Efficient Automatic STE Refinement Using Responsibility

Hana Chockler[1] and Orna Grumberg[2] and Avi Yadgar[2]

[1] IBM Research
Mount Carmel, Haifa 31905, Israel.
hanac@il.ibm.com

[2] Computer Science Department
Technion, Haifa, Israel.
orna,yadgar@cs.technion.ac.il

Abstract. Symbolic Trajectory Evaluation (STE) is a powerful technique for hardware model checking. It is based on 3-valued symbolic simulation, using 0,1, and X ("unknown"). X is used to abstract away values of circuit nodes, thus reducing memory and runtime of STE runs. The abstraction is derived from a given user specification.
An STE run results in "*pass*" (1), if the circuit satisfies the specification, "*fail*" (0) if the circuit falsifies it, and "*unknown*" (X), if the abstraction is too coarse to determine either of the two. In the latter case, refinement is needed: The X values of some of the abstracted inputs should be replaced. The main difficulty is to choose an appropriate subset of these inputs that will help to eliminate the "*unknown*" STE result, while avoiding an unnecessary increase in memory and runtime. The common approach to this problem is to manually choose these inputs.
This work suggests a novel approach to automatic refinement for STE, which is based on the notion of *responsibility*. For each input with X value we compute its *Degree of Responsibility* (DoR) to the "*unknown*" STE result. We then refine those inputs whose DoR is maximal.
We implemented an efficient algorithm, which is linear in the size of the circuit, for computing the approximate DoR of inputs. We used it for refinements for STE on several circuits and specifications. Our experimental results show that DoR is a very useful device for choosing inputs for refinement. In comparison with previous works on automatic refinement, our computation of the refinement set is faster, STE needs fewer refinement iterations and uses less overall memory and time.

1 Introduction

Symbolic Trajectory Evaluation (STE) [13] is a powerful technique for hardware model checking. STE is based on combining 3-valued abstraction with symbolic simulation. It is applied to a circuit M, described as a graph over *nodes* (gates and latches). The specification consists of assertions in a restricted temporal language. An assertions is of the form $A \Longrightarrow C$, where the *antecedent* A expresses constraints on nodes n at different times t, and the *consequent* C expresses requirements that should hold on such nodes (n, t). Abstraction in STE is derived from the specification by initializing all inputs not appearing in A to the X ("unknown") value.

An STE run may result in "*pass*" (1), if the circuit satisfies the specification, "*fail*" (0) if the circuit falsifies it, and "*unknown*" (X), if the abstraction is too coarse to determine either of the two. In the latter case, a refinement is needed: The X values of some of the abstracted inputs should be changed.

The main challenge in this setting is to choose an appropriate subset of these inputs, that will help to eliminate the "*unknown*" STE result. Selecting a "right" set of inputs for refinement is crucial for the success of STE: refining too many inputs may result in memory and time explosion. On the other hand, selecting too few inputs or selecting inputs that do not affect the result of the verification will lead to many iterations with an "*unknown*" STE result.

The common approach to this problem is to manually choose the inputs for refinement. This, however, might be labor-intensive and error-prone. Thus, an automatic refinement is desired.

In this work we suggest a novel approach to automatic refinement for STE, which is based on the notion of *responsibility*. For each input with X value we compute its *Degree of Responsibility* (DoR) to the "*unknown*" STE result. We then refine those inputs whose DoR is maximal.

To understand the notion of responsibility, consider first the following concepts. We say that event B *counterfactually depends* on event A [9] if A and B both hold, and had A not happened then B would not have happened. Halpern and Pearl broadened the notion of causality saying that A is a *cause* of B if there exists some change of the current situation that creates the counterfactual dependence between A and B [8].

As an example, consider the circuit in Figure 1(a). The event "$n = 1$" counterfactually depends on the event "$n_1 = 1$". Next consider the circuit in Figure 1(b). "$n_1 = 0$" is a cause of "$n = 0$". This is because if we change n_2 from 0 to 1, then "$n = 0$" counterfactually depends on "$n_1 = 0$". Similarly, "$n_2 = 0$ is a cause of "$n = 0$".

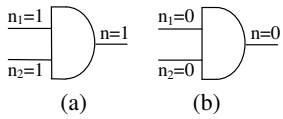


Fig. 1. Cause

The notion of responsibility and of weighted responsibility, introduced in [4], quantifies the change that is needed in order to create the counterfactual dependence. The DoR of A for B is taken to be $1/(k+1)$, where k is the size of the minimal change that creates the counterfactual dependence. For instance, in the example above, the DoR of "$n_1 = 0$" for "$n = 0$" is $1/2$, because the minimal change that creates a counterfactual dependence is of size 1 (changing the value of n_2 from 0 to 1). In this work we use weighted DoR in order to obtain a finer-grain quantification for changes, in the context of STE.

Computing responsibility in circuits is known to be intractable in general [4]. Inspired by the algorithm for read-once formulas in [3], we developed the algorithm $RespSTE$ for efficiently computing an *approximate* DoR. Computing the responsibility of the inputs for some output of a circuit involves one traversal of the circuit for each X valued input in the *cone of influence* of the output. The overall complexity is therefore only quadratic in the size of the circuit.

In order to evaluate our algorithm $RespSTE$, we implemented it and used it in conjunction with *Forte*, a BDD based STE tool by Intel [14]. We applied it to several circuits and specifications. We compared our results with the automatic refinement for STE, suggested in [15]. In all cases, the comparison shows a significant speedup. A significant reduction in BDD nodes is also gained in most of the assertions. In some of the cases, our algorithm needed fewer refinement iterations.

The DoRs we compute gives us a quantitative measure of the importance of each input to the STE "*unknown*" result. By examining these values, we conclude that this quantitative measure is of high quality and is complying with our understanding of the problem as users who are familiar with the models.

When using these results for automatic refinement, the quality of the results is reflected by the number of refinement iterations that were required, and in the number of symbolic variables that were added to the assertion. We point out that even when a non-automatic (manual) refinement is applied, our DoRs can serve as recommended priorities on the candidate inputs for refinement.

Related Work Abstraction-Refinement takes a major role in model checking [6, 10] for reducing the state explosion problem. In [5], it is shown that the abstraction in STE is an abstract interpretation via a Galois connection. In [17], an automatic abstraction-refinement for symbolic simulation is suggested. However, the first automatic refinement for STE has been suggested in [15]. In this refinement scheme, the values of the circuit nodes, as computed by STE, are used in order to trace X paths, and refine the STE assertion by adding symbolic variables to A. While this work is the closest to ours, it is essentially different from using the responsibility concept. We compare our results to this work in Section 5. In [2], an automatic refinement for GSTE is suggested. This method, like [15], traverses the circuit nodes after running STE, and performs a model and an assertion refinement. This method is also essentially different from ours, as it is aimed at solving GSTE problems, where an assertion graph describes the specification, and is used in the refinement process.

SAT based refinements were suggested in [12] and [7]. The method presented in [12] is used for assisting manual refinement. The method presented in [7] takes an automatic CEGAR approach which is applicable only in the suggested SAT based framework. In [1], a method for automatic *abstraction* without refinement is suggested. We believe that our algorithm can complement such a framework.

The rest of the paper is organized as follows. In Section 2 we give the needed background for STE and present the formal definitions of causality and responsibility. Section 3 shows how to define and compute the degrees of responsibility (DoR) in the context of STE refinement. Section 4 describes the abstraction and refinement loop for STE with responsibility. Section 5 presents our experiments and concludes with an evaluation of the results.

2 Preliminaries

2.1 Symbolic Trajectory Evaluation (STE)

A hardware model M is a *circuit*, represented by a directed graph. The graph's nodes $\mathcal{N}$ are input and internal nodes, where internal nodes are latches and combinational gates. A combinational gate represents a Boolean operator. The graph of M may contain cycles, but not combinational cycles. A graph of a circuit is shown in Figure 4(a). Given a directed edge (n_1, n_2), we say that n_1 is an *input* of n_2. We denote by (n, t) the value of node n at time t. The value of a gate (n, t) is the result of applying its operator on the inputs of n at time t. The value of a latch (n, t) is determined by the value of its input

at time $t-1$. The *bounded cone of influence* (BCOI) of a node (n,t) contains all nodes (n',t') with $t' \leq t$ that may influence the value of (n,t), and is defined recursively as follows: the BCOI of a combinational node at time t is the union of the BCOI of its inputs at time t, and the BCOI of a latch at time t is the union of the BCOI of its inputs at time $t-1$. The BCOI of a node with no inputs is the empty set.

In STE, a node can get a value in a quaternary domain $\mathcal{Q} = \{0, 1, X, \bot\}$. A node whose value cannot be determined by its inputs is given the value X("unknown"). $\bot$ is used to describe an over constrained node. This might occur when there is a contradiction between an external assumption on the circuit and its actual behavior.

A *state* s in M is an assignment of values from $\mathcal{Q}$ to every node, $s : \mathcal{N} \rightarrow \mathcal{Q}$. A *trajectory* π is an infinite series of states, describing a run of M. We denote by $\pi(i), i \in \mathbb{N}$, the state at time i in π, and by $\pi(i)(n), i \in \mathbb{N}, n \in \mathcal{N}$, the value of node n in the state $\pi(i)$. $\pi^i, i \in \mathbb{N}$, denotes the suffix of π starting at time i.

Let $\mathcal{V}$ be a set of *symbolic Boolean variables* over the domain $\{0,1\}$. A *symbolic expression* over $\mathcal{V}$ is an expression consisting of quaternary operations, applied to $\mathcal{V} \cup \mathcal{Q}$. The truth tables of the quaternary operators are given in Figure 2.

AND	X	0	1	$\bot$
X	X	0	X	$\bot$
0	0	0	0	$\bot$
1	X	0	1	$\bot$
$\bot$	$\bot$	$\bot$	$\bot$	$\bot$

OR	X	0	1	$\bot$
X	X	X	1	$\bot$
0	X	0	1	$\bot$
1	1	1	1	$\bot$
$\bot$	$\bot$	$\bot$	$\bot$	$\bot$

NOT	
X	X
0	1
1	0
$\bot$	$\bot$

Fig. 2. Quaternary Operations

A *symbolic state* over $\mathcal{V}$ is a mapping from each node of M to a symbolic expression. A symbolic state represents a set of states, one for each assignment to $\mathcal{V}$. A *symbolic trajectory* over $\mathcal{V}$ is an infinite series of symbolic states, compatible with the circuit. It represents a set of trajectories, one for each assignment to $\mathcal{V}$. Given a symbolic trajectory π and an assignment ϕ to $\mathcal{V}$, $\phi(\pi)$ denotes the trajectory that is received by applying ϕ to all the symbolic expressions in π.

A *Trajectory Evaluation Logic* (TEL) formula is defined recursively over $\mathcal{V}$ as follows: $f ::= n \text{ is } p \mid f_1 \wedge f_2 \mid p \rightarrow f \mid \mathbf{N}f$, where $n \in \mathcal{N}$, p is a Boolean expression over $\mathcal{V}$, and $\mathbf{N}$ is the next time operator. The *maximal depth* of a TEL formula f is the maximal time t for which a constraint exists in f on some node n, plus 1.

Given a TEL formula f over $\mathcal{V}$, a symbolic trajectory π over $\mathcal{V}$, and an assignment ϕ to $\mathcal{V}$, we define the satisfaction of f as in [15]:
$[\phi, \pi \models f] = \bot \;\leftrightarrow\; \exists i \geq 0, n \in \mathcal{N} : \phi(\pi)(i)(n) = \bot$. Otherwise:
$[\phi, \pi \models n \text{ is } p] = 1 \;\leftrightarrow \phi(\pi)(0)(n) = \phi(p)$
$[\phi, \pi \models n \text{ is } p] = 0 \;\leftrightarrow\; \phi(\pi)(0)(n) \neq \phi(p)$ and $\phi(\pi)(0)(n) \in \{0,1\}$
$[\phi, \pi \models n \text{ is } p] = X \;\leftrightarrow\; \phi(\pi)(0)(n) = X$ $\qquad \phi, \pi \models p \rightarrow f \equiv \neg\phi(p) \vee \phi, \pi \models f$
$\phi, \pi \models f_1 \wedge f_2 \equiv (\phi, \pi \models f_1 \wedge \phi, \pi \models f_2)$ $\qquad \phi, \pi \models \mathbf{N}f \equiv \phi, \pi^1 \models f$
Note that given an assignment ϕ to $\mathcal{V}$, $\phi(p)$ is a constant (0 or 1).
We define the truth value of $\pi \models f$ as follows:
$[\pi \models f] = 0 \;\leftrightarrow\; \exists \phi : [\phi, \pi \models f] = 0$
$[\pi \models f] = X \;\leftrightarrow\; \forall \phi : [\phi, \pi \models f] \neq 0$ and $\exists \phi : [\phi, \pi \models f] = X$
$[\pi \models f] = 1 \;\leftrightarrow\; \forall \phi : [\phi, \pi \models f] \notin \{0, X\}$ and $\exists \phi : [\phi, \pi \models f] = 1$
$[\pi \models f] = \bot \;\leftrightarrow\; \forall \phi : [\phi, \pi \models f] = \bot$

This definition creates levels of importance between 0 and X. If there exists an assignment such that $[\phi, \pi \models f] = 0$, the truth value of $\pi \models f$ is 0, even if there are other assignments such that $[\phi, \pi \models f] = X$.

STE assertions are of the form $A \Rightarrow C$, where A (the antecedent) and C (the consequent) are TEL formulae. A expresses constraints on circuit nodes at specific times,

and C expresses requirements that should hold on circuit nodes at specific times. We define the truth value of $[M \models A \Rightarrow C]$ as follows:

$[M \models A \Rightarrow C] = 1 \leftrightarrow \forall \pi : [\pi \models A] = 1$ implies $[\pi \models C] = 1$

$[M \models A \Rightarrow C] = \bot \leftrightarrow \forall \pi : [\pi \models A] = \bot$

$[M \models A \Rightarrow C] = 0 \leftrightarrow \exists \pi : [\pi \models A] = 1$ and $[\pi \models C] = 0$

$[M \models A \Rightarrow C] = X \leftrightarrow [M \models A \Rightarrow C] \neq 0$ and $\exists \pi : [\pi \models A] = 1$ and $[\pi \models C] = X$

t	n_1	n_2	n_3	n_4	n_5	n_6
0	X	X	X	0	X	0
1	X	X	0	X	1	X

Fig. 3. Symbolic Simulation

As in [15], an *antecedent failure* is the case where $[M \models A \Rightarrow C] = \bot$. For a node n at time t we say that "(n, t) is X-possible", if there exists a trajectory π and an assignment ϕ such that $\phi(\pi)(t)(n)$ is X. If n at time t is also constrained by C, then we say that it is *undecided*. In that case, $[M \models A \Rightarrow C] = X$. Consider the circuit and STE assertion in Figure 4(a). The table in Figure 3 corresponds to a symbolic simulation of this assertion. n_5 at time 1 is evaluated to 1, and thus the assertion holds.

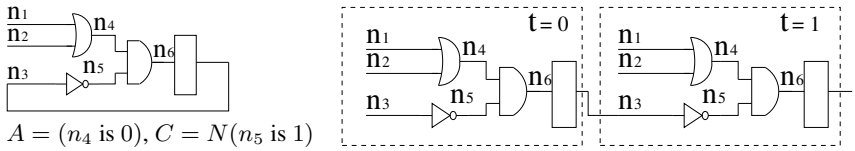


Fig. 4. (a) A circuit M (b) An Unrolling of M to depth 2

2.2 Refinement in STE

A major strength of STE is the use of abstraction. The abstraction is determined by the assignment of the value X to input nodes in M by A. However, if the abstraction is too coarse, then there is not enough information for proving or falsifying the STE assertion. That is, $[M \models A \Rightarrow C] = X$.

The common abstraction and refinement process in STE consists of the following steps: the user writes an STE assertion $A \Rightarrow C$ for M, and receives a result from STE. If $[M \models A \Rightarrow C] = \bot$ (an antecedent failure), then there is a contradiction between A and M, and the user has to write a new assertion. If $[M \models A \Rightarrow C] = 0$, or $[M \models A \Rightarrow C] = 1$, the process ends with the corresponding result. If $[M \models A \Rightarrow C] = X$, a refinement is required. In this case, there is some *X-possible* node (n, t), which is undecided. The user has to manually decide how to refine the specification such that the X truth value will be eliminated.

For automatic refinement, we assume that the STE assertion correctly describes the desired behavior of the model, and that disagreements between the assertion and the model originate from errors in the model. Thus, the refinement should preserve the meaning of the original assertion. Note, that refinement is only performed in cases where an antecedent failure does not occur.

An automatic refinement can be obtained by creating a new antecedent for the STE assertion. The refinement of A should preserve the semantics of $A \Rightarrow C$. Formally, let $A_{new} \Rightarrow C$ denote the refined assertion and let $runs(M)$ denote the set of all concrete trajectories of M. We require that $A_{new} \Rightarrow C$ holds on $runs(M)$ iff $A \Rightarrow C$ holds on $runs(M)$.

Refinement Strategy In [15], refinement steps add constraints to A by forcing the values of some input nodes at certain times to the value of *fresh symbolic variables*. That is, symbolic variables that are not already in $\mathcal{V}$. By initializing an input (in, t) with a fresh symbolic variable instead of X, the value of (in, t) is accurately represented, and knowledge about its effect on M is added. However, it does not constrain input behavior that was allowed by A, nor does it allow input behavior that was forbidden by A. Thus, the semantics of A is preserved. In [15] it is proven that if $A_{new} \Rightarrow C$ holds in M, then so does $A \Rightarrow C$. Also, if $A_{new} \Rightarrow C$ yields a counterexample ce, then ce is also a counterexample w.r.t $A \Rightarrow C$.

2.3 Causality and Responsibility

In this section, we review the definitions of causality and responsibility. We start with causality. The most intuitive definition of causality is *counterfactual causality*, going back to Hume [9],which is formally defined as follows.

Definition 1 (Counterfactual causality). *We say that an event A is a* counterfactual cause *of event B if the following conditions hold: (a) both A and B are true, and (b) if we assign A the value false, then B becomes false. We sometimes refer to the dependence of B on A as a* counterfactual dependence.

In this paper, we use a simplified version of the definition of causality from [8]. In order to define causality formally, we start with the definition of causal models (again, due to [8]).

Definition 2 (Causal model). *A* causal model *M is a tuple $\langle \mathcal{U}, \mathcal{D}, \mathcal{R}, \mathcal{F} \rangle$, where $\mathcal{U}$ is the set of* exogenous *variables (that is, variables whose value is determined by constraints outside of the model), $\mathcal{D}$ is the set of* endogenous *variables (that is, variables whose value is determined by the model and the current assignment to $\mathcal{D}$), $\mathcal{R}$ associates with each variable in $\mathcal{U} \cup \mathcal{D}$ a nonempty range of values, and the function $\mathcal{F}$ associates with every variable $Y \in \mathcal{D}$ a function F_Y that describes how the value of Y is determined by the values of all other variables in $\mathcal{U} \cup \mathcal{D}$.*

A context *$\vec{d}$ is an assignment for variables in $\mathcal{D}$ (the values of variables in $\mathcal{U}$ are considered constant).*

In this paper we restrict our attention to models in which variables do not depend on each other.

A *causal formula* φ is a formula over the set of variables $\mathcal{U} \cup \mathcal{D}$. A causal formula φ is true or false in a causal model given a context $\vec{d}$. We write $(M, \vec{d}) \models \varphi$ if φ is true in M given a context $\vec{d}$. We write $(M, \vec{d}) \models [\vec{Z} \leftarrow \vec{z}]\varphi$ if φ holds in the model M given the context $\vec{d}$ and the assignment $\vec{z}$ to the variables in the set $\vec{Z} \subset \mathcal{V}$, such that $\vec{z}$ overrides $\vec{d}$ for variables in $\vec{Z}$.

With these definitions in hand, we can give the simplified definition of cause based on the definition in [8]. The main simplification is due to the fact that in our models, variables do not depend on each other, and thus there is no need to explicitly check various cases of mutual dependence between variables.

Definition 3 (Cause). *For a constant y, we say that $Y = y$ is a* cause *of φ in $(M, \vec{d})$ if the following conditions hold:*

AC1. $(M, \vec{d}) \models (Y = y) \wedge \varphi$.

AC2. *There exists a partition* $(\vec{Z}, \vec{W})$ *of* $\mathcal{D}$ *with* $Y \in \vec{Z}$ *and some setting* $(y', \vec{w}')$ *of the variables in* $Y \cup \vec{W}$ *such that:*

(a) $(M, \vec{d}) \models [Y \leftarrow y', \vec{W} \leftarrow \vec{w}']\neg\varphi$. *That is, changing* $(Y, \vec{W})$ *from their original assignment* $(y, \vec{w})$ *(where* $\vec{w} \subset \vec{d}$*) to* $(y', \vec{w}')$ *changes* φ *from* **true** *to* **false**.

(b) $(M, \vec{d}) \models [Y \leftarrow y, \vec{W} \leftarrow \vec{w}']\varphi$. *That is, setting* $\vec{W}$ *to* $\vec{w}'$ *should have no effect on* φ *as long as* Y *has the value* y.

Essentially, Definition 3 says that $Y = y$ is a cause of φ if both $Y = y$ and φ hold in the current context $\vec{d}$, and there exists a change in $\vec{d}$ that creates a counterfactual dependence between $Y = y$ and φ.

The definition of responsibility introduced in [4] refines the "all-or-nothing" concept of causality by measuring the degree of responsibility of $Y = y$ for the truth value of φ in $(M, \vec{d})$. The following definition is due to [4]:

Definition 4 (Responsibility). *Let* k *be the smallest size of* $\vec{W} \subset \mathcal{D}$ *such that* $\vec{W}$ *satisfies the condition* **AC2** *in Definition 3. Then, the* degree of responsibility (DoR) *of* $Y = y$ *for the value of* φ *in* $(M, \vec{d})$*, denoted* $dr((M, \vec{d}), Y = y, \varphi)$*, is* $1/(k + 1)$.

Thus, the degree of responsibility measures the minimal number of changes that have to be made in $\vec{d}$ in order to make $Y = y$ a counterfactual cause of φ. If $Y = y$ is not a cause of φ in $(M, \vec{d})$, then the minimal set $\vec{W}$ in Definition 4 is taken to have cardinality ∞, and thus the degree of responsibility of $Y = y$ is 0. If φ counterfactually depends on $Y = y$, its degree of responsibility is 1. In other cases the degree of responsibility is strictly between 0 and 1. Note that $Y = y$ is a cause of φ iff the degree of responsibility of $Y = y$ for the value of φ is greater than 0.

As we argue in Section 3.1, in our setting it is reasonable to attribute weights to the variables in order to capture the cost of changing their value. Thus, we use the *weighted* version of the definition of the degree of responsibility, also introduced in [4]:

Definition 5 (Weighted responsibility). *Let* $wt(Y)$ *be the weight of* Y *and* $wt(\vec{W})$ *the sum of the weights of variables in the set* $\vec{W}$*. Then, the* weighted degree of responsibility *of* $Y = y$ *for* φ *is* $wt(Y)/(k + wt(Y))$*, where* k *is the minimal* $wt(\vec{W})$ *of a* $\vec{W} \subset \mathcal{D}$ *for which AC2 holds. This definition agrees with Definition 4 if the weights of all variables are* 1.

Remark 1. We note that in general, there is no connection between the degree of responsibility of $Y = y$ for the value of φ and a probability that φ counterfactually depends on $Y = y$. Basically, responsibility is concerned with the minimal number of changes in a given context that creates a counterfactual dependence, whereas probability is measured over the space of all possible assignments to variables in $\mathcal{D}$.

3 Responsibility in STE Graphs

In section 4 we will show how to refine STE assertions by using the degree of responsibility (*dr*) of inputs for *X-possible* nodes. Consider a model circuit M, and an STE assertion $A \Rightarrow C$, such that $[M \models A \Rightarrow C] = X$ and let r be an undecided node. In this section we show how M can be viewed as a causal model, and present an algorithm for computing the degree of responsibility of an input to M for "r is *X-possible*".

3.1 STE circuits as causal models

In order to verify the assertion $A \Rightarrow C$, M has to be simulated k times, where k is the *maximal depth* of A and C. We create a graph by unrolling M k times. Each node $n \in M$ has k instances in the new graph. The ith instance of node n represents node n at time i. In the new graph, the connectivity of the input and gate nodes remains the same. The latches are connected such that the input to a latch at time t are the nodes at time $t-1$, and the latch is an input to nodes at time t. Due to the new connectivity of the latches, and since M does not have combinational cycles, the unrolled graph is a *DAG*. The leaves of the new graph are k instances of each of the inputs to M, and the initial values of the latches.

We assume that the only nodes assigned by A are leaves. It is straightforward to extend the discussion to internal nodes that are assigned by A, and to nodes that get their value from propagating the assignments of A. Consider the circuit and STE assertion in Figure 4(a). The corresponding unrolling is shown in Figure 4(b). $t = 0$ and $t = 1$ are two instances of the circuit. The inputs to the latch n_3 in $t = 1$ are the nodes of $t = 0$, thus eliminating the cycle of the original circuit. The inputs to the new circuit are the first instance of n_3 (the initial value of the latch), and the two instances of n_1 and n_2. From herein we denote by M the unrolled graph of the circuit.

Regarding M as a *causal model* requires the following definitions: 1) a set of variables and their partition into $\mathcal{U}$ and $\mathcal{D}$, the exogenous and endogenous variables, respectively. 2) $\mathcal{R}$, the range of the endogenous variables. 3) values for the exogenous variables $\mathcal{U}$. 4) a context $\vec{d}$, which is an assignment to the variables in $\mathcal{D}$. 5) $\mathcal{F}$, a function which associates each variable $Y \in D$ with a function F_Y that describes its dependence in all the other variables.

We define the inputs of M to be the variables of the causal model. The inputs that are assigned 1 or 0 by the antecedent A are considered the exogenous variables $\mathcal{U}$, and their values are determined by A. The values of these variables cannot change, and are viewed as part of the model M. The rest of the inputs to M are the endogenous variables $\mathcal{D}$. The range of the variables in $\mathcal{D}$ is $\{0, 1, X\} \cup \mathcal{V}$, where $\mathcal{V}$ is the set of symbolic variables used by A.[1]The context $\vec{d}$ is the current assignment to $\mathcal{D}$, imposed by the antecedent A. Last, since the variables are inputs to a circuit, their values do not depend on each other. Therefore, the function F associates each variable with the identity function.

Next we have to define a causal formula φ. For an undecided node r, we want to compute the responsibility of the leaves having X values for "r is *X-possible*". We define the *causal formula* φ to be "r is *X-possible*". Since the context $\vec{d}$ is imposed by the antecedent A, and Since "r is *X-possible*" holds under A, we have $(M, \vec{d}) \models \varphi$.

We will compute a weighted degree of responsibility, as described in Definition 5. We choose $wt(n) = 1$ if $\vec{d}(n) \in \mathcal{V}$, and $wt(n) = 2$ if $\vec{d}(n) = X$. Next we explain this choice of weights. For computing the degree of responsibility, we consider changes in the context $\vec{d}$ that replace the assignments to some of the variables in $\mathcal{D}$ from X or $v_i \in \mathcal{V}$ to a Boolean value. When running STE, a symbolic variable may assume either of the Boolean values. On the other hand, a leaf that is assigned X cannot take

[1] For simplicity of presentation, we do not distinguish between a symbolic variable $v_i \in \mathcal{V}$ and its corresponding element in $\mathcal{R}$.

a Boolean value without changing the antecedent of the STE assertion. Therefore, we consider changing $\vec{d}$ for a variable n such that $\vec{d}(n) \in \mathcal{V}$ to be easier than for a variable n such that $\vec{d}(n) = X$. Thus, our choice of weights takes into account the way in which STE regards X and $v_i \in \mathcal{V}$.

We have shown how an unrolled model M can be viewed as a causal model. Let $I_X(r)$ and $I_V(r)$ be the sets of leaves in $BCOI(r)$, for which A assigns X and symbolic variables, respectively. From herein, for $l \in I_X(r)$, we denote by $dr(M, l, r)$ the degree of responsibility of "l is X" for "r is *X-possible*". Next we present an algorithm that computes an approximate degree of responsibility of each leaf in $I_X(r)$ for "r is *X-possible*".

3.2 Computing Degree of Responsibility in Trees

Computing responsibility in circuits is known to be $\mathrm{FP}^{\Sigma_2^P[\log n]}$-complete [2] in general [4], and thus intractable. In order to achieve an efficiently computable approximation, our algorithm is inspired by the algorithm for read-once formulas in [3]. It involves one traversal of the circuit for each $l \in I_X(r)$ and its overall complexity is only quadratic in the size of M. We start by describing an exact algorithm for M which is a tree, and then introduce the changes for M which is a DAG.

We define the following values that are used by our algorithm.

- $c_0(n, M)$: the minimal sum of weights of leaves in $I_X(r) \cup I_V(r)$ that we have to assign 0 or 1 in order to make n evaluate to 0.
- $c_1(n, M)$: the minimal sum of weights of leaves in $I_X(r) \cup I_V(r)$ that we have to assign 0 or 1 in order to make n evaluate to 1.
- $s(n, M, l)$: the minimal sum of weights of leaves in $I_X(r) \cup I_V(r)$ that we have to assign 0 or 1 in order to make "n is *X-possible*" counterfactually depend on "l is X". If there is no such number, that is, there is no change in the context that causes this dependability, we define $s(n, M, l) = \infty$.

If clear from the context, we omit M from the notation of c_0, c_1 and s.

We would like to compute the degree of responsibility of every leaf $l \in I_X(r)$ for "r is *X-possible*". Therefore, for each $l \in I_X(r)$, our algorithm computes $s(r, l)$. We denote by $A(n)$ the assignment to node n in M, imposed by A. We discuss a model M with AND and NOT operators. Extending the discussion to $OR, NAND$ and NOR operators is straightforward. Given r and $l \in I_X(r)$, our algorithm computes $s(r, l)$ by starting at r, and executing the recursive computation described next. Note that only values that are actually needed for determining $s(r, l)$ are computed.
For a node n, $s(n, l)$ is recursively computed by:

- For n a leaf: if $(n = l)$ then $s(n, l) = 0$, because the value of l counterfactually depends on itself. Otherwise, $s(n, l) = \infty$ since a leaf does not depend on other leaves.

[2] $\mathrm{FP}^{\Sigma_2^P[\log n]}$ is the class of functions computable in polynomial time with $\log n$ queries to oracle in Σ_2.

- For $n = n_1 \wedge \ldots \wedge n_m$: W.l.o.g. we assume that l belongs to the subtree of M rooted in n_1 (since M is a tree, l belongs to a subtree of only one input of n). In order to make the value of n counterfactually depend on the value of l, all input to n, except for n_1, should be 1, and the value of n_1 should counterfactually depend on the value of l. Thus, $s(n, l) = s(n_1, l) + \sum_{i=2}^{m} c_1(n_i)$.
- For $n = \neg n_1$: $s(n, l) = s(n_1, l)$, since "n is *X-possible*" iff "n_1 is *X-possible*".

For a node n, $c_0(n)$ and $c_1(n)$ are recursively computed by:

- For n a leaf:
  - If $A(n) = 0$, $c_0(n) = 0$, because no change in the assignments to $I_X(r) \cup I_V(r)$ is required. $c_1(n) = \infty$, because no change in the assignments to $I_X(r) \cup I_V(r)$ will change the value of n.
  - Similarly, if $A(n) = 1$, $c_1(n) = 0$ and $c_0(n) = \infty$.
  - If $A(n) = X$, $c_0(n) = c_1(n) = 2$, because only the value of n has to be changed, and the weight of n is 2.
  - If n is associated with a symbolic variable, $c_0(n) = c_1(n) = 1$, because only the value of n has to be changed, and the weight of n is 1.
- For $n = n_1, \wedge \ldots \wedge n_m$:
  - It is enough to change the value of one of its inputs to 0 in order to change the value of n to 0, thus $c_0(n) = \min_{i \in \{1,\ldots,n\}} c_0(n_i)$.
  - The values of all the inputs of n should be 1 in order for n to be 1, thus $c_1(n) = \sum_{i=1}^{n} c_1(n_i)$.
- For $n = \neg n_1$
  - $c_1(n) = c_0(n_1)$ and $c_0(n) = c_1(n_1)$, as any assignment that gives n_1 the value 0 or 1, gives n the value 1 or 0, respectively.

The computation above directly follows the definitions of c_0, c_1 and s, and thus its proof of correctness is straightforward. For a node r and leaf l, computing the values $c_0(n), c_1(n)$ and $s(n, l)$ for all $n \in BCOI(r)$ is linear in the size of M. Therefore, given r, computing $s(r, l)$ for all $l \in I_X(r)$ is at most quadratic in the size of M. Note that for a node n, the values $c_0(n)$ and $c_1(n)$ do not depend on a particular leaf, and thus are computed only once.

We demonstrate the computations done by our algorithm on the circuit in Figure 5. The antecedent associates l_2, l_4 and l_1, l_3 with symbolic variables and X, respectively. For node out, "out is *X-possible*" holds. We want to compute $s(out, l_3)$. l_3 is in the subtree of n_2. Therefore, $s(out, l_3) = c_1(n_1) + s(n_2, l_3)$. Since n_1 is an AND gate, $c_1(n_1) = c_1(l_1) + c_1(l_2)$. n_2 is also an AND gate, and therefore $s(n_2, l_3) = c_1(l_4) + s(l_3, l_3)$. The weight of the leaves is according to their assignment. Therefore, $c_1(l_2) = c_1(l_4) = 1$ and $c_1(l_1) = 2$. Additionally, $s(l_3, l_3) = 0$.

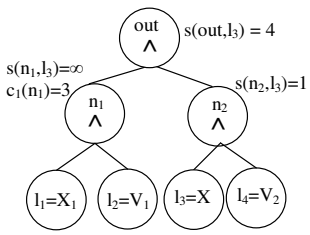


Fig. 5. Computing Responsibility

Finally, the degree of responsibility of "l is X" for "r is *X-possible*", $dr(M, l, r)$ is $\frac{2}{s(r,l)+2}$. If $s(r, l) = \infty$, then $dr(M, l, r) = 0$, which matches Definition 5. In our example, $dr(M, l_3, out) = \frac{2}{s(out,l_3)+2} = \frac{1}{3}$

3.3 Computing an Approximate Degree of Responsibility in DAGs

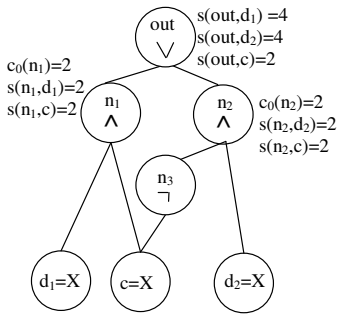
We now introduce a change to the definition of $s(n, l)$, resulting in an efficiently computable approximation of the degree of responsibility in DAGs, as required for STE.

For a DAG M, and a node $n = n_1 \wedge \ldots \wedge n_m$, we no longer assume that l belongs to a subtree of only one input of n. Let $N^S = \{n_i | s(n_i, l) \neq \infty, i \in \{1, \ldots, m\}\}$, and let $N^\infty = \{n_i | s(n_i, l) = \infty, i \in \{1, \ldots, m\}\}$. We define $s(n, l)$ to be:

$$s(n, l) = \sum_{n_i \in N^S} \frac{s(n_i, l)}{|N^S|} + \sum_{n_i \in N^\infty} c_1(n_i)$$

Recall that $dr(M, l, n)$ is inversely proportional to $s(n, l)$. Thus, our new definition gives higher degree of responsibility to leaves that belong to subtrees of multiple inputs of n. Such leaves are likely to be control signals, or otherwise more effective candidates for refinement than other variables.

We demonstrate the effect of this definition on the multiplexer in Figure 6. d_1 and d_2 are the data inputs to the multiplexer, and c is its control input. If $c = 1$, then $out = d_1$, else, $out = d_2$. The value $s(out, d_1)$ is given by $s(out, d_1) = c_0(n_2) + s(n_1, d_1) = 4$. The same computation applies to d_2. On the other hand, c belongs to the subtrees of both n_1 and n_2. Therefore, $s(out, c) = \frac{s(n_1,c)+s(n_2,c)}{2} = 2$. Consequently, $dr(c, out) = \frac{1}{2}$, whereas $dr(d_1, out) = dr(d_2, out) = \frac{1}{3}$.



$out = (c \wedge d_1) \vee (\neg c \wedge d_2)$

Fig. 6. A multiplexer.

The rest of the algorithm remains as in Section 3.2. Note that since M is a *DAG*, rather than a tree, not changing the computation of c_0 and c_1 makes it an approximation, as it does not take into account possible dependencies between inputs of nodes.

4 Applying Responsibility to Automatic Refinement

Refinement of an STE assertion is required when the return value of an STE run is X. In that case, the set of undecided nodes is returned by STE. The goal of the refinement is to add information such that undecided nodes become decided. In this section we show how we employ the concept of responsibility for efficiently refining STE assertions.

The outline of the refinement algorithm follows the discussion in section 2.2: First, a refinement goal r is selected from within the set of undecided nodes. Then, a set of input nodes in $I_X(r)$ is chosen, to be initialized to new symbolic variables.

Choosing a Refinement goal Our refinement algorithm chooses a single refinement goal on each refinement iteration. This way, the verification process might be stopped early if a constraint over a single node does not hold, without handling the other undecided nodes. Additionally, conceptual relations between the undecided nodes may make

them depend on a similar set of inputs. Thus, refinement targeted at one node may be useful for the other nodes as well. For example, all bits of a data vector are typically affected by the same set of control signals.

We would like to add as little symbolic variables as possible. Thus, from within the set of undecided nodes, we choose the nodes with the minimal number of inputs in its BCOI, and among these we choose the one with the minimal number of nodes in its BCOI. If multiple nodes have the minimal number of inputs in their BCOI, we arbitrarily pick one of them. This approach has also been taken in [15].

Choosing Input Nodes Given a refinement goal r, we have to choose a subset of nodes $I_{ref} \subseteq I_X(r)$ that will be initialized to new symbolic variables, trying to prevent the occurrence of "r is *X-possible*". We choose the nodes in $I_X(r)$ with the highest degree of responsibility for "r is *X-possible*", as computed by the algorithm in Section 3.3. These nodes have the most effect on "r is *X-possible*", and are likely to be the most effective nodes for refinement. Our experimental results support this choice of nodes, as shown in Section 5.

Given the refinement algorithm described above, we construct *RespSTE*, an iterative algorithm for verifying STE assertions: for a model M and an STE assertion $A \Rightarrow C$, while STE returns $[M \models A \Rightarrow C] = X$, RespSTE iteratively chooses a refinement root $r \in M$, computes the degree of responsibility of each leaf $l \in I_X(r)$ for "r is *X-possible*" and introduces new symbolic variables to A, for all leaves with the highest degree of responsibility. A pseudo code of RespSTE is given in Figure 7.

5 Experimental Results

For evaluating our algorithm $RespSTE$, presented in Section 4, we implemented and used it in conjunction with *Forte*, a BDD based STE tool by Intel [14].

For our experiments we used the Content Addressable Memory (*CAM*) module from Intel's GSTE tutorial, and IBM's Calculator 2 design [16]. These models and their specifications are interesting and challenging for model checking. All experiments use dedicated computers with 3.2Ghz Intel Pentium CPU, and 3GB RAM, running Linux operating system.

```
RespSTE(M, A, C)
while [M ⊨ A ⇒ C] = X do
   r ← choose refinement target
   for all l ∈ I_X(r) do
      compute dr(r, l)
   end for
   max ← max{dr(l, r)) | l ∈ I_X(r)}
   I_ref ← {l | l ∈ I_X(r), dr(l, r) = max}
   ∀ l_i ∈ I_ref, add symbolic variable v_{l_i} to A
end while
```

Fig. 7. RespSTE.

5.1 Verifying CAM Module

A CAM is a memory module that for each data entry holds a tag entry. Upon receiving an associative read (aread) command, the CAM samples the input "tagin". If a matching tag is found in the CAM, it gives the "hit" output signal the value 1, and outputs the corresponding data entry to "dout". Otherwise, "hit" is given the value 0. The verification

of the aread operation using STE is described in [11]. The CAM that we used is shown in Figure 8. It contains 16 entries. Each entry has a data size of 64 bits and a tag size of 8 bits. It contains 1152 latches, 83 inputs and 5064 combinational gates.

We checked the CAM against three assertions. For all the assertions, RespSTE added the smallest number of symbolic variables required for proving or falsifying the assertion. Next we discuss Assertion 1.

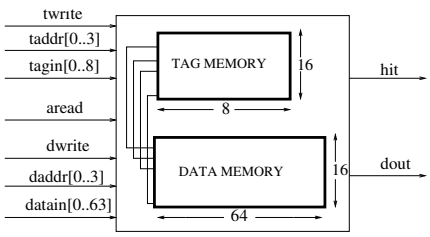


Fig. 8. Content Addressable Memory

Given $\overrightarrow{\text{TAG}}$ and $\overrightarrow{A}$, vectors of symbolic variables, Assertion 1 is: (tagin is $\overrightarrow{\text{TAG}}$) $\wedge$ (taddr is $\overrightarrow{A}$) $\wedge$ (twrite is 1) $\wedge$ **N** ((aread is 1) $\wedge$ (tagin is $\overrightarrow{\text{TAG}}$)) $\implies$ **N** (hit is 1). This is to check that if a tag value $\overrightarrow{\text{TAG}}$ is written to an address $\overrightarrow{A}$ in the tag memory at time 0, and at time 1 $\overrightarrow{\text{TAG}}$ is read, then it should be found in the tag memory, and hit should be 1. If at time 1 there is no write operation to the tag memory ($(twrite, 1) = 0$), then $\overrightarrow{\text{TAG}}$ should be found in address $\overrightarrow{A}$. If $(twrite, 1) = 1$, $\overrightarrow{\text{TAG}}$ should still be found, since it is written again to the tag memory. Therefore, Assertion 1 should pass. However, since twrite and taddr at time 1 are X, the CAM cannot determine whether to write the value of $(tagin, 1)$ to the tag memory, and to which tag entry to write it. As a result, the entire tag memory at time 1 is X, causing $(hit, 1)$ to be X. Thus, $[M \models A \Rightarrow C] = X$. In two consecutive refinement iterations, $(twrite, 1)$ and $(tadder, 1)$ are associated with new symbolic variables, and the assertion passes. The refinement steps of Assertion 1 are presented in Figure 10(a). Each row in the table describes a single refinement iteration, the name of the goal node, and the name and time of the inputs for which symbolic variables were added.

5.2 Verifying Calculator 2

Calculator 2 design [16], shown in Figure 9, is used as a case study design in simulation based verification. It contains 2781 latches, 157 inputs and 56960 combinational gates. The calculator has two internal arithmetic pipelines: one for add/sub and one for shifts. It receives commands from 4 different ports, and outputs the results accordingly. The calculator supports 4 types of commands: add, sub, shift right and shift left. The response is 1 for good, 2 for underflow, overflow or invalid command, 3 for an internal error and 0 for no response. When running the calculator, reset has to be 1 for the first 3 cycles.

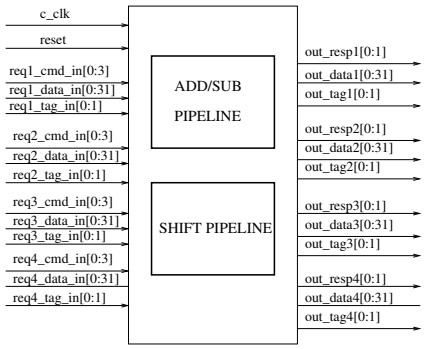


Fig. 9. Calculator

We checked the calculator against four assertions. For all but one of the assertions, RespSTE added the smallest number of symbolic variables required for proving or falsifying the assertion. Next we discuss Assertion 2.

Assertion 2 sets the command sent by a port P_i to add. The msb bits of the sent data are constrained to 0 to avoid an overflow. No constraints are imposed on the commands sent by other ports. The requirement is that the output data for P_i would match the expected data. Assertion 2 fails due to an erroneous specification. The calculator gives priority to the lower indexed ports. Thus, if both ports 1 and 3 send an add command, port 3 does not receive a response at the first possible cycle. Due to the implementation of the priority queue, commands of at least 3 ports have to be definite for falsifying the assertion. $I_X((out_resp2[0], 7))$ contains cmd, data and tag inputs of all ports at cycles 3 and 4. Out of them, RespSTE added the least number of inputs required for falsifying the assertion. The refinement steps of Assertion 2 are presented in Figure 10(b).

5.3 Evaluation of Results

In [15], an algorithm called *autoSTE* for automatic refinement in STE, is presented. *autoSTE* exploits the results of the STE run, as computed by *Forte*, in order to identify trajectories along which all nodes have the value X. The input nodes of these trajectories are the candidates for refinement. Heuristics are used for choosing subsets of these candidates.

As.	It	Goal	Added Vars
1	1	hit,1	twrite,1
1	2	hit,1	taddr [0:3], 1

(a) CAM

As.	It	Goal	Added Vars
2	1	out_resp2 [0],7	req1_cmd [0:2],3
2	2	out_resp2 [0],7	req1_cmd [3],3
2	3	out_resp2 [0],7	req2_cmd [0:3],3

(b) Calculator 2

Fig. 10. Refinement Steps.

We compared our experimental results with those obtained by *autoSTE*. For the sake of comparison, we used in our experiments the same parametric representation of the STE assertions as in [15]. The final results of RespSTE and its comparison with autoSTE are shown in Table 1.

The comparison shows a significant speedup in all of the assertions, and up to $18.5\times$ speedup in the larger ones. A significant reduction in BDD nodes is also gained in most of the assertions. For some of the assertions, RespSTE added less symbolic variables or required less refinement iterations than autoSTE. The overall performance of RespSTE was better than autoSTE even when this was not the case.

Altogether, our experiments demonstrated that using the degree of responsibility as a measure for refinement is a good choice. It provides a quantitative measure of the importance of each input to an undecided node being *X-possible*. By examining these values, we conclude that this quantitative measure reflects the actual importance of the inputs in the model. The results obtained by RespSTE agree with the decisions of a user who is familiar with the circuit. When using these results for automatic refinement, the quality of the results demonstrates itself in the number of refinement iterations that were required, and in the total number of symbolic variables that were added to the antecedent.

Acknowledgements We thank Rotem Oshman and Rachel Tzoref for the fruitful discussions, and the reviewers for their useful comments.

		RespSTE					AutoSTE			
		result	Iterations	Vars	BDD Nodes	Time	Iterations	Vars	BDD Nodes	Time
CAM	1	pass	2	5	3201	2	2	5	4768	3
	2	fail	5	11	30726	5	7	11	57424	20
	3	fail	1	8	14127	3	3	13	29006	17
Calc 2	1	fail	2	5	7735	32	2	2	6241	87
	2	fail	3	8	19717	25	2	8	20134	100
	3	fail	1	8	262201	43	1	8	530733	220
	4	pass	4	16	14005	27	11	16	17323	494

Table 1. Experimental Results. AutoSTE is the algorithm presented in [15]. "Iterations" is the number of refinement iterations that were performed, "Time" is the total runtime in seconds until verification / falsification of the property, "Vars" is the total number of symbolic variables that were added by the refinements, and "BDD Nodes" is the number of BDD nodes used by Forte.

References

1. Sara Adams, Magnus Bjork, Tom Melham, and Carl Seger. Automatic Abstraction in Symbolic Trajectory Evaluation . In *FMCAD '07*, 2007.
2. Yan Chen, Yujing He, Fei Xie, and Jin Yang. Automatic Abstraction Refinement for Generalized Symbolic Trajectory Evaluation. In *FMCAD '07*, 2007.
3. H. Chockler, J. Y. Halpern, and O. Kupferman. What causes a system to satisfy a specification? *ACM TOCL*. To appear.
4. H. Chockler and J.Y. Halpern. Responsibility and blame: a structural-model approach. *Journal of Artificial Intelligence Research (JAIR)*, 22:93–115, 2004.
5. C-T. Chou. The mathematical foundation of symbolic trajectory evaluation. In *CAV*, 1999.
6. E. M. Clarke, O. Grumberg, S. Jha, Y. Lu, and H. Veith. Counterexample-guided abstraction refinement. In *CAV*, 2000.
7. Orna Grumberg, Assaf Schuster, and Avi Yadgar. 3-Valued Circuit SAT for STE with Automatic Refinement . In *ATVA '07*, 2007.
8. J.Y. Halpern and J. Pearl. Causes and explanations: A structural-model approach — part 1: Causes. In *Uncertainty in Artificial Intelligence: Proceedings of the Seventeenth Conference (UAI-2001)*, pages 194–202, San Francisco, CA, 2001. Morgan Kaufmann Publishers.
9. D. Hume. *A treatise of human nature*. John Noon, London, 1739.
10. Robert P. Kurshan. *Computer-Aided Verification of coordinating processes - the automata theoretic approach*. 1994.
11. M. Pandey, R.Raimi, R. E. Bryant, and M. S. Abadir. Formal verification of content addressable memories using symbolic trajectory evaluation. *DAC*, 00:167, 1997.
12. Jan-Willem Roorda and Koen Claessen. Sat-based assistance in abstraction refinement for symbolic trajectory evaluation. In *CAV'06*, pages 175–189, 2006.
13. C.-J. H. Seger and R. E. Bryant. Formal verification by symbolic evaluation of partially-ordered trajectories. *Formal Methods in System Design*, 6(2), 1995.
14. C.-J. H. Seger, R. B. Jones, J. W. O'Leary, T. F. Melham, M. Aagaard, C. Barrett, and D. Syme. An industrially effective environment for formal hardware verification. *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, 24(9), 2005.
15. Rachel Tzoref and Orna Grumberg. Automatic refinement and vacuity detection for symbolic trajectory evaluation. In *CAV06*, pages 190–204, 2006.
16. B. Wile, W. Roesner, and J. Goss. *Comprehensive Functional Verification: The Complete Industry Cycle*. Morgan-Kaufmann, 2005.
17. J.C. Wilson. *Symbolic Simulation Using Automatic Abstraction of Internal Node Values*. PhD thesis, Stanford University, Dept. of Electrical Engineering, 2001.