistency-based diagnosis finds candidates that cannot have caused the fault.

Although the approach is not able to minimize the number of faults to one, it reduces the components to be considered faulty to a fraction in a quite efficient way.

5 Conclusion

We have shown two methods to automatically locate faults in sequential systems, assuming that the specification is given in PSL.

The correction approach finds faults and automatically suggests corrections for the user's review. This is a very powerful method that is complete for invariants. For other properties, it may not find all possible faults, but experience has shown it that the heuristics we have implemented work very well. Efficiency is a problem for some designs: the complexity of the method is comparable to that of BDD-based model checking, and may not work for larger circuits.

The localization approach specializes on safety properties. It is based on SAT solving and is more efficient than the first. It is not as precise, that is, it may find more fault locations than necessary, but experience shows that it reduces the amount of code to be considered considerably.

We expect that both methods will be very valuable to designers and verification engineers, because it can significantly reduce the time between the discovery that a fault exists and the implementation of the fix.

Previous work has only been able to address a very small set of faults (forgotten or extraneous inverter, and/or gate switched, etc.), which makes it impractical for fresh designs. Furthermore previous work has not been able to locate faults that were detected with temporal properties. Existing work has rather assumed the existence of a reference model, which in our setting does not exist.

6 References

- [1] F. Aloul, A. Ramani, I. Markov, and K. Sakallah. PBS: A backtrack search pseudo Boolean solver. In *Symposium on the theory an applications of satisfiability testing (SAT)*, pages 346–353, 2002.
- [2] T. Ball, M. Naik, and S. K. Rajamani. From symptom to cause: Localizing errors in counterexample traces. In *30th Symposium on Principles of Programming Languages (POPL 2003)*, pages 97–105, January 2003.
- [3] P.-Y. Chung, Y.-M. Wang, and I. N. Hajj. Logic design error diagnosis and correction. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 2:320–332, 1994.
- [4] E. M. Clarke, O. Grumberg, and D. A. Peled. *Model Checking*. MIT Press, Cambridge, MA, 1999.
- [5] L. Console, G. Friedrich, and D. Theseider Dupré. Model-based diagnosis meets error diagnosis in logic programs. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI'93)*, pages 1494–1499. Morgan-Kaufmann, 1993.
- [6] L. Console and P. Torasso. A spectrum of logical definitions of model-based diagnosis. *Computational Intelligence*, 7(3):133–141, 1991.
- [7] G. Friedrich, M. Stumptner, and F. Wotawa. Model-based diagnosis of hardware designs. In *European Conference on Artificial Intelligence*, pages 491–495, 1996.
- [8] D. Gabbay, A. Pnueli, S. Shelah, and J. Stavi. On the temporal analysis of fairness. In *Proceedings of the Symposium on Principles of Programming Languages*, pages 163–173, 1980.
- [9] A. Groce. Error explanation with distance metrics. In *International Conference on Tools and Algorithms for Construction and Analysis of Systems (TACAS'04)*, pages 108–122, Barcelona, Spain, March-April 2004. LNCS 2988.
- [10] A. Groce and W. Visser. What went wrong: Explaining counterexamples. In *Model Checking of Software: 10th International SPIN Workshop*, pages 121–135. Springer-Verlag, May 2003. LNCS 2648.
- [11] W. Hamscher and R. Davis. Diagnosing circuits with state: An inherently underconstrained problem. In *Proceedings of the Fourth National Conference on Artificial Intelligence (AAAI-84)*, pages 142–147, Austin, TX, 1984.
- [12] H. Jin, K. Ravi, and F. Somenzi. Fate and free will in error traces. In *International Conference on Tools and Algorithms for Construction and Analysis of Systems (TACAS'02)*, pages 445–459, Grenoble, France, April 2002. LNCS 2280.
- [13] B. Jobstmann, A. Griesmayer, and R. Bloem. Program repair as a game. To appear at Computer Aided Verification '05, retrieve from www.ist.tugraz.at/verify/view/Projects/ProgramRepair, 2005.
- [14] J. de Kleer and B. C. Williams. Diagnosing multiple faults. *Artificial Intelligence*, 32:97–130, 1987.
- [15] H.-T. Liaw, J.-H. Tsiah, and I. N. Hajj. Efficient automatic diagnosis of digital circuits. In *Proceedings of the IEEE International Conference on Computer Aided Design*, pages 464–467, 1990.
- [16] J. C. Madre, O. Coudert, and J. P. Billon. Automating the diagnosis and the rectification of design error with PRIAM. In *Proceedings of the IEEE International Conference on Computer Aided Design*, pages 30–33, 1989.
- [17] Z. Manna and A. Pnueli. *The Temporal Logic of Reactive and Concurrent Systems *Specification**. Springer-Verlag, 1991.

- [18] C. Mateis, M. Stumptner, and F. Wotawa. A value-based diagnosis model for Java programs. In *Proceedings of the Eleventh International Workshop on Principles of Diagnosis*, 2000.
- [19] D. L. Poole, R. Goebel, and R. Aleliunas. Theorist: a logical reasoning system for defaults and diagnosis. In N. Cercone and G. McCalla, editors, *The Knowledge Frontier: Essays in the Representation of Knowledge*, pages 331–352. Springer Verlag, 1987.
- [20] K. Ravi and F. Somenzi. Minimal assignments for bounded model checking. In *International Conference on Tools and Algorithms for Construction and Analysis of Systems (TACAS'04)*, pages 31–45, Barcelona, Spain, March-April 2004. LNCS 2988.
- [21] R. Reiter. A theory of diagnosis from first principles. *Artificial Intelligence*, 32:57–95, 1987.
- [22] M. Renieris and S. P. Reiss. Fault localization with nearest neighbor queries. In *International Conference on Automated Software Engineering*, pages 30–39, Montreal, Canada, October 2003.
- [23] M. Stumptner and F. Wotawa. A model-based approach to software debugging. In *Proceedings on the Seventh International Workshop on Principles of Diagnosis*, 1996.
- [24] M. Stumptner and F. Wotawa. Debugging functional programs. In *Proceedings on the 16th International Joint Conference on Artificial Intelligence*, 1999.
- [25] M. Tomita, T. Yamamoto, F. Sumikawa, and K. Hirano. Rectification of multiple logic design errors in multiple output circuits. In *Proceedings of the Design Automation Conference*, pages 212–217, 1994.
- [26] M. Vardi and P. Wolper. Reasoning about infinite computations. *Information and Computation*, 115:1–37, 1994.
- [27] A. Wahba and D. Borrione. Design error diagnosis in sequential circuits. In *Correct Hardware Design and Verification Methods (CHARME'95)*, pages 171–188, 1995.
- [28] A. Zeller. Isolating cause-effect chains from computer programs. In *10th International Symposium on the Foundations of Software Engineering (FSE-10)*, pages 1–10, November 2002.
- [29] A. Zeller and R. Hildebrandt. Simplifying and isolating failure-inducing input. *IEEE Transactions on Software Engineering*, 28(2):183–200, February 2002.