Symbolic Assume-Guarantee Reasoning through BDD Learning

Fei He
Tsinghua University, China

Bow-Yaw Wang
Academia Sinica, Taiwan

Liangze Yin
Tsinghua University, China

Lei Zhu
Tsinghua University, China

ABSTRACT

Both symbolic model checking and assume-guarantee reasoning aim to circumvent the state explosion problem. Symbolic model checking explores many states simultaneously and reports numerous erroneous traces. Automated assume-guarantee reasoning, on the other hand, infers contextual assumptions by inspecting spurious erroneous traces. One would expect that their integration could further improve the capacity of model checking. Yet examining numerous erroneous traces to deduce contextual assumptions can be very time-consuming. The integration of symbolic model checking and assume-guarantee reasoning is thus far from clear. In this paper, we present a progressive witness analysis algorithm for automated assume-guarantee reasoning to exploit a multitude of traces from BDD-based symbolic model checkers. Our technique successfully integrates symbolic model checking with automated assume-guarantee reasoning by directly inferring BDD's as implicit assumptions. It outperforms monolithic symbolic model checking in four benchmark problems and an industrial case study in experiments.

Categories and Subject Descriptors

D.2.4 [**Software Engineering**]: Software/Program Verification; F.3.1 [**Logics and Meanings of Programs**]: Specifying and Verifying and Reasoning about Programs

General Terms

Verification, Reliability

Keywords

Software model checking, symbolic model checking, assume-guarantee reasoning, algorithmic learning, safety

1. INTRODUCTION

The advent of multi-core processors not only improves the efficiency of computing but also increases the complexity of programming. Intricate interactions among concurrent processes often incur unexpected faulty program behaviors eluding programmers' inspection. Program verification hence plays an even more important role during software development. Model checking is a program verification technique that automatically analyzes the correctness of programs by formal mathematical reasoning [18, 16].

In model checking, programmers specify a model for the system under verification, and a formal property about intended system behaviors. A model checker verifies the model against the property by exploring all system behaviors. If any behavior does not fulfill the intended property, the model checker will report it. Otherwise, the model satisfies the property conclusively since there is no violation. Exploring all behaviors can be very expensive computationally since the number of system states can grow exponentially in the number of system components. The state explosion problem greatly impedes the effectiveness of model checking.

Symbolic model checking [1] is a well-known technique for ameliorating the state explosion problem [7, 18]. In symbolic model checking, system states are implicitly represented by predicates, as well as the initial states and transition relation of the system. Using data structures such as Binary Decision Diagrams (BDD's), a large number of states are explored simultaneously by standard predicate operations (conjunction, disjunction, quantifier elimination, etc). Since it is no longer necessary to enumerate states one at a time, the state explosion problem can be alleviated.

By separating concerns of software functionality, component-based designs improve reusability and quality of software systems [5, 30]. In component-based software engineering, a software system is composed of several components of different functionality. When each software component has clearly specified functions and interfaces, its reusability is improved. Time of developing software systems is subsequently reduced. Incidentally, the separation of concerns is not only found in software development but also verification.

Assume-guarantee reasoning is a technique to improve the capacity of model checking. In assume-guarantee reasoning, one verifies whether every component behaves correctly under certain carefully chosen contextual assumptions [21]. If so, the system of components is correct by the soundness of assume-guarantee reasoning. Since each component is verified against contextual assumptions separately, the state explosion problem can be avoided. Choosing contextual assumptions however is not easy. Oftentimes, programmers

[1]By "symbolic", we mean BDD-based techniques unless stated otherwise.

ICSE'14, May 31 – June 7, 2014, Hyderabad, India
ACM 978-1-4503-2756-5/14/05
http://dx.doi.org/10.1145/2568225.2568253

have to provide contextual assumptions manually. Such laborious tasks are very time consuming and can be extremely difficult to carry out on large systems.

In order to address the assumption generation problem, machine learning is applied to infer contextual assumptions. In [20, 1, 8, 35, 24, 32, 25, 13, 34], the L^* learning algorithm for regular languages [2] is adopted to solve the assumption generation problem. Instead of programmers, one employs the L^* learning algorithm to contrive contextual assumptions under the supervision of a mechanical teacher. When the L^* learning algorithm purports a contextual assumption, one performs assume-guarantee reasoning. If the contextual assumption is strong enough to establish the correctness of the system, we are done. Otherwise, the model checker reports an erroneous behavior under the purported contextual assumption. When the contextual assumption is overly weak, the reported behavior may not be a valid system behavior. The mechanical teacher has to check if the erroneous behavior is valid. If not, the mechanical teacher analyzes the spurious erroneous behavior and guides the learning algorithm to revise the next contextual assumption.

Both symbolic model checking and automated assume-guarantee reasoning aim to circumvent the state explosion problem. One wonders if their integration can further improve the capacity of model checking. On closer examination, it is not hard to see that an effective integration is not straightforward. First, the L^* learning algorithm represents contextual assumptions explicitly. Converting explicit contextual assumptions to implicit representations induces overheads.[2] Second, when a purported contextual assumption fails to verify the system, a symbolic model checker in fact gives numerous erroneous behaviors. The mechanical teacher has to analyze lots of behaviors to supervise learning algorithms. Since it is infeasible to enumerate all implicitly represented behaviors, it is unclear how to extract necessary information to infer purported contextual assumptions efficiently. Third, a learning algorithm purports a new contextual assumption after receiving a spurious behavior from the mechanical teacher. The mechanical teacher actually possesses a number of spurious behaviors after symbolic model checking. Efficiently passing numerous spurious behaviors to the learning algorithm is not at all obvious. A more sophisticated mechanism which takes full advantage of symbolic model checking is certainly desirable.

In this paper, we present a progressive analysis algorithm to take advantage of both symbolic model checking and automated assume-guarantee reasoning. When spurious behaviors are obtained, the mechanical teacher guides the learning algorithm to remove a spurious behavior from contextual assumptions. When the learning algorithm purports a revised contextual assumption, our progressive algorithm checks if previous spurious behaviors are eliminated in the new contextual assumption. If not, another spurious behavior is returned to guide the learning algorithm to the next revision. Only when all spurious behaviors are accounted for, can the verification algorithm proceed to perform assume-guarantee reasoning with the purported contextual assumption. Observe that assume-guarantee reasoning is applied only when known spurious behaviors are eliminated. Our progressive approach hence minimizes the invocation of model checkers and maximizes the utility of the information in implicitly represented spurious behaviors.

[2]Implicit learning attempts to address the problem [12, 11], but other problems peculiar to symbolic model checking remain.

Our technical contributions are summarized as follows.

- We adopt a BDD learning algorithm to generate implicit contextual assumptions in our fully symbolic technique. Different from [11], we implement the classification tree-based BDD learning algorithm [31] in our symbolic assume-guarantee reasoning technique.
- We propose a progressive analysis algorithm for the mechanical teacher in automated assume-guarantee reasoning. The new algorithm takes advantage of implicitly represented behaviors from symbolic model checkers. It enables the integration of symbolic model checking and automated assume-guarantee reasoning to ameliorate the state explosion problem.
- We compare our new technique with the monolithic symbolic model checker NuSMV 2.4.3 [15]. Experimental results show that our technique outperforms monolithic model checking in four benchmark problems (mini-Rubik's cube, Rubik's cube, dining philosophers, and dining cryptographers). Our compositional technique moreover improves both time and space in an industrial gate control system.

The paper is organized as follows. We review related work in Section 2. After preliminaries (Section 3), automated assume-guarantee reasoning with implicit learning is briefly reviewed in Section 4. Section 5 presents our technical contribution. It is followed by experimental results (Section 6). Finally, we conclude our presentation in Section 7.

2. RELATED WORK

Automated assume-guarantee reasoning is proposed in [20]. The authors apply the L^* algorithm to generate explicit deterministic finite automata as contextual assumptions [2]. Several optimizations for the L^* algorithm are available [32, 9, 35, 4]. A tree-based contextual assumption generation algorithm is also developed in [26, 27]. Interface synthesis by learning is reported in [28, 3]. Predicate abstraction has been used to represent software state space symbolically [33, 22], but deriving symbolic representations of *transitions* is different. Indeed, contextual assumptions inferred by these techniques are still represented explicitly. Converting explicit contextual assumptions to BDD's for symbolic model checking can induce overheads. Inferring explicit contextual assumptions is not optimal for symbolic model checking.

Learning implicit contextual assumptions based on the CDNF algorithm [6] is proposed in [12]. The work [11] compares the CDNF algorithm and the L^*-based BDD learning algorithm [23] in the context of assume-guarantee reasoning through implicit learning. Rather than BDD-based symbolic model checking, both works [12, 11] employ a SAT-based symbolic model checker. Since SAT-based symbolic model checking reports only one erroneous behavior, the simple witness analysis algorithm suffices. On the other hand, our progressive algorithm is designed to analyze numerous erroneous behaviors. It enables programmers to deploy a BDD-based symbolic model checker in automated assume-guarantee reasoning. Additionally, we implement the classification tree-based BDD learning algorithm [31]. The new algorithm is asymptotically more efficient than the L^*-based learning algorithm implemented in [11] by a linear factor.

The symbolic compositional verification technique in [1, 32] is only relevant in appearance. Based on the L^* learning algorithm, the symbolic L^* algorithm uses BDD's to encode input symbols on transitions of contextual assumptions. States of contextual assumptions however are still represented explicitly. Converting semi-symbolic contextual assumptions for symbolic model checking may induce overheads. Moreover, the symbolic L^* algorithm infers deterministic contextual assumptions in a partially implicit representation. Our technique adopts a BDD learning algorithm and infers nondeterministic contextual assumptions in BDD's, a fully implicit representation.

Finally, some drawbacks of automated assume-guarantee reasoning with the basic L^* algorithm are discussed in [19]. Many issues have been addressed in various optimizations of the basic algorithm since then [32, 9, 35, 26, 27, 4]. We adopt a BDD learning algorithm and infer contextual assumptions implicitly. Observations in [19] no longer apply. Moreover, the work [19] compares compositional reasoning with monolithic verification within enumerative model checking, a context very different from symbolic model checking.

3. PRELIMINARIES

Let $\mathbb{B} = \{\mathsf{ff}, \mathsf{tt}\}$ be the *Boolean domain* with the *truth values* ff and tt. For a set $\mathbf{x}$ of Boolean variables, define $\mathbf{x}' = \{x' : x \in \mathbf{x}\}$ and $'\mathbf{x} = \{'x : x \in \mathbf{x}\}$. A *valuation* over $\mathbf{x}$ is a mapping from $\mathbf{x}$ to $\mathbb{B}$. We write $Val_{\mathbf{x}}$ for the set of valuations over $\mathbf{x}$. A *predicate* $\phi(\mathbf{x})$ (or ϕ when the variables $\mathbf{x}$ are clear from the context) is a Boolean function over the Boolean variables $\mathbf{x}$. For a predicate ϕ and a valuation $s \in Val_{\mathbf{x}}$, *s satisfies* ϕ (written $s \models \phi$) if ϕ evaluates to tt by assigning $s(x)$ to $x \in \mathbf{x}$. Similarly, a pair of valuations $(s,t) \in Val_{\mathbf{x}} \times Val_{\mathbf{x}}$ *satisfies* a predicate $\psi(\mathbf{x}, \mathbf{x}')$ (written $(s,t) \models \psi$) if ψ evaluates to tt by assigning $s(x)$ and $t(x)$ to $x \in \mathbf{x}$ and $x' \in \mathbf{x}'$ respectively. We write $[\![\phi]\!]$ for the set of satisfying valuations for the predicate ϕ, namely, $[\![\phi]\!] = \{s \in Val_{\mathbf{x}} : s \models \phi\}$. A predicate ϕ is a *tautology* (written $\models \phi$) if $s \models \phi$ for every valuation $s \in Val_{\mathbf{x}}$.

Reduced and ordered binary decision diagrams (BDD's) are a representation for predicates. For any predicate ϕ, we write $BDD(\phi)$ for the BDD representing ϕ. The representation is canonical in the sense that $BDD(\phi) = BDD(\psi)$ if and only if $\phi = \psi$. For instance, consider $\phi = x_0 \vee \neg x_0$ and $\psi = x_1 \vee \neg x_1$. Then $BDD(\phi) = BDD(\psi)$ since both ϕ and ψ are tautologies. Boolean operations and quantification are allowed in BDD's. One can compute $BDD(\neg\phi)$, $BDD(\phi \wedge \psi)$, $BDD(\phi \vee \psi)$, $BDD(\forall x.\phi)$, and $BDD(\exists x.\phi)$ from $BDD(\phi)$ and $BDD(\psi)$ for any predicates ϕ, ψ, and variable $x \in \mathbf{x}$.

A *transition system* $M = \langle \mathbf{x}, \iota, \tau \rangle$ consists of a set $\mathbf{x}$ of Boolean variables, an *initial predicate* $\iota(\mathbf{x})$, and a *transition predicate* $\tau(\mathbf{x}, \mathbf{x}')$. An $\mathbf{x}$*-state* is a valuation over $\mathbf{x}$. An $\mathbf{x}$-state s is *M-initial* if $s \models \iota$. A pair (s,t) of $\mathbf{x}$-states is called an *M-transition* if $(s,t) \models \tau$. If (s,t) is an M-transition, t is an *M-successor* of s. An *M-trace* is a sequence $\sigma = [s_0, s_1, \cdots, s_n]$ of $\mathbf{x}$-states such that s_0 is M-initial and (s_i, s_{i+1}) is an M-transition for $i = 0, \ldots, n-1$. We write $Tr(M)$ for the set of M-traces. An $\mathbf{x}$-state s is *M-reachable* if there is an M-trace ending in s. For any predicate π, a sequence $\sigma = [s_0, s_1, \ldots, s_n]$ of $\mathbf{x}$-states *satisfies* π (written $\sigma \models \pi$) if $s_i \models \pi$ for $i = 0, \ldots, n$. We say M *satisfies* π (written $M \models \pi$) if $\sigma \models \pi$ for every $\sigma \in Tr(M)$.

Let $M = \langle \mathbf{x}, \iota, \tau \rangle$ be a transition system and π a predicate. The *invariant checking* problem is to decide whether M satisfies π. A *model checker* solves the invariant checking problem by exploring M-reachable states and storing them in predicates. When M does not satisfy π, a model checker returns a sequence of predicates $[S_0(\mathbf{x}), S_1(\mathbf{x}), \ldots, S_k(\mathbf{x})]$ that

1. $\models S_0 \Rightarrow \iota$ (every $\mathbf{x}$-state in $[\![S_0]\!]$ is M-initial);
2. $\models \exists \mathbf{x}.S_k \wedge \neg\pi$ (an $\mathbf{x}$-state in $[\![S_k]\!]$ does not satisfy π); and
3. for every $0 < i \leq k$, $\models S_i \Rightarrow \exists '\mathbf{x}.S_{i-1}('\mathbf{x}) \wedge \tau('\mathbf{x}, \mathbf{x})$ (every $\mathbf{x}$-state in $[\![S_i]\!]$ is an M-successor of an $\mathbf{x}$-state in $[\![S_{i-1}]\!]$).

The sequence $[S_0, S_1, \ldots, S_k]$ is called a *witness* to $M \not\models \pi$.[3] For a transition system $M = \langle \mathbf{x}, \iota, \tau \rangle$ and a witness $[S_0, S_1, \ldots, S_k]$ to $M \not\models \pi$, an M-trace $\sigma = [s_0, s_1, \ldots, s_k]$ is *represented* in $[S_0, S_1, \ldots, S_k]$ if $s_i \models S_i$ for every $0 \leq i \leq k$. For any witness $[S_0, S_1, \ldots, S_k]$ to $M \not\models \pi$, there is an M-trace σ represented in $[S_0, S_1, \ldots, S_k]$ such that $\sigma \not\models \pi$. Note that a witness represents a set of M-traces.

Let $M_i = \langle \mathbf{x}_i, \iota_i, \tau_i \rangle$ be transition systems for $i = 0, 1$ ($\mathbf{x}_i$'s are not necessarily disjoint). The composition $M_0 \| M_1 = \langle \mathbf{x}, \iota, \tau \rangle$ is a transition system where $\mathbf{x} = \mathbf{x}_0 \cup \mathbf{x}_1$, $\iota = \iota_0 \wedge \iota_1$, and $\tau = \tau_0 \wedge \tau_1$. The *projection* $s|_{\mathbf{y}}$ of an $\mathbf{x}$-state s on $\mathbf{y} \subseteq \mathbf{x}$ is a $\mathbf{y}$-state such that $s|_{\mathbf{y}}(y) = s(y)$ for $y \in \mathbf{y}$. The *projection* $\sigma|_{\mathbf{y}}$ of a sequence $\sigma = [s_0, s_1, \ldots, s_n]$ of $\mathbf{x}$-states on $\mathbf{y} \subseteq \mathbf{x}$ is the sequence $[s_0|_{\mathbf{y}}, s_1|_{\mathbf{y}}, \cdots, s_n|_{\mathbf{y}}]$ of $\mathbf{y}$-states. Note that $\sigma \in Tr(M_0 \| M_1)$ if and only if $\sigma|_{\mathbf{x}_0} \in Tr(M_0)$ and $\sigma|_{\mathbf{x}_1} \in Tr(M_1)$.

Let $M = \langle \mathbf{x}, \iota, \tau \rangle$ be a transition system. A transition system $N = \langle \mathbf{x}, \lambda, \theta \rangle$ *simulates* M (written $M \preceq N$) if $\models \iota \Rightarrow \lambda$ and $\models \tau \Rightarrow \theta$. In other words, N simulates M if every M-initial $\mathbf{x}$-state is N-initial, and every M-transition is also an N-transition.

A *proof rule* is of the form $\dfrac{\Phi_0 \quad \Phi_1 \quad \cdots \quad \Phi_k}{\Psi}$ where $\Phi_0, \ldots, \Phi_k$ are the *premises* of the proof rule, and Ψ is its *conclusion*. A proof rule is *sound* if its conclusion holds when its premises are fulfilled; it is *invertible* if its premises can be fulfilled when its conclusion holds.

THEOREM 1 ([12]). *Let $M_i = \langle \mathbf{x}_i, \iota_i, \tau_i \rangle$ be transition systems for $i = 0, 1$, $\mathbf{x} = \mathbf{x}_0 \cup \mathbf{x}_1$, and $\pi(\mathbf{x})$ a predicate. The following proof rule is sound and invertible:*

$$\frac{M_1 \preceq A \qquad M_0 \| A \models \pi}{M_0 \| M_1 \models \pi}$$

In the proof rule of Theorem 1, the transition system A is called a *contextual assumption* of M_0.

4. IMPLICIT LEARNING FRAMEWORK

In automated assume-guarantee reasoning through implicit learning [12], a learning algorithm for predicates is used to generate contextual assumptions automatically. In this section, we review BDD learning and the framework of assume-guarantee reasoning with implicit learning.

4.1 Learning BDD's

Let $f(\mathbf{x})$ be an unknown *target* predicate. A BDD learning algorithm infers $BDD(f)$ by making queries. It assumes a

[3]Most model checkers only report sequences of states for debugging purposes, but such witnesses are in fact computed.

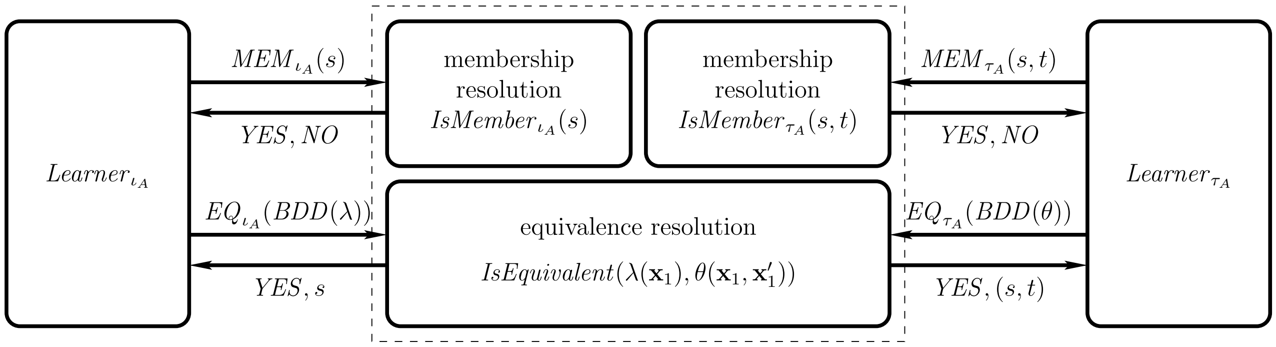
$MEM_{\iota_A}(s)$
YES, NO
$Learner_{\iota_A}$
$EQ_{\iota_A}(BDD(\lambda))$
YES, s
membership resolution $IsMember_{\iota_A}(s)$
membership resolution $IsMember_{\tau_A}(s,t)$
equivalence resolution $IsEquivalent(\lambda(\mathbf{x}_1), \theta(\mathbf{x}_1, \mathbf{x}'_1))$
$MEM_{\tau_A}(s,t)$
YES, NO
$Learner_{\tau_A}$
$EQ_{\tau_A}(BDD(\theta))$
$YES, (s,t)$

Figure 2: Implicit Learning Framework

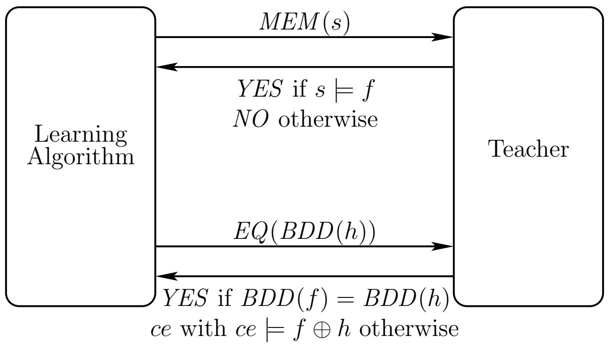

Figure 1: Angluin's Learning Model

teacher who knows the target predicate f and answers the following types of queries (Figure 1) [2, 6, 23, 31]:

- *Membership queries* $MEM(s)$ with $s \in Val_{\mathbf{x}}$. If $s \models f$, the teacher answers *YES*; otherwise, the teacher answers *NO*.
- *Equivalence queries* $EQ(BDD(h))$ with a *conjecture* $BDD(h(\mathbf{x}))$. If $BDD(f) = BDD(h)$, the teacher answers *YES*. Otherwise, the teacher sends a *counterexample* $ce \in Val_{\mathbf{x}}$ such that $ce \models f \oplus h$ where $\oplus$ is the exclusive-or operator.

A membership query $MEM(s)$ asks if the unknown target predicate f evaluates to $\mathtt{tt}$ under the valuation $s \in Val_{\mathbf{x}}$. An equivalence query $EQ(BDD(h))$, on the other hand, purports a conjecture h and asks if $BDD(f) = BDD(h)$. Recall that BDD's are canonical. When $BDD(f) \neq BDD(h)$, $f \neq h$. Hence there is a valuation $ce \in Val_{\mathbf{x}}$ such that f and h evaluate to different truth values. In other words, $f \oplus h$ evaluates to $\mathtt{tt}$ under a certain ce. The teacher returns such a valuation ce as a counterexample to the equivalence query.

Since the number of predicates over $\mathbf{x}$ is finite, any unknown predicate can be inferred by enumeration. For instance, one can obtain the truth table of the unknown target predicate by asking $2^{|\mathbf{x}|}$ membership queries and generate the BDD accordingly. The naive BDD learning algorithm however requires an exponential number of membership queries in the number of Boolean variables. Remarkably, it is shown that the BDD of an arbitrary target predicate can be learned within a linear number of equivalence and a quadratic number of membership queries in its size.

Theorem 2 ([31]). *For any unknown target predicate $f(\mathbf{x})$, one can infer $BDD(f)$ with at most n equivalence queries and $2n(\lceil \lg m \rceil + 3n)$ membership queries where n is the size of $BDD(f)$ and m is the number of Boolean variables in $\mathbf{x}$.*

4.2 Implicit Learning

The basic idea of implicit learning is to guide the learning algorithm to infer the initial and transition predicates of a contextual assumption [20, 12]. Since a contextual assumption consists of two predicates, two instances of the BDD learning algorithm are deployed. The instance $Learner_{\iota_A}$ infers the initial predicate of a contextual assumption, and the other instance $Learner_{\tau_A}$ infers its transition predicate. Both instances make membership and equivalence queries to a mechanical teacher. Figure 2 shows the implicit learning framework. For clarity, subscripts are used to differentiate queries from the two learners in the figure: $Learner_{\iota_A}$ makes membership query $MEM_{\iota_A}(s)$ and equivalence query $EQ_{\iota_A}(BDD(\lambda))$; and $Learner_{\tau_A}$ makes membership query $MEM_{\tau_A}(s,t)$ and equivalence query $EQ_{\tau_A}(BDD(\theta))$.

For a membership query $MEM_{\iota_A}(s)$, the mechanical teacher checks if the valuation s satisfies the initial predicate of M_1. If so, the mechanical teacher answers *YES*. Otherwise, it answers *NO*. Conceptually, the mechanical teacher uses the initial predicate of M_1 as the unknown target predicate (Algorithm 1). If no other contextual assumption is found, $Learner_{\iota_A}$ will eventually infer the initial predicate of M_1.

```
// $M_1 = \langle \mathbf{x}_1, \iota_1, \tau_1 \rangle$
**Input**: $s \in Val_{\mathbf{x}_1}$ : a valuation
**Output**: *YES* or *NO*
**if** $s \models \iota_1$ **then** send *YES* **else** send *NO*;
```

Algorithm 1: $IsMember_{\iota_A}(s)$

Similarly, membership queries from $Learner_{\tau_A}$ are resolved by using the transition predicate of M_1 as the unknown target predicate (Algorithm 2). Again, $Learner_{\tau_A}$ will infer the transition predicate of M_1 eventually if no other contextual assumption is found. Observe that membership queries from the learners are resolved separately. Synchronization is not needed between membership query resolution algorithms.

Equivalence queries, on the other hand, require synchronization. Assume that $Learner_{\iota_A}$ makes an equivalence query $EQ_{\iota_A}(BDD(\lambda))$. The mechanical teacher has to answer whether $BDD(\lambda)$ represents the initial predicate of a contex-

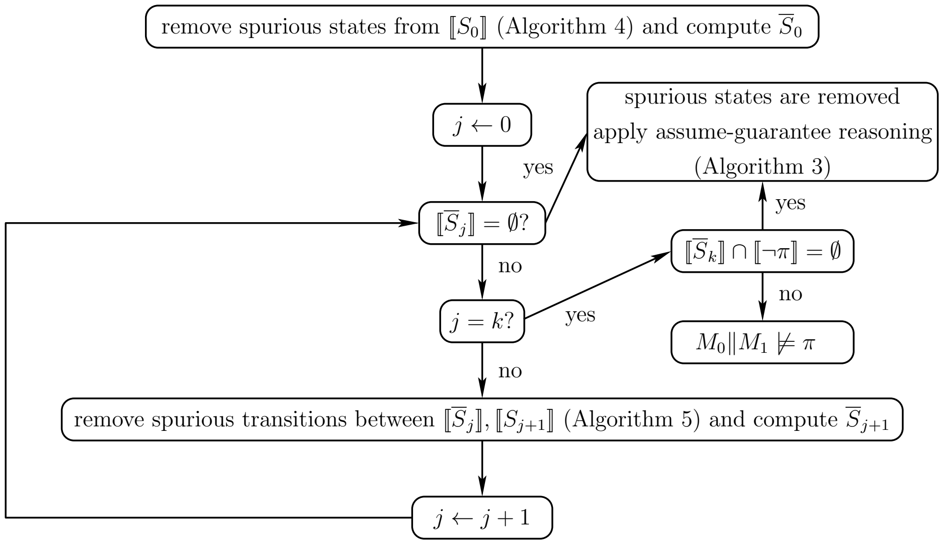
remove spurious states from $[\![S_0]\!]$ (Algorithm 4) and compute $\overline{S}_0$
$j \leftarrow 0$
spurious states are removed
apply assume-guarantee reasoning
(Algorithm 3)
yes
yes
$[\![\overline{S}_j]\!] = \emptyset$?
$[\![\overline{S}_k]\!] \cap [\![\neg\pi]\!] = \emptyset$
no
no
$j = k$?
yes
$M_0 \| M_1 \not\models \pi$
no
remove spurious transitions between $[\![\overline{S}_j]\!], [\![S_{j+1}]\!]$ (Algorithm 5) and compute $\overline{S}_{j+1}$
$j \leftarrow j + 1$

Figure 3: Overview of Progressive Witness Analysis

```
// $M_1 = \langle \mathbf{x}_1, \iota_1, \tau_1 \rangle$
Input: $s, t \in Val_{\mathbf{x}_1}$ : valuations
Output: $YES$ or $NO$
if $(s,t) \models \tau_1$ then send $YES$ else send $NO$;
```

Algorithm 2: $IsMember_{\tau_A}(s,t)$

tual assumption A. To do so, it needs to check if the premises $M_1 \preceq A$ and $M_0 \| A \models \pi$ hold. The mechanical teacher needs the transition predicate of the contextual assumption A in order to check the premise. Hence the $EQ_{\iota_A}(BDD(\lambda))$ from $Learner_{\iota_A}$ cannot be resolved without the conjecture $BDD(\theta(\mathbf{x}_1, \mathbf{x}_1'))$ in $EQ_{\tau_A}(BDD(\theta))$ from $Learner_{\tau_A}$. Similarly, the $EQ_{\tau_A}(BDD(\theta))$ from $Learner_{\tau_A}$ cannot be resolved without the conjecture $BDD(\lambda(\mathbf{x}_1))$ in $EQ_{\iota_A}(BDD(\lambda))$ from $Learner_{\iota_A}$. Resolving either of the equivalence queries must be synchronized with the other. Subsequently, there is but one equivalence query resolution algorithm in the mechanical teacher.

When the equivalence query resolution algorithm receives the queries $EQ_{\iota_A}(BDD(\lambda))$ and $EQ_{\lambda_A}(BDD(\theta))$ from the learners, the mechanical teacher constructs a *purported* contextual assumption $A_{-1} = \langle \mathbf{x}_1, \lambda, \theta \rangle$ and checks if A_{-1} can serve as a contextual assumption (Algorithm 3). The algorithm first verifies if A_{-1} simulates M_1. If not, a counterexample is sent to $Learner_{\iota_A}$ or $Learner_{\tau_A}$. If A_{-1} simulates M_1, Algorithm 3 then invokes a model checker to verify the premise $M_0 \| A_{-1} \models \pi$. If the premise is also fulfilled, the proof rule in Theorem 1 applies. By the soundness of the proof rule, the mechanical teacher reports "$M_0 \| M_1 \models \pi$." Otherwise, a witness $[S_0, S_1, \ldots, S_k]$ to $M_0 \| A_{-1} \not\models \pi$ is obtained from the model checker. Recall that $M_1 \preceq A_{-1}$. $[S_0, S_1, \ldots, S_k]$ is not necessarily a witness to $M_0 \| M_1 \not\models \pi$. Particularly, the witness may represent an $M_0 \| A_{-1}$-trace that does not satisfy π even though $M_0 \| M_1 \models \pi$. Further analysis is required for the witness to $M_0 \| A_{-1} \not\models \pi$.

```
// $M_i = \langle \mathbf{x}_i, \iota_i, \tau_i \rangle$ for $i = 0, 1$ and $\mathbf{x} = \mathbf{x}_0 \cup \mathbf{x}_1$
Input: $\lambda(\mathbf{x}_1)$ : an initial predicate; $\theta(\mathbf{x}_1, \mathbf{x}_1')$ : a
       transition predicate
Output: "$M_0 \| M_1 \models \pi$", a witness to $M_0 \| M_1 \not\models \pi$, a
        counterexample to $EQ_{\iota_A}(BDD(\lambda))$, or a
        counterexample to $EQ_{\tau_A}(BDD(\theta))$
$A_{-1} \leftarrow \langle \mathbf{x}_1, \lambda, \theta \rangle$;
if $s \models \iota_1 \wedge \neg\lambda$ then
    send $s$ as the counterexample to $EQ_{\iota_A}(BDD(\lambda))$;
    receive an equivalence query $EQ_{\iota_A}(BDD(\lambda'))$;
    return $IsEquivalent(\lambda', \theta)$;
if $(s,t) \models \tau_1 \wedge \neg\theta$ then
    send $(s,t)$ as the counterexample to $EQ_{\tau_A}(BDD(\theta))$;
    receive an equivalence query $EQ_{\tau_A}(BDD(\theta'))$;
    return $IsEquivalent(\lambda, \theta')$;
if $M_0 \| A_{-1} \models \pi$ then report "$M_0 \| M_1 \models \pi$";
let $[S_0, S_1, \ldots, S_k]$ be a witness to $M_0 \| A_{-1} \not\models \pi$;
return $AnalyzeWitness(A_{-1}, S_0, S_1, \ldots, S_k)$;
```

Algorithm 3: $IsEquivalent$ (λ, θ)

5. ANALYZING SYMBOLIC WITNESSES

In order to describe witness analysis, we identify invalid behaviors. Let A be a purported contextual assumption. An $M_0 \| A$-initial state is *spurious* if it is not $M_0 \| M_1$-initial; and an $M_0 \| A$-transition is *spurious* if it is not an $M_0 \| M_1$-transition. Similarly, an $M_0 \| A$-trace is *spurious* if it is not an $M_0 \| M_1$-trace; and an $M_0 \| A$-reachable state is *spurious* if it is not $M_0 \| M_1$-reachable. Consider a trace $\sigma \in Tr(M_0 \| A_{-1})$ such that $\sigma \not\models \pi$. Then $\sigma|_{\mathbf{x}_0} \in Tr(M_0)$ and $\sigma|_{\mathbf{x}_1} \in Tr(A_{-1})$. Let $\sigma|_{\mathbf{x}_1} = [s_0, s_1, \ldots, s_n]$. If s_0 is M_1-initial and (s_i, s_{i+1}) is an M_1-transition for every $0 \leq i < n$, then $\sigma|_{\mathbf{x}_1}$ is in fact an M_1-trace. We conclude $M_0 \| M_1 \not\models \pi$. If, for instance, (s_3, s_4) is not an M_1-transition, then (s_3, s_4) is a spurious A_{-1}-transition. We should return (s_3, s_4) as a counterexample so that the learning algorithm removes the spurious transition from future conjectures. Since only one trace is inspected, we call this procedure the *simple witness analysis* algorithm. In [12, 11], a SAT-based symbolic model checker is used to verify the premise $M_0 \| A_{-1} \models \pi$. Since a SAT-based symbolic model checker returns only one trace as a witness to $M_0 \| A_{-1} \not\models \pi$, the simple witness analysis algorithm applies. Similarly, the simple algorithm suffices when an explicit model checker is used [20, 19].

The situation becomes slightly more complicated if the mechanical teacher uses a symbolic model checker. In this case, a witness to $M_0 \| A_{-1} \not\models \pi$ represents a number of traces in general. The mechanical teacher could pick an arbitrary $M_0 \| A_{-1}$-trace in the witness and analyze the trace as before. Lots of information in the witness however would be wasted. Consider, for instance, spurious transitions of different traces represented in a witness. Since only one trace is inspected by the mechanical teacher, a spurious transition of the inspected trace is returned as a counterexample. Spurious transitions in other traces may still remain in the next conjecture from the learning algorithm. They may again appear in another witness when the mechanical teacher checks the premises with the next purported contextual assumption. This is clearly a waste of computation resources.

On the other hand, examining all traces in a witness is by no means straightforward. A witness represents numerous traces. It is infeasible to inspect all transitions in all traces iteratively. Moreover, BDD learning algorithms take but one counterexample for each equivalence query. Yet the mechanical teacher already have numerous counterexamples from spurious transitions after symbolic model checking. It is unclear how one may convey all these counterexamples to BDD learning algorithms effectively. These questions must be addressed properly in order to integrate symbolic model checking with automated assume-guarantee reasoning.

We propose to analyze spurious traces progressively. Given a witness $[S_0, S_1, \ldots, S_k]$ to $M_0 \| A \not\models \pi$, our progressive witness analysis algorithm works as follows (Figure 3). First, the algorithm obtains a predicate $\overline{S}_0$ by eliminating all spurious $M_0 \| A$-initial states from $[\![S_0]\!]$. The set $[\![\overline{S}_0]\!]$ thus contains $M_0 \| M_1$-reachable states. Inductively, assume that the algorithm has obtained the predicate $\overline{S}_j$ by eliminating all spurious $M_0 \| A$-reachable states from $[\![S_j]\!]$. The set $[\![\overline{S}_j]\!]$ hence contains only $M_0 \| M_1$-reachable states. The witness analysis algorithm checks if $[\![\overline{S}_j]\!]$ is empty. If so, all known spurious $M_0 \| A$-traces are eliminated. The mechanical teacher applies the assume-guarantee reasoning proof rule in Theorem 1 with the purported assumption A. Otherwise, the algorithm obtains a predicate $\overline{S}_{j+1}$ by eliminating all spurious $M_0 \| A$-transitions. The set $[\![\overline{S}_{j+1}]\!]$ hence contains only $M_0 \| M_1$-reachable states. When the sequence of predicates $[\overline{S}_0, \overline{S}_1, \ldots, \overline{S}_k]$ is finally obtained, the set $[\![\overline{S}_k]\!]$ contains only $M_0 \| M_1$-reachable states. If $[\![\overline{S}_k]\!]$ contains a state not satisfying π, the sequence $[\overline{S}_0, \overline{S}_1, \ldots, \overline{S}_k]$ is indeed a witness to $M_0 \| M_1 \not\models \pi$.

We now give details of the procedure that eliminates spurious initial states. Algorithm 4 returns a revised initial predicate $\overline{\lambda}(\mathbf{x}_1)$ for the purported contextual assumption $A = \langle \mathbf{x}_1, \lambda, \theta \rangle$, which removes spurious $M_0 \| A$-initial states from $[\![S]\!]$. Conceptually, Algorithm 4 considers the purported contextual assumption $\overline{A} = \langle \mathbf{x}_1, \overline{\lambda}, \theta \rangle$ in the while loop. The contextual assumption $\overline{A}$ is initialized to A. In each iteration, we have an $\mathbf{x}$-state u such that $u \models \iota_0 \wedge \overline{\lambda}$ but $u \not\models \iota_0 \wedge \iota_1$. The state $u \in [\![S]\!]$ is a spurious $M_0 \| A$-initial state. Algorithm 4 then sends $u|_{\mathbf{x}_1}$ as a counterexample to $EQ_{\iota_A}(BDD(\overline{\lambda}))$. Informally, the counterexample $u|_{\mathbf{x}_1}$ informs $Learner_{\iota_A}$ that the conjecture $BDD(\overline{\lambda})$ is incorrect at

```
// $M_i = \langle \mathbf{x}_i, \iota_i, \tau_i \rangle$ for $i = 0, 1$ and $\mathbf{x} = \mathbf{x}_0 \cup \mathbf{x}_1$
Input: $\langle \mathbf{x}_1, \lambda, \theta \rangle$ : a purported contextual assumption;
       $S(\mathbf{x})$ : a predicate
Output: $\overline{\lambda}(\mathbf{x}_1)$ : an initial predicate
$\overline{\lambda} \leftarrow \lambda$;
while $u \models \neg\iota_1 \wedge \iota_0 \wedge \overline{\lambda} \wedge S$ do
    // $\overline{A} = \langle \mathbf{x}_1, \overline{\lambda}, \theta \rangle$ is the purported contextual
       assumption
    send $u|_{\mathbf{x}_1}$ as the counterexample to $EQ_{\iota_A}(BDD(\overline{\lambda}))$;
    receive an equivalence query $EQ_{\iota_A}(BDD(\overline{\lambda}'))$;
    $\overline{\lambda} \leftarrow \overline{\lambda}'$;
end
return $\overline{\lambda}$;
```

Algorithm 4: *UpdateInit*$(\langle \mathbf{x}_1, \lambda, \theta \rangle, S)$

the valuation $u|_{\mathbf{x}_1}$. After receiving a new equivalence query $EQ_{\iota_A}(BDD(\overline{\lambda}'))$, Algorithm 4 considers the new purported contextual assumption $\langle \mathbf{x}_1, \overline{\lambda}', \theta \rangle$ in the next iteration.

When the algorithm returns an initial predicate $\overline{\lambda}$, we have $\models [\iota_0 \wedge \overline{\lambda} \wedge S] \Rightarrow \iota_1$ and hence $\models [\iota_0 \wedge \overline{\lambda} \wedge S] \Rightarrow [\iota_0 \wedge \iota_1]$. That is, every $M_0 \| \overline{A}$-initial state is also $M_0 \| M_1$-initial. Note that $\overline{A}$ does not necessarily simulate M_1. Lemma 1 summarizes *UpdateInit*$(\langle \mathbf{x}_1, \lambda, \theta \rangle, S)$ (Algorithm 4).

LEMMA 1. *Let $M_i = \langle \mathbf{x}_i, \iota_i, \tau_i \rangle$ be transition systems for $i = 0, 1$ and $\mathbf{x} = \mathbf{x}_0 \cup \mathbf{x}_1$. If $\overline{\lambda}(\mathbf{x}_1)$ is returned by UpdateInit $(\langle \mathbf{x}_1, \lambda, \theta \rangle, S)$ (Algorithm 4), then $\models [\iota_0 \wedge \overline{\lambda} \wedge S] \Rightarrow [\iota_0 \wedge \iota_1]$.*

The mechanical teacher eliminates spurious transitions from reachable states in a similar manner. Algorithm 5 returns a revised transition predicate $\overline{\theta}(\mathbf{x}_1, \mathbf{x}'_1)$ for the purported contextual assumption $A = \langle \mathbf{x}_1, \lambda, \theta \rangle$, which removes spurious $M_0 \| A$-transitions between $[\![R]\!]$ and $[\![S]\!]$. Conceptually, Algorithm 5 considers the purported contextual assumption $\overline{A} = \langle \mathbf{x}_1, \lambda, \overline{\theta} \rangle$ in the while loop. At first, the contextual assumption $\overline{A}$ is A. In each iteration, we have a transition $(u, v) \in [\![R]\!] \times [\![S]\!]$ such that $(u, v) \models \tau_0 \wedge \overline{\theta}$ but $(u, v) \not\models \tau_0 \wedge \tau_1$. The $M_0 \| \overline{A}$-transition (u, v) between $[\![R]\!]$ and $[\![S]\!]$ is spurious. Algorithm 5 then sends $(u|_{\mathbf{x}_1}, v|_{\mathbf{x}_1})$ as a counterexample to $EQ_{\tau_A}(BDD(\overline{\theta}))$. The counterexample $(u|_{\mathbf{x}_1}, v|_{\mathbf{x}_1})$ effectively informs the learning algorithm that the conjecture $BDD(\overline{\theta})$ is incorrect at the valuation $(u|_{\mathbf{x}_1}, v|_{\mathbf{x}_1})$. After receiving another equivalence query $EQ_{\tau_A}(BDD(\overline{\theta}'))$, Algorithm 5 considers the new purported contextual assumption $\langle \mathbf{x}_1, \lambda, \overline{\theta}' \rangle$ in the next iteration.

When the algorithm returns a transition predicate $\overline{\theta}$, we have $\models [\tau_0 \wedge \overline{\theta} \wedge R \wedge S(\mathbf{x}')] \Rightarrow \tau_1$ and hence $\models [\tau_0 \wedge \overline{\theta} \wedge R \wedge S(\mathbf{x}')] \Rightarrow [\tau_0 \wedge \tau_1]$. That is, every $M_0 \| \overline{A}$-transition between $[\![R]\!]$ and $[\![S]\!]$ is also an $M_0 \| M_1$-transition. Note that $\overline{A}$ does not necessarily simulate M_1. The following lemma summarizes *UpdateStep*$(\langle \mathbf{x}_1, \lambda, \theta \rangle, R, S)$ (Algorithm 5).

LEMMA 2. *Let $M_i = \langle \mathbf{x}_i, \iota_i, \tau_i \rangle$ be transition systems for $i = 0, 1$ and $\mathbf{x} = \mathbf{x}_0 \cup \mathbf{x}_1$. If $\overline{\theta}(\mathbf{x}_1, \mathbf{x}'_1)$ is returned by UpdateStep$(\langle \mathbf{x}_1, \lambda, \theta \rangle, R, S)$ (Algorithm 5), then $\models [\tau_0 \wedge \overline{\theta} \wedge R \wedge S(\mathbf{x}')] \Rightarrow [\tau_0 \wedge \tau_1]$.*

We are ready to give details of our progressive witness analysis algorithm. The witness analysis algorithm proceeds by stages (Algorithm 6). Let $A_{-1} = \langle \mathbf{x}_1, \lambda, \theta \rangle$ be a purported

```
// $M_i = \langle \mathbf{x}_i, \iota_i, \tau_i \rangle$ for $i = 0,1$ and $\mathbf{x} = \mathbf{x}_0 \cup \mathbf{x}_1$
**Input**: $\langle \mathbf{x}_1, \lambda, \theta \rangle$ : a purported contextual assumption;
        $R(\mathbf{x}), S(\mathbf{x})$ : predicates
**Output**: $\overline{\theta}(\mathbf{x}_1, \mathbf{x}_1')$ : a transition predicate
$\overline{\theta} \leftarrow \theta$;
**while** $(u, v) \models \neg\tau_1 \wedge \tau_0 \wedge \overline{\theta} \wedge R \wedge S(\mathbf{x}')$ **do**
    // $\overline{A} = \langle \mathbf{x}_1, \lambda, \overline{\theta} \rangle$ is the purported contextual
        assumption
    send $(u|_{\mathbf{x}_1}, v|_{\mathbf{x}_1})$ as the counterexample to
    $EQ_{\tau_A}(BDD(\overline{\theta}))$;
    receive an equivalence query $EQ_{\tau_A}(BDD(\overline{\theta}'))$;
    $\overline{\theta} \leftarrow \overline{\theta}'$;
**end**
**return** $\overline{\theta}$;
```

Algorithm 5: *UpdateStep*$(\langle \mathbf{x}_1, \lambda, \theta \rangle, R, S)$

contextual assumption with $M_1 \preceq A_{-1}$, and $[S_0, S_1, \dots, S_k]$ a witness to $M_0 \| A_{-1} \not\models \pi$. The witness analysis algorithm first invokes *UpdateInit*(A_{-1}, S_0) (Algorithm 4) to find an initial predicate $\overline{\lambda}(\mathbf{x}_1)$. Define $\overline{\theta}_0 = \theta$, $A_0 = \langle \mathbf{x}_1, \overline{\lambda}, \overline{\theta}_0 \rangle$, and $\overline{S}_0 = \iota_0 \wedge \overline{\lambda} \wedge S_0$. By Lemma 1, every $M_0 \| A_0$-initial state in $[\![\overline{S}_0]\!]$ is $M_0 \| M_1$-initial. The following lemma summarizes the first two lines of Algorithm 6.

LEMMA 3. *Let $M_i = \langle \mathbf{x}_i, \iota_i, \tau_i \rangle$ be transition systems for $i = 0,1$ and $\mathbf{x} = \mathbf{x}_0 \cup \mathbf{x}_1$. Right after line 2 of Algorithm 6, we have*

1. $\models \overline{S}_0(\mathbf{x}) \Rightarrow S_0(\mathbf{x})$*; and*
2. *every* $\mathbf{x}$*-state in* $[\![\overline{S}_0]\!]$ *is* $M_0 \| M_1$*-reachable.*

```
   // $M_i = \langle \mathbf{x}_i, \iota_i, \tau_i \rangle$ for $i = 0,1$ and $\mathbf{x} = \mathbf{x}_0 \cup \mathbf{x}_1$
   **Input**: $A_{-1} = \langle \mathbf{x}_1, \lambda, \theta \rangle$ : a purported contextual
           assumption; $[S_0(\mathbf{x}), S_1(\mathbf{x}), \dots, S_k(\mathbf{x})]$ : a witness
           to $M_0 \| A_{-1} \not\models \pi$
   **Output**: "$M_0 \| M_1 \models \pi$", a witness to $M_0 \| M_1 \not\models \pi$, a
            counterexample to $EQ_{\iota_A}(BDD(\lambda))$, or a
            counterexample to $EQ_{\tau_A}(BDD(\theta))$
1  $\overline{\lambda} \leftarrow$ UpdateInit$(A_{-1}, S_0)$ (Algorithm 4);
2  $\overline{S}_0 \leftarrow \iota_0 \wedge \overline{\lambda} \wedge S_0$;
3  $\overline{\theta}_0 \leftarrow \theta$;
4  **for** $j \leftarrow 0$ **to** $k - 1$ **do**
5      $A_j \leftarrow \langle \mathbf{x}_1, \overline{\lambda}, \overline{\theta}_j \rangle$;
6      **if** $\models \neg\overline{S}_j$ **then**
7          **return** IsEquivalent$(\overline{\lambda}, \overline{\theta}_j)$ (Algorithm 3);
8      **else**
9          $\overline{\theta}_{j+1} \leftarrow$ UpdateStep$(A_j, \overline{S}_j, S_{j+1})$ (Algorithm 5);
10         $\overline{S}_{j+1} \leftarrow$
           $\exists {}'\mathbf{x}.\tau_0({}'\mathbf{x}_0, \mathbf{x}_0) \wedge \overline{\theta}_{j+1}({}'\mathbf{x}_1, \mathbf{x}_1) \wedge \overline{S}_j({}'\mathbf{x}) \wedge S_{j+1}$;
11     **end**
12 **end**
13 **if** $\models \exists \mathbf{x}.\overline{S}_k \wedge \neg\pi$ **then** **report** $[\overline{S}_0, \overline{S}_1, \dots, \overline{S}_k]$;
14 **else** **return** IsEquivalent$(\overline{\lambda}, \overline{\theta}_k)$ (Algorithm 3);
```

Algorithm 6: *AnalyzeWitness*$(\langle \mathbf{x}_1, \lambda, \theta \rangle, S_0, S_1, \dots, S_k)$

At stage j, we have obtained a set $[\![\overline{S}_j]\!]$ of $M_0 \| M_1$-reachable states and a purported contextual assumption $A_j = \langle \mathbf{x}_1, \overline{\lambda}, \overline{\theta}_j \rangle$. The witness analysis algorithm eliminates spurious $M_0 \| A_j$-transitions from $[\![\overline{S}_j]\!]$ by invoking *UpdateStep* $(A_j, \overline{S}_j, S_{j+1})$ (Algorithm 5). It thus obtains a transition predicate $\overline{\theta}(\mathbf{x}_1, \mathbf{x}_1')$. Define $\overline{\theta}_{j+1} = \overline{\theta}$, $A_{j+1} = \langle \mathbf{x}_1, \overline{\lambda}, \overline{\theta}_{j+1} \rangle$, and $\overline{S}_{j+1} = \exists{}'\mathbf{x}.\tau_0({}'\mathbf{x}_0, \mathbf{x}_0) \wedge \overline{\theta}_{j+1}({}'\mathbf{x}_1, \mathbf{x}_1) \wedge \overline{S}_j({}'\mathbf{x}) \wedge S_{j+1}$. By Lemma 2, every $M_0 \| A_{j+1}$-transition between $[\![\overline{S}_j]\!]$ and $[\![\overline{S}_{j+1}]\!]$ is also an $M_0 \| M_1$-transition. Hence $[\![\overline{S}_{j+1}]\!]$ consists of $M_0 \| M_1$-reachable states. The witness analysis algorithm then proceeds to the next stage. The following lemma summarizes an iteration of the **for** loop at line 4 of Algorithm 6.

LEMMA 4. *Let $M_i = \langle \mathbf{x}_i, \iota_i, \tau_i \rangle$ be transition systems for $i = 0,1$ and $\mathbf{x} = \mathbf{x}_0 \cup \mathbf{x}_1$. At line 12 of Algorithm 6, we have*

1. $\models \overline{S}_{j+1}(\mathbf{x}) \Rightarrow S_{j+1}(\mathbf{x})$*; and*
2. *every* $\mathbf{x}$*-state in* $[\![\overline{S}_{j+1}]\!]$ *is* $M_0 \| M_1$*-reachable.*

By Lemma 3 and 4, $[[\![\overline{S}_0]\!], [\![\overline{S}_1]\!], \dots, [\![\overline{S}_j]\!]]$ is a sequence of $M_0 \| M_1$-reachable $\mathbf{x}$-states. There are two scenarios:

- $[\![\overline{S}_j]\!]$ is empty for some $0 \leq j < k$. $[\overline{S}_0, \overline{S}_1, \dots, \overline{S}_j]$ cannot be a witness to $M_0 \| M_1 \not\models \pi$. Recall that the purported contextual assumption A_j does not necessarily simulate M_1. The witness analysis algorithm invokes Algorithm 3 to check if A_j can serve as a contextual assumption (line 7, Algorithm 6).
- None of $[\![\overline{S}_0]\!], [\![\overline{S}_1]\!], \dots, [\![\overline{S}_{k-1}]\!]$ is empty. Algorithm 6 checks whether there is an $\mathbf{x}$-state in $[\![\overline{S}_k]\!]$ violating the predicate π. If so, $[\overline{S}_0, \overline{S}_1, \dots, \overline{S}_k]$ is a witness to $M_0 \| M_1 \not\models \pi$. Otherwise, the witness analysis algorithm invokes Algorithm 3 to check if $A_k = \langle \mathbf{x}_1, \overline{\lambda}, \overline{\theta}_k \rangle$ can be a contextual assumption (line 13, Algorithm 6).

Note that the witness analysis algorithm does not perform model checking. It instead revises purported contextual assumptions by inspecting the witness progressively. Only when all known spurious initial states and transitions are accounted for, can assume-guarantee reasoning be performed with purported contextual assumptions. Also note that purported contextual assumptions are *not* increasingly stronger. The learning algorithm has no information about unqueried M_1-transitions. It is free to include such transitions arbitrarily. The relation between successive purported contextual assumptions is thus very different from those of counterexample guided abstraction refinement [17].

THEOREM 3 (SOUNDNESS). *Let $M_i = \langle \mathbf{x}_i, \iota_i, \tau_i \rangle$ be transition systems for $i = 0,1$, $\mathbf{x} = \mathbf{x}_0 \cup \mathbf{x}_1$, and $\pi(\mathbf{x}), \lambda(\mathbf{x}_1)$, $\theta(\mathbf{x}_1, \mathbf{x}_1')$ predicates.*

1. *If IsEquivalent(λ, θ) (Algorithm 3) reports "$M_0 \| M_1 \models \pi$," then the purported contextual assumption $A = \langle \mathbf{x}_1, \lambda, \theta \rangle$ fulfills both premises of the proof rule in Theorem 1.*
2. *If IsEquivalent(λ, θ) (Algorithm 3) reports a sequence of predicates $[\overline{S}_0, \overline{S}_1, \dots, \overline{S}_k]$ for some $k \geq 0$, then $[\overline{S}_0, \overline{S}_1, \dots, \overline{S}_k]$ is a witness to $M_0 \| M_1 \not\models \pi$.*

For completeness, recall that the predicates ι_1 and τ_1 are the target predicates of *Learner*$_{\iota_A}$ and *Learner*$_{\tau_A}$ respectively. The transition system M_1 will be inferred as a purported contextual assumption if no other contextual assumption is found. Algorithm 3 can then decide whether $M_0 \| M_1$ satisfies π conclusively.

THEOREM 4 (COMPLETENESS). *Let $M_i = \langle \mathbf{x}_i, \iota_i, \tau_i \rangle$ be transition systems for $i = 0, 1$, $\mathbf{x} = \mathbf{x}_0 \cup \mathbf{x}_1$, and $\pi(\mathbf{x})$ a predicate.*

1. *If $M_0 \| M_1 \models \pi$, Learner$_{\iota_A}$ and Learner$_{\tau_A}$ will infer certain predicates $\lambda(\mathbf{x}_1)$ and $\theta(\mathbf{x}_1, \mathbf{x}'_1)$ such that IsEquivalent(λ, θ) (Algorithm 3) reports "$M_0 \| M_1 \models \pi$."*

2. *If $M_0 \| M_1 \not\models \pi$, Learner$_{\iota_A}$ and Learner$_{\tau_A}$ will infer certain predicates $\lambda(\mathbf{x}_1)$ and $\theta(\mathbf{x}_1, \mathbf{x}'_1)$ such that IsEquivalent(λ, θ) (Algorithm 3) returns a sequence $[\overline{S}_0, \overline{S}_1, \ldots, \overline{S}_k]$ for some $k \geq 0$.*

Optimizations: The BDD learning algorithms in [23, 31] are based on the learning algorithms for regular languages [2, 29]. As observed in [11], the first equivalence query made by BDD learning algorithms is always $EQ(BDD(\mathsf{ff}))$ when there is more than one Boolean variable. BDD learning algorithms hence revise the first purported contextual assumption $A_{-1} = \langle \mathbf{x}_1, \mathsf{ff}, \mathsf{ff} \rangle$ by adding M_1-initial $\mathbf{x}_1$-states or M_1-transitions. Consequently, BDD learning algorithms often infer the component M_1 as a contextual assumption. By complementing queries from BDD learning algorithms, one can ensure the first equivalence query to be $EQ(BDD(\mathsf{tt}))$ and hence obtain more effective contextual assumptions [11].

```
// $M_0 = \langle \mathbf{x}_0, \iota_0, \tau_0 \rangle$, $A_{-1} = \langle \mathbf{x}_1, \lambda, \theta \rangle$ and $\mathbf{x} = \mathbf{x}_0 \cup \mathbf{x}_1$
Input: a witness $[S_0(\mathbf{x}), S_1(\mathbf{x}), \ldots, S_k(\mathbf{x})]$ to
       $M_0 \| A_{-1} \not\models \pi$
Output: a witness to $M_0 \| A_{-1} \not\models \pi$ contains only
        traces violating $\pi$
$S_k \leftarrow S_k \wedge \neg\pi$;
for $j \leftarrow k - 1$ to 0 do
    $S_j \leftarrow S_j \wedge \exists \mathbf{x}'.\tau_0(\mathbf{x}_0, \mathbf{x}'_0) \wedge \theta(\mathbf{x}_1, \mathbf{x}'_1) \wedge S_{j+1}(\mathbf{x}')$;
end
return $[S_0, S_1, \ldots, S_k]$;
```

Algorithm 7: *WitnessBackward*$([S_0, S_1, \ldots, S_k])$

We can further reduce the number of spurious traces from symbolic model checking. Witnesses obtained by symbolic model checkers may contain unnecessary information. Consider a witness to $M_0 \| A_{-1} \not\models \pi$. If a spurious $M_0 \| A_{-1}$-trace in the witness satisfies π, it is not useful in revising the contextual assumption. Subsequently, it is more economic to consider $M_0 \| A_{-1}$-traces violating π in the witness. When the witness analysis algorithm (Algorithm 6) obtains a witness $[S_0, S_1, \ldots, S_k]$ to $M_0 \| A \not\models \pi$, the mechanical teacher should perform backward image computation to obtain useful spurious traces from the witness (Algorithm 7).

6. EXPERIMENTS

We compare three different verification techniques in the experiments: the NuSMV monolithic symbolic model checking, automated assume-guarantee reasoning with simple witness analysis, and automated assume-guarantee reasoning with progressive witness analysis. The two automated assume-guarantee reasoning techniques are implemented in NuSMV [15]. Both adopt the classification tree-based BDD learning algorithm to infer contextual assumptions [31].

Five examples are reported in the experiments: mini-Rubik's cube [36], Rubik's cube, dining philosophers, dining cryptographers [10], and an industrial gate control system [37]. We heuristically choose components to generate contextual assumptions from. Each experiment is allocated with one CPU core and 4GB of memory. All experiments are conducted on a 64-bit Linux 3.2.0 server with 4 2.40GHz Intel Xeon E5620 quadcore CPU's and 32GB RAM.

Table 1: Mini-Rubik's Cube

property	notsolved		overlap2	
	peak	time	peak	time
NuSMV	6.41×10^6	16.69s	7.28×10^7	1575.37s
AG1	-	-	-	-
AG*	1.78×10^6	3.08s	3.61×10^7	571.36s

Table 2: Rubik's Cube

property	corner		centercorner	
	peak	time	peak	time
NuSMV	–	–	–	–
AG1	–	–	–	–
AG*	1.31×10^7	300.24s	3.79×10^6	540.39s

property	overlap3	
	peak	time
NuSMV	–	–
AG1	–	–
AG*	3.22×10^7	363.41s

Mini-Rubik's cube. A mini-Rubik's cube is the $2 \times 2 \times 2$ version of the Rubik's cube. There are eight miniature cubes (called *cubies*) in the mini-Rubik's cube. Each cubie can be located in eight different positions. It moreover has three visible facets and hence three orientations in each position. Subsequently, a state of the mini-Rubik's cube consists of the position and the orientation of its eight cubies (Table 1).

In the table, the row NuSMV shows the results of monolithic symbolic model checking. The rows AG1 and AG* show the results of assume-guarantee reasoning with the simple and our progressive witness analysis algorithms (Algorithm 6) respectively. The column "peak" shows the number of peak BDD nodes during verification. Finally, the verification time is shown. In the table, the symbol "–" indicates either memory-out (4GB) or time-out (7200 seconds).

The property *notsolved* specifies that the mini-Rubik's cube cannot be solved. It fails from a chosen initial state. A witness gives a solution to the puzzle. Assume-guarantee reasoning with the simple witness analysis algorithm does not finish within the given time limit. Monolithic symbolic model checking uses more than six millions peak BDD nodes. On the other hand, assume-guarantee reasoning with our progressive witness analysis algorithm requires less than two millions of peak BDD nodes, and improves the verification time by 81%. The property *overlap2* specifies that the first two cubies cannot be in the same position from every initial state. Monolithic symbolic model checking requires more than seventy millions of peak BDD nodes and takes more than twenty-six minutes to conclude the verification. Assume-guarantee reasoning with our new witness analysis algorithm uses a half of peak BDD nodes and improves the verification time by 64%.

Rubik's Cube. A $3 \times 3 \times 3$ Rubik's cube has twenty six visible cubies. We partition visible cubies into three types. The first type consists of six cubies at the center of six faces of the cube. Each cubie of the first type has only one visible facet

Table 3: Dining Philosophers

nodes	4		5	
	peak	time	peak	time
NuSMV	2.16×10^6	2.63s	4.77×10^6	55.51s
AG1	1.41×10^6	1.62s	2.44×10^6	23.32s
AG*	1.41×10^6	1.46s	2.44×10^6	23.28s

nodes	6		7	
	peak	time	peak	time
NuSMV	5.23×10^7	1215.90s	–	–
AG1	6.95×10^6	231.67s	3.53×10^7	1796.82s
AG*	6.95×10^6	227.10s	3.53×10^7	1815.30s

Table 4: Dining Cryptographers

n	9		10	
	peak	time	peak	time
NuSMV	9.42×10^6	154.29s	21.27×10^6	576.64s
AG1	-	-	-	-
AG*	9.45×10^6	215.43s	19.98×10^6	821.77s

n	11		12	
	peak	time	peak	time
NuSMV	38.79×10^6	1823.84s	-	-
AG1	-	-	-	-
AG*	40.40×10^6	1866.74s	72.28×10^6	5715.22s

and thus one orientation. The second type contains eight cubies at the corners of the cube. Each cubie of this type has three visible facets and three orientations. The last type of cubies are on the edges of the Rubik's cube. Each edge cubie has two visible facets and two orientations.

Table 2 shows the experimental results. The property *corner* specifies that the eight corner cubies cannot be at their solved positions from a chosen initial state. The property *centercorner* states that the first two types of cubies cannot be at their solved positions from a chosen initial state. The property *overlap3* specifies that the first two center cubies cannot be at the same position from every initial state. For each property, neither monolithic symbolic model checking nor assume-guarantee reasoning with the simple witness analysis algorithm can verify these properties within the given resource bound. On the other hand, assume-guarantee reasoning with our progressive witness analysis algorithm performs rather well. The inferred contextual assumptions are significantly smaller than their targets. Subsequently, assume-guarantee reasoning with progressive witness analysis is able to verify all properties in this example.

Dining Philosophers. In this example, n philosophers sit in a round table. Two neighboring philosophers share a fork. When a philosopher decides to eat, she has to pick up the two forks shared with her two neighbors. In the experiment with n nodes, there are n processes for philosophers and n for forks. We verify that the first two philosophers cannot eat at the same time (Table 3).

Monolithic symbolic model checking is again outperformed by both automated assume-guarantee reasoning techniques. It cannot verify the experiment with 7 philosophers whereas assume-guarantee reasoning can finish the verification in about 30 minutes. Also note that monolithic symbolic model checking always has a larger number of peak BDD nodes. In the experiment with 6 nodes, the number is an order of magnitude more than those of assume-guarantee reasoning.

Dining Cryptographers. The dining cryptographers problem is an example of privacy protocols. In the problem, n cryptographers dine around a table. After finishing their meal, they are told that the meal has been paid for. The cryptographers are curious to know whether they are treated by an unknown generous funding agency or by a colleague among themselves. Being cryptographers, they respect their privacy very much. If a colleague paid, the cryptographers do not wish to identify the person. The dining cryptographers problem is to determine whether a funding agency or an anonymous cryptographer paid the meal.

The cryptographers can solve the problem as follows. Initially, each cryptographer selects a secret bit. She then obtains a shared secret bit with her left neighbor (say, by computing the parity of their two secret bits). Similarly, she obtains another secret bit shared with her right neighbor. Each cryptographer then announces a bit. If she did not pay the meal, she announces the parity of her two shared bits. If she paid, the complement of the parity of her two shared bits is announced. It can be shown that the parity of all announced bits is even if an unknown funding agency pays the bill. We verify this fact in the example.

Automated assume-guarantee reasoning with progressive witness analysis is able to verify 12 cryptographers within 1.6 hours. In fact, the verification of 13 cryptographers is complete within 4.2 hours. In contrast, monolithic symbolic model checking uses up 4GB of memory on 12 cryptographers. If we increase the memory bound to 8GB, the monolithic technique is able to verify the instance of 12 cryptographers in 15.8 hours but still cannot finish the instance of 13 cryptographers. Assume-guarantee reasoning with progressive witness analysis outperforms monolithic model checking for this example. On the other hand, assume-guarantee reasoning with simple witness analysis performs very disappointingly. It fails to verify any instance with more than nine cryptographers. Analyzing witnesses progressively is essential to take full advantages of assume-guarantee reasoning in the dining cryptographers problem.

Gate Control System. Designed for a Buddhist ceremony on a stage, the gate control system is used in LingShan Buddhist Palace in Jiangsu, China [37]. The system consists of five parts: the controller, lifting platform, push-pull unit, latch, and motors. The lifting platform, push-pull unit, and latch lift gates to and from the warehouse under the stage. A motor is attached to each gate and moves along a circular track. All these devices are operated by the controller. Figure 4 shows the system architecture.

Consider the opening scene of the ceremony, where every gate arises from the warehouse orderly and moves to the circular track. In order to bring gates from the warehouse underneath, the lifting platform elevates a gate from the warehouse to the stage. A push-pull unit then pushes the gate to the circular track. At the closing scene, a motor first moves a gate adjacent to the lifting platform along the track. A push-pull unit then pulls the gate from the circular track to the platform. Finally, the lifting platform descends the gate to the warehouse below. For safety reasons, a latch locks the lifting platform when the platform gets to a designated

Table 5: Gate Control System

property	gate1		gate2		lpf1		lpf2	
	peak	time	peak	time	peak	time	peak	time
NuSMV	0.561×10^6	0.49s	0.561×10^6	0.49s	0.884×10^6	1.13s	0.884×10^6	1.15s
AG1	0.702×10^6	1.03s	0.338×10^6	0.28s	0.471×10^6	4.36s	1.190×10^6	2.87s
AG*	0.370×10^6	0.45s	0.218×10^6	0.45s	0.282×10^6	0.62s	0.283×10^6	0.61s

property	latch1		latch2		ppu	
	peak	time	peak	time	peak	time
NuSMV	1.076×10^6	1.11s	1.076×10^6	1.12s	0.862×10^6	0.83s
AG1	0.467×10^6	0.82s	0.596×10^6	1.14s	1.014×10^6	1.91s
AG*	0.222×10^6	0.24s	0.189×10^6	0.49s	0.482×10^6	1.34s

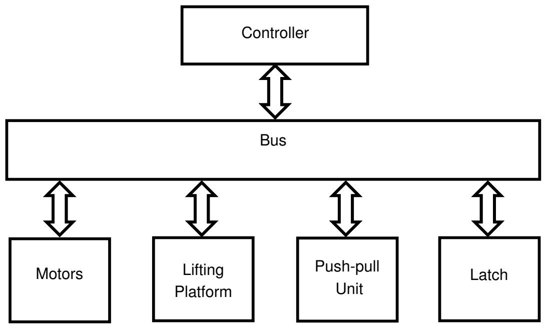

Figure 4: Structure of Gate Control System

position. Since all gates move on the same circular track, collision between gates must be avoided.

Table 5 gives the results. The property `gate1` states that a gate must stop when the user presses the stop button. The property `gate2` specifies that a gate must stop when it reaches the boundary. The properties `lpf1`, `latch1`, and `ppu` are similar to `gate1` except the devices are the lifting platform, latch, and push-pull unit, respectively. The properties `lpf2` and `latch2` are similar to `gate2` except the devices are the lifting platform and latch, respectively.

For all properties but `ppu`, assume-guarantee reasoning with progressive witness analysis is more efficient than the NuSMV monolithic verification algorithm. On average, our new technique attains 25.9% of speedup. Assume-guarantee reasoning with progressive witness analysis moreover improves memory efficiency. For the seven properties verified on the gate control system, assume-guarantee reasoning uses less peak BDD nodes. It saves 62.3% of memory than monolithic symbolic model checking on average.

Assume-guarantee reasoning with simple witness analysis on the other hand is unsatisfactory. It takes significantly more time than monolithic symbolic model checking in four properties (`gate1`, `lpf1`, `lpf2`, and `ppu`). The simple witness analysis algorithm moreover is not very memory efficient. It uses more peak BDD nodes than the monolithic verification algorithm in three properties (`gate1`, `lpf2`, and `ppu`).

7. CONCLUSION

Symbolic model checking and automated assume-guarantee reasoning are integrated by the progressive witness analysis algorithm proposed in this paper. Experimental results show that the our technique can improve not only the efficiency but also the capacity of symbolic model checking. Particularly, assume-guarantee reasoning with progressive witness analysis is more scalable than monolithic symbolic model checking in symmetric and parametric examples such as dining philosophers and dining cryptographers. Our experiments also demonstrate the importance of witness analysis in automated assume-guarantee reasoning with symbolic model checking. An ill-designed witness analysis algorithm can significantly impede the performance. Analyzing witnesses progressively is essential to integrate symbolic model checking with automated assume-guarantee reasoning.

In our experiments, we partition systems by trials and hence may not have the optimal partition. How to partition systems into components remains an important and challenging problem. Our technique currently finds contextual assumptions over all context variables. Since a property often depends on a subset of context variables, it suffices to find contextual assumptions over such variables. In [14], an algorithm inferring Boolean functions over relevant variables is proposed. A similar learning algorithm for BDD's may further improve the performance of symbolic assume-guarantee reasoning.

8. ACKNOWLEDGMENT

This work was supported by the Chinese National 973 Plan (2010CB328003), the NSF of China (61272001, 60903030, 91218302), the Chinese National Key Technology R&D Program (SQ2012BAJY4052), the Importation and Development of High-Caliber Talents Project of Beijing Municipal Institutions (YETP0167), the Tsinghua University Initiative Scientific Research Program, and the National Science Council of Taiwan (102-2221-E-001 -017 -).

9. REFERENCES

[1] R. Alur, P. Madhusudan, and W. Nam. Symbolic compositional verification by learning assumptions. In K. Etessami and S. K. Rajamani, editors, *Computer Aided Verification*, volume 3576 of *Lecture Notes in Computer Science*, pages 548–562. Springer, 2005.

[2] D. Angluin. Learning regular sets from queries and counterexamples. *Information and Computation*, 75(2):87–106, November 1987.

[3] D. Beyer, T. A. Henzinger, and V. Singh. Algorithms for interface synthesis. In W. Damm and H. Hermanns, editors, *Computer Aided Verification*, volume 4590 of *Lecture Notes in Computer Science*, pages 4–19. Springer, 2007.

[4] B. Bollig, P. Habermehl, C. Kern, and M. Leucker. Angluin-style learning of NFA. In C. Boutilier, editor,

International Joint Conferences on Artificial Intelligence, pages 1004–1009, 2009.

[5] A. J. N. Brad J. Cox. *Object-Oriented Programming: An Evolutionary Approach.* Addison-Wesley, 1991.

[6] N. H. Bshouty. Exact learning Boolean function via the monotone theory. *Information and Computation*, 123(1):146–153, 1995.

[7] J. R. Burch, E. M. Clarke, K. L. McMillan, D. L. Dill, and L. J. Hwang. Symbolic model checking: 10^{20} states and beyond. *Information and Computation*, 98(2):142–170, June 1992.

[8] S. Chaki, E. M. Clarke, N. Sinha, and P. Thati. Automated assume-guarantee reasoning for simulation conformance. In K. Etessami and S. K. Rajamani, editors, *Computer Aided Verification*, volume 3576 of *Lecture Notes in Computer Science*, pages 534–547. Springer, 2005.

[9] S. Chaki and O. Strichman. Optimized L^*-based assume-guarantee reasoning. In O. Grumberg and M. Huth, editors, *Tools and Algorithms for the Construction and Analysis of Systems*, volume 4424 of *Lecture Notes in Computer Science*, pages 276–291. Springer, 2007.

[10] D. Chaum. The dining cryptographers problem: Unconditional sender and recipient untraceability. *Journal of Cryptology*, 1(1):65–75, 1988.

[11] Y.-F. Chen, E. M. Clarke, A. Farzan, F. He, M.-H. Tsai, Y.-K. Tsay, B.-Y. Wang, and L. Zhu. Comparing learning algorithms in automated assume-guarantee reasoning. In *Leveraging Applications of Formal Methods, Verification and Validation (1)*, volume 6415 of *Lecture Notes in Computer Science*, pages 643–657. Springer, 2010.

[12] Y.-F. Chen, E. M. Clarke, A. Farzan, M.-H. Tsai, Y.-K. Tsay, and B.-Y. Wang. Automated assume-guarantee reasoning through implicit learning. In T. Touili, B. Cook, and P. Jackson, editors, *Computer Aided Verification*, volume 6174 of *Lecture Notes in Computer Science*, pages 511–526. Springer, 2010.

[13] Y.-F. Chen, A. Farzan, E. M. Clarke, Y.-K. Tsay, and B.-Y. Wang. Learning minimal separating DFA's for compositional verification. In S. Kowalewski and A. Philippou, editors, *Tools and Algorithms for the Construction and Analysis of Systems*, volume 5505 of *Lecture Notes in Computer Science*, pages 31–45. Springer, 2009.

[14] Y.-F. Chen and B.-Y. Wang. Learning boolean functions incrementally. In M. Parthasarathy and S. A. Seshia, editors, *Computer Aided Verification*, volume 7358 of *Lecture Notes in Computer Science*, pages 55–70. Springer, 2012.

[15] A. Cimatti, E. Clarke, F. Giunchiglia, and M. Roveri. NuSMV: a new Symbolic Model Verifier. In N. Halbwachs and D. Peled, editors, *Computer Aided Verification*, number 1633 in Lecture Notes in Computer Science, pages 495–499. Springer, 1999.

[16] E. M. Clarke, E. A. Emerson, and J. Sifakis. Model checking: Algorithmic verification and debugging. *Communications of ACM*, 52(11):74–84, 2009.

[17] E. M. Clarke, O. Grumberg, S. Jha, Y. Lu, and H. Veith. Counterexample-guided abstraction refinement for symbolic model checking. *Journal of the ACM*, 50(5):752–794, 2003.

[18] E. M. Clarke, O. Grumberg, and D. A. Peled. *Model Checking.* The MIT Press, Cambridge, Massachusetts, 1999.

[19] J. M. Cobleigh, G. S. Avrunin, and L. A. Clarke. Breaking up is hard to do: An evaluation of automated assume-guarantee reasoning. *ACM Trans. Software Engineering Methodology*, 17(2), 2008.

[20] J. M. Cobleigh, D. Giannakopoulou, and C. S. Păsăreanu. Learning assumptions for compositional verification. In H. Garavel and J. Hatcliff, editors, *Tools and Algorithms for the Construction and Analysis of Systems*, volume 2619 of *Lecture Notes in Computer Science*, pages 331–346. Springer, 2003.

[21] W.-P. de Roever, F. de Boer, U. Hanneman, J. Hooman, Y. Lakhnech, M. Poel, and J. Zwiers. *Concurrency Verification: Introduction to Compositional and Noncompositional Methods.* Number 54 in Cambridge Tracts in Theoretical Computer Science. Cambridge University Press, 2001.

[22] C. Flanagan and S. Qadeer. Predicate abstraction for software verification. In *Principles of Programming Languages*, pages 191–202. ACM, 2002.

[23] R. Gavaldà and D. Guijarro. Learning ordered binary decision diagrams. In K. P. Jantke, T. Shinohara, and T. Zeugmann, editors, *Algorithmic Learning Theory*, volume 997 of *Lecture Notes in Computer Science*, pages 228–238. Springer, 1995.

[24] M. Gheorghiu, D. Giannakopoulou, and C. S. Păsăreanu. Refining interface alphabets for compositional verification. In O. Grumberg and M. Huth, editors, *Tools and Algorithms for the Construction and Analysis of Systems*, volume 4424 of *Lecture Notes in Computer Science*, pages 292–307. Springer, 2007.

[25] D. Giannakopoulou and C. S. Păsăreanu. Special issue on learning techniques for compositional reasoning. *Formal Methods in System Design*, 32(3):173–174, 2008.

[26] A. Gupta, K. L. McMillan, and Z. Fu. Automated assumption generation for compositional verification. In W. Damm and H. Hermanns, editors, *Computer Aided Verification*, volume 4590 of *Lecture Notes in Computer Science*, pages 420–432. Springer, 2007.

[27] A. Gupta, K. L. McMillan, and Z. Fu. Automated assumption generation for compositional verification. *Formal Methods in System Design*, 32(3):285–301, 2008.

[28] T. A. Henzinger, R. Jhala, and R. Majumdar. Permissive interfaces. In *Foundations of Software Engineering*, ESEC/FSE-13, pages 31–40. ACM, 2005.

[29] M. J. Kearns and U. V. Vazirani. *An Introduction to Computational Learning Theory.* MIT Press, 1994.

[30] B. Meyer. *Object-Oriented Software Construction.* Prentice Hall, 1997.

[31] A. Nakamura. An efficient query learning algorithm for ordered binary decision diagrams. *Information and Computation*, 201(2):178–198, 2005.

[32] W. Nam, P. Madhusudan, and R. Alur. Automatic symbolic compositional verification by learning assumptions. *Formal Methods in System Design*, 32(3):207–234, 2008.

[33] H. Saídi and S. Graf. Construction of abstract state graphs with PVS. In Grumberg, editor, *Computer Aided Verification*, volume 1254 of *Lecture Notes in Computer Science*, pages 72–83. Springer, 1997.

[34] R. Singh, D. Giannakopoulou, and C. S. Pasareanu. Learning component interfaces with may and must abstractions. In T. Touili, B. Cook, and P. Jackson, editors, *Computer Aided Verification*, volume 6174 of *Lecture Notes in Computer Science*, pages 527–542. Springer, 2010.

[35] N. Sinha and E. M. Clarke. SAT-based compositional verification using lazy learning. In W. Damm and H. Hermanns, editors, *Computer Aided Verification*, volume 4590 of *Lecture Notes in Computer Science*, pages 39–54. Springer, 2007.

[36] L. Théry. Proof pearl: Revisiting the mini-Rubik in Coq. In O. A. Mohamed, C. Muñoz, and S. Tahar, editors, *Theorem Proving in Higher Order Logics*, volume 5170 of *Lecture Notes in Computer Science*, pages 310–319. Springer, 2008.

[37] R. Wang, M. Zhou, L. Yin, L. Zhang, J. Sun, G. Ming, and M. Bozga. Modeling and validation of PLC-controlled systems: A case study. In *Theoretical Aspects of Software Engineering*, pages 161–166. IEEE, 2012.