Comparing Learning Algorithms in Automated Assume-Guarantee Reasoning*

Yu-Fang Chen[1], Edmund M. Clarke[2], Azadeh Farzan[3], Fei He[4], Ming-Hsien Tsai[5], Yih-Kuen Tsay[5], Bow-Yaw Wang[1,4,6], and Lei Zhu[4]

[1] Academia Sinica, Taiwan
[2] Carnegie Mellon University, USA
[3] University of Toronto, Canada
[4] Tsinghua University, China
[5] National Taiwan University, Taiwan
[6] INRIA, France

Abstract. We compare two learning algorithms for generating contextual assumptions in automated assume-guarantee reasoning. The CDNF algorithm implicitly represents contextual assumptions by a conjunction of DNF formulae, while the OBDD learning algorithm uses ordered binary decision diagrams as its representation. Using these learning algorithms, the performance of assume-guarantee reasoning is compared with monolithic interpolation-based Model Checking in parametrized hardware test cases.

1 Introduction

Suppose one would like to verify whether the composition of M_0 and M_1 satisfies a property π. Consider the following assume-guarantee reasoning rule:

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

The rule states that it suffices to find a contextual assumption A such that the composition of M_0 and A satisfies the property, and that M_1 is simulated by A.

* This research was sponsored by the GSRC under contract no. 1041377 (Princeton University), National Science Foundation under contracts no. CCF0429120, no. CNS0926181, no. CCF0541245, and no. CNS0931985, Semiconductor Research Corporation under contract no. 2005TJ1366, General Motors under contract no. GM-CMUCRLNV301, Air Force (Vanderbilt University) under contract no. 18727S3, the Office of Naval Research under award no. N000141010188, the National Science Council of Taiwan projects no. NSC97-2221-E-001-003-MY3, no. NSC97-2221-E-001-006-MY3, no. NSC97-2221-E-002-074-MY3, and no. NSC99-2218-E-001-002-MY3, Natural Sciences and Engineering Research Council of Canada NSERC Discovery Award, Chinese National 973 Plan under grant no. 2010CB328003, the NSF of China under grants no. 60635020, 60903030 and 90718039, the FORMES Project within LIAMA Consortium, and the French ANR project SIVES ANR-08-BLAN-0326-01.

T. Margaria and B. Steffen (Eds.): ISoLA 2010, Part I, LNCS 6415, pp. 643–657, 2010.

If verifying $M_0 \| A \models \pi$ requires less resources than verifying $M_0 \| M_1 \models \pi$, the scalability of verification can be improved by finding such contextual assumptions. Indeed, complete information about M_1 may not be necessary for verifying the property π. Oftentimes, simple contextual assumptions are sufficient to establish properties of interest. Assume-guarantee reasoning offers the flexibility to simplify the verification problem with respect to properties. It is considered as a viable technique to alleviate the state explosion problem.

To effectively apply assume-guarantee reasoning, it is essential to construct a contextual assumption that fulfills the premises and admits efficient verification. By applying the L^* learning algorithm for finite automata [1] and devising a mechanical teacher to answer queries, the learning-based technique in [11] successfully infers contextual assumptions without human intervention. The scalability of automated assume-guarantee reasoning is further improved in [7]. Adopting an implicit representation and applying instead the CDNF learning algorithm for Boolean functions [3], the new technique is able to take advantages of the succinct representation and infer contextual assumptions of larger sizes. Preliminary experimental results show that automated assume-guarantee reasoning through implicit reasoning can outperform the monolithic interpolation-based Model Checking algorithm in some parametrized test cases [7].

Because of their potential applications in practice, several learning algorithms for Boolean functions have been developed [3,12,19]. In this paper, we investigate two of these learning algorithms in the context of automated assume-guarantee reasoning through implicit learning. We compare the performance of the CDNF algorithm [3] and a learning algorithm for ordered binary decision diagrams (OBDD's) [2,12] in automated assume-guarantee reasoning. Both learning algorithms use the same learning model proposed in [1]. The CDNF algorithm is based on the monotone theory and represents an arbitrary Boolean function as a conjunction of DNF formulae [3]. It learns any Boolean function with a polynomial number of queries in the number of variables, and the minimal CNF size and the minimal DNF size of the target function.

The OBDD learning algorithm, on the other hand, is based on the L^* algorithm [12]. For a fixed variable ordering, a valuation on n Boolean variables can be represented by a string in $\{0,1\}^n$. A set of valuations hence corresponds to a finite language. Since any finite language is regular, the L^* algorithm can infer the minimal deterministic finite automaton recognizing the satisfying valuations of any Boolean function. The minimal deterministic finite automaton in turn can be transformed to an OBDD. It is shown that any OBDD is learnable with a polynomial number of queries in the size of the target OBDD.

The two learning algorithms have very different characteristics. Subsequently, their practical costs in the context of assume-guarantee reasoning are not at all clear. To investigate this issue, we assess the effectiveness of both learning algorithms by an extensive set of parametrized test cases. Using two different learning algorithms to infer contextual assumptions, we compare the performance of automated assume-guarantee through implicit learning against the monolithic interpolation-based Model Checking algorithm in [18]. Five parametrized

hardware test cases are taken: the MSI cache coherence protocol [4], the PCI bus protocol [5], a simple bus control protocol [10], the Gigamax cache coherence protocol [9], and synchronous bus arbiters [17]. Each test case has over 15 experiments of different sizes. Three different algorithms are compared in each experiment. Our extensive experiments hopefully can give insights to research directions.

In [15], the CDNF algorithm is used to generate propositional loop invariants in sequential programs. The same learning algorithm is used in [7] to infer contextual assumptions for assume-guarantee reasoning. Applying algorithmic learning to generate contextual assumptions was first proposed in [11]. Following that work, many optimizations have been proposed (see, for example, [20,6,21,13,8]). These optimizations explicitly generate deterministic finite automata as contextual assumptions. In contrast, the work [7] implicitly infers nondeterministic finite automata as contextual assumptions and improves the scalability.

This paper is organized as follows. After preliminaries (Section 2), the learning model and automated assume-guarantee reasoning through implicit learning are reviewed in Section 3 and 4 respectively. They are followed by brief descriptions of the learning algorithms (Section 5). Section 6 gives the experimental results. We conclude in Section 7.

2 Preliminaries

$\mathbb{B} = \{\mathsf{F}, \mathsf{T}\}$ is the Boolean domain. Let $\mathbf{x}$ be a set of Boolean variables and $|\mathbf{x}|$ the size of $\mathbf{x}$. A *Boolean function* $\theta(\mathbf{x})$ *over* $\mathbf{x}$ is a function from $\mathbb{B}^{|\mathbf{x}|}$ to $\mathbb{B}$. We also define $\mathbf{x}'$ to be the set of Boolean variables $\{x' : x \in \mathbf{x}\}$.

A *valuation* $\nu : \mathbf{x} \to \mathbb{B}$ *over* $\mathbf{x}$ is a function from Boolean variables to truth values. Let $\phi(\mathbf{x})$ be a Boolean function over $\mathbf{x}$ and ν a valuation over $\mathbf{x}$. If $\mathbf{y} \subseteq \mathbf{x}$ is a set of Boolean variables, $\nu{\downarrow_{\mathbf{y}}}$ is the *restriction* of ν on $\mathbf{y}$. That is, $\nu{\downarrow_{\mathbf{y}}}$: $\mathbf{y} \to \mathbb{B}$ and $\nu{\downarrow_{\mathbf{y}}}\ (y) = \nu(y)$ for all $y \in \mathbf{y}$. We write $\phi[\nu]$ for the result of evaluating ϕ by replacing each $x \in \mathbf{x}$ with $\nu(x)$. Moreover, let $\psi(\mathbf{x}, \mathbf{x}')$ be a Boolean function over $\mathbf{x}$ and $\mathbf{x}'$. If ν and ν' are valuations over $\mathbf{x}$, we write $\psi[\nu, \nu']$ for the result of evaluating ψ by replacing each $x \in \mathbf{x}$ with $\nu(x)$ and each $x' \in \mathbf{x}'$ with $\nu'(x)$. For example, assume $\nu(x) = \mathsf{F}$ and $\nu'(x) = \mathsf{T}$. If $\phi(x) = \neg x$, $\phi[\nu] = \mathsf{T}$ and $\phi[\nu'] = \mathsf{F}$. If $\psi(x, x') = \neg x \wedge x'$, $\psi[\nu, \nu'] = \mathsf{T}$ and $\psi[\nu', \nu] = \mathsf{F}$.

A *transition system* $M = (\mathbf{x}, \iota(\mathbf{x}), \tau(\mathbf{x}, \mathbf{x}'))$ consists of its *state variables* $\mathbf{x}$, its *initial predicate* $\iota(\mathbf{x})$, and its *transition relation* $\tau(\mathbf{x}, \mathbf{x}')$. A *trace of* M $\alpha = \nu^0\nu^1 \cdots \nu^t$ is a finite sequence of valuations where ν^i is a valuation over $\mathbf{x}$, such that $\iota[\nu^0] = \mathsf{T}$ and $\tau[\nu^i, \nu^{i+1}] = \mathsf{T}$ for $0 \leq i < t$. Define $\mathit{Trace}(M) = \{\alpha : \alpha \text{ is a trace of } M\}$. If $\alpha = \nu^0\nu^1 \cdots \nu^t$ is a finite sequence of valuations over $\mathbf{x}$ and $\mathbf{y} \subseteq \mathbf{x}$, $\alpha{\downarrow_{\mathbf{y}}} = \nu^0{\downarrow_{\mathbf{y}}}\ \nu^1{\downarrow_{\mathbf{y}}} \cdots \nu^t{\downarrow_{\mathbf{y}}}$ is the *restriction* of α on $\mathbf{y}$.

Let $M = (\mathbf{x}, \iota_M(\mathbf{x}), \tau_M(\mathbf{x}, \mathbf{x}'))$ be a transition system. A *state predicate* $\pi(\mathbf{x})$ is a Boolean function over $\mathbf{x}$. We say M *satisfies* π (denoted by $M \models \pi$) if for any $\alpha = \nu^0\nu^1 \cdots \nu^t \in \mathit{Trace}(M)$, we have $\pi[\nu^i] = \mathsf{T}$ for $0 \leq i \leq t$. Given a transition system M and a state predicate π, the *invariant checking* problem is to decide whether M satisfies π. *Model Checking* is an automatic technique to solve the

invariant checking problem. When deciding whether $M \models \pi$, a Model Checking algorithm returns a witness if M does not satisfy π. A *witness* to $M \not\models \pi$ is a trace $\nu^0\nu^1 \cdots \nu^t$ of M such that $\pi(\nu^i) = \mathsf{T}$ for $0 \leq i < t$ but $\pi(\nu^t) = \mathsf{F}$.

Let $N = (\mathbf{x}, \iota_N(\mathbf{x}), \tau_N(\mathbf{x}, \mathbf{x}'))$ be a transition system. We say M *is simulated by* N or N *simulates* M (denoted by $M \preceq N$) if $\forall \mathbf{x}.\iota_M(\mathbf{x}) \implies \iota_N(\mathbf{x})$ and $\forall \mathbf{x}\mathbf{x}'.\tau_M(\mathbf{x}, \mathbf{x}') \implies \tau_N(\mathbf{x}, \mathbf{x}')$ hold. In words, M is simulated by N if the initial condition of M is stronger than that of N and every transition allowed in M is also allowed in N. Clearly, if $M \preceq N$, then $\mathit{Trace}(M) \subseteq \mathit{Trace}(N)$.

Let $\mathbf{x}_i$ be sets of Boolean variables for $i = 0, 1$ ($\mathbf{x}_i$'s are not necessarily disjoint). Consider $M_i = (\mathbf{x}_i, \iota_i(\mathbf{x}_i), \tau_i(\mathbf{x}_i, \mathbf{x}_i'))$ for $i = 0, 1$. The *composition* of M_0 and M_1 is the transition system $M_0 \| M_1 = (\mathbf{x}_0 \cup \mathbf{x}_1, \iota_0(\mathbf{x}_0) \wedge \iota_1(\mathbf{x}_1), \tau_0(\mathbf{x}_0, \mathbf{x}_0') \wedge \tau_1(\mathbf{x}_1, \mathbf{x}_1'))$. Note that for any finite sequence of valuations α over $\mathbf{x}_0 \cup \mathbf{x}_1$, $\alpha \in \mathit{Trace}(M_0 \| M_1)$ if and only if $\alpha{\downarrow_{\mathbf{x}_0}} \in \mathit{Trace}(M_0)$ and $\alpha{\downarrow_{\mathbf{x}_1}} \in \mathit{Trace}(M_1)$.

An *assume-guarantee reasoning rule* is of the form $\dfrac{\Theta_0 \cdots \Theta_m}{\Delta}$ where $\Theta_0, \ldots, \Theta_m$ are its *premises* and Δ its *conclusion*. An assume-guarantee reasoning rule is *sound* if its conclusion holds when its premises are fulfilled. A rule is *invertible* if its premises can be fulfilled when its conclusion holds. We use the following assume-guarantee reasoning rule throughout the paper:

Lemma 1. *Let $M_i = (\mathbf{x}_i, \iota_i(\mathbf{x}_i), \tau_i(\mathbf{x}_i, \mathbf{x}_i'))$ be transition systems for $i = 0, 1$, and π a state predicate over $\mathbf{x}_0 \cup \mathbf{x}_1$. The following rule is sound and invertible:*

$$\frac{M_0 \| A \models \pi \qquad M_1 \preceq A}{M_0 \| M_1 \models \pi} \tag{1}$$

where $A = (\mathbf{x}_1, \iota_A(\mathbf{x}_1), \tau_A(\mathbf{x}_1, \mathbf{x}_1'))$ is a transition system.

Let $M_i = (\mathbf{x}_i, \iota_i(\mathbf{x}_i), \tau_i(\mathbf{x}_i, \mathbf{x}_i'))$ be transition systems for $i = 0, 1$ and π a state predicate over $\mathbf{x}_0 \cup \mathbf{x}_1$, a transition system $A = (\mathbf{x}_1, \iota_A(\mathbf{x}_1), \tau_A(\mathbf{x}_1, \mathbf{x}_1'))$ such that $M_0 \| A \models \pi$ and $M_1 \preceq A$ is called a *contextual assumption* of M_0.

3 The Learning Model

Before reviewing the learning-based approach to inferring contextual assumptions, we briefly describe the learning model used in [1,3,7]. For any unknown *target* Boolean function $\lambda(\mathbf{x})$ over a fixed set of Boolean variables $\mathbf{x}$, an exact learning algorithm for Boolean functions computes a representation of $\lambda(\mathbf{x})$ by interacting with a teacher. The *teacher* knows the Boolean function $\lambda(\mathbf{x})$ and answers two types of queries made by the learning algorithm:

- *Membership query $\mathit{MEM}(\nu)$ for the target $\lambda(\mathbf{x})$,* where ν is a valuation over $\mathbf{x}$. If $\lambda[\nu] = \mathsf{T}$, the teacher answers *YES*; and *NO*, otherwise.
- *Equivalence query $\mathit{EQ}(\theta)$ for the target $\lambda(\mathbf{x})$,* where $\theta(\mathbf{x})$ is a Boolean function over $\mathbf{x}$. If the *conjecture* $\theta(\mathbf{x})$ is equivalent to the target Boolean function $\lambda(\mathbf{x})$, the teacher answers *YES*. Otherwise, the teacher provides a valuation ν over $\mathbf{x}$ where $\theta[\nu] \neq \lambda[\nu]$. The valuation ν serves as a *counterexample* to the equivalence query $\mathit{EQ}(\theta)$.

Assume $\lambda(x, y) = (x \wedge \neg y) \vee (\neg x \wedge y)$ is the target Boolean function over x and y. If the learning algorithm makes the query $MEM(\nu_0)$ where $\nu_0(x) = \nu_0(y) = \mathsf{F}$ (denoted by $\nu_0(xy) = \mathsf{FF}$), the teacher answers NO for $\lambda(\mathsf{F}, \mathsf{F}) = \mathsf{F}$. For a different valuation $\nu_1(xy) = \mathsf{TF}$, the teacher answers YES. Similarly, consider the equivalence query $EQ(x \vee y)$. The teacher should provide the valuation $\nu_2(xy) = \mathsf{TT}$ as a counterexample, since $\mathsf{T} \vee \mathsf{T} = \mathsf{T} \neq \mathsf{F} = \lambda(\mathsf{T}, \mathsf{T})$. For another equivalence query $EQ((x \vee \neg y) \wedge (\neg x \vee y))$, the teacher answers YES.

4 Learning a Contextual Assumption

In automated assume-guarantee reasoning through learning, one applies an exact learning algorithm to infer a contextual assumption that fulfills both premises of the assume-guarantee reasoning rule (1). In order to do so, a mechanical teacher is designed to answer queries from the learning algorithm. Assume $A = (\mathbf{x}_1, \iota_A(\mathbf{x}_1), \tau_A(\mathbf{x}_1, \mathbf{x}'_1))$ is a contextual assumption satisfying both premises. The teacher is required to resolve four types of queries:

- the membership query $MEM(\mu)$ for the target $\iota_A(\mathbf{x}_1)$;
- the membership query $MEM(\mu, \mu')$ for the target $\tau_A(\mathbf{x}_1, \mathbf{x}'_1)$;
- the equivalence query $EQ(\iota)$ for the target $\iota_A(\mathbf{x}_1)$; and
- the equivalence query $EQ(\tau)$ for the target $\tau_A(\mathbf{x}_1, \mathbf{x}'_1)$.

If A was known, it would be straightforward to design such a mechanical teacher. All queries can easily be resolved by evaluating or comparing the initial predicate or the transition relation of the purported contextual assumption. However, such a contextual assumption is yet to be inferred and current unknown to us. We thus look for a replacement in the design of the mechanical teacher.

In [7], the mechanical teacher simply uses M_1 in place of the unknown contextual assumption. Clearly, inferring M_1 as the contextual assumption in the assume-guarantee reasoning rule (1) is not beneficial: the first premise is precisely the conclusion when the contextual assumption A is M_1. However, several conjectures will be proposed while the learning algorithm is inferring M_1. If one of them satisfies both premises, it can serve as a contextual assumption and conclude the verification. Since contextual assumptions are not unique, one expects that another contextual assumption will be generated before M_1 is inferred.

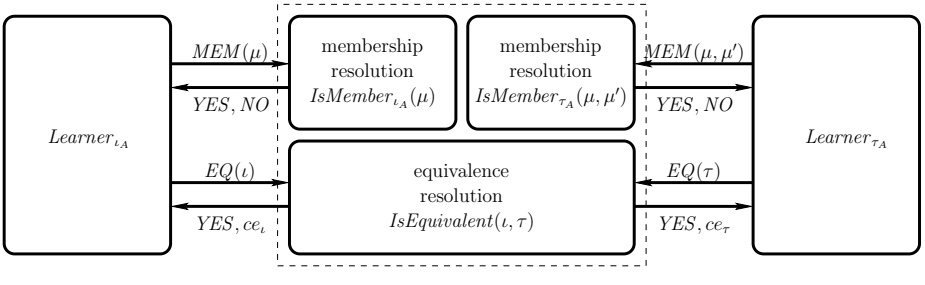

Fig. 1. Structure of Contextual Assumption Generator

We adopt the architecture of the mechanical teacher proposed in [7]. Recall that a contextual assumption $A = (\mathbf{x}_1, \iota_A(\mathbf{x}_1), \tau_A(\mathbf{x}_1, \mathbf{x}'_1))$ consists of two Boolean formulae. We hence deploy two instances of the learning algorithm (Figure 1): one infers the initial predicate $\iota_A(\mathbf{x}_1)$; the other infers the transition relation $\tau_A(\mathbf{x}_1, \mathbf{x}'_1)$. In the figure, the instances of the learning algorithm are shown on the sides. The instance $Learner_{\iota_A}$ is intended to compute the initial predicate $\iota_A(\mathbf{x}_1)$; the instance $Learner_{\tau_A}$ is to compute the transition relation $\tau_A(\mathbf{x}_1, \mathbf{x}'_1)$. The mechanical teacher is denoted by the dashed box in the middle.

The teacher consists of three query resolution algorithms. The algorithms $IsMember_{\iota_A}(\mu)$ and $IsMember_{\tau_A}(\mu, \mu')$ resolve membership queries for the initial predicate and the transition relation respectively (Algorithm 1). Since the mechanical teacher uses M_1 as the target, membership queries are resolved by evaluating the initial predicate or the transition relation of M_1 respectively.

Input: $MEM(\mu)$: a membership query for the target $\iota_A(\mathbf{x}_1)$
Output: *YES* or *NO*
`/* ` $\iota_1(\mathbf{x}_1)$ `is the initial predicate of` M_1 `*/`
if $\iota_1[\mu] = \mathsf{T}$ **then return** *YES* **else return** *NO*;

(a) $IsMember_{\iota_A}(\mu)$

Input: $MEM(\mu, \mu')$: a membership query for the target $\tau_A(\mathbf{x}_1, \mathbf{x}'_1)$
Output: *YES* or *NO*
`/* ` $\tau_1(\mathbf{x}_1, \mathbf{x}'_1)$ `is the transition relation of` M_1 `*/`
if $\tau_1[\mu, \mu'] = \mathsf{T}$ **then return** *YES* **else return** *NO*;

(b) $IsMember_{\tau_A}(\mu, \mu')$

Algorithm 1. Membership Query Resolution Algorithms

The algorithm $IsEquivalent(\iota, \tau)$ resolves both types of equivalence queries (Algorithm 2). The equivalence query resolution algorithm waits until the equivalence query $EQ(\iota)$ from $Learner_{\iota_A}$ and the equivalence query $EQ(\tau)$ from $Learner_{\tau_A}$ are available. It then checks the premise $M_1 \preceq C$ in the assume-guarantee reasoning rule (1) with $C = (\mathbf{x}_1, \iota(\mathbf{x}_1), \tau(\mathbf{x}_1, \mathbf{x}'_1))$. If $M_1 \not\preceq C$, a counterexample is returned to either $Learner_{\iota_A}$ or $Learner_{\tau_A}$ to refine the current conjecture. For instance, assume $\forall \mathbf{x}_1.\iota_1(\mathbf{x}_1) \Rightarrow \iota(\mathbf{x}_1)$ is false. There is a valuation μ such that $\iota_1[\mu] = \mathsf{T}$ and $\iota[\mu] = \mathsf{F}$. The equivalence query resolution algorithm returns μ to $Learner_{\iota_A}$ as the counterexample to the equivalence query $EQ(\iota)$. It then waits for another equivalence query $EQ(\iota')$ from $Learner_{\iota_A}$, and restarts with the new conjectured transition system $C' = (\mathbf{x}_1, \iota'(\mathbf{x}_1), \tau(\mathbf{x}_1, \mathbf{x}'_1))$.

Assume the equivalence query resolution algorithm has verified $M_1 \preceq C$. It then checks if the other premise $M_0 \| C \models \pi$ is fulfilled. If so, we have found a contextual assumption that establishes the property. Otherwise, there is a trace α witnessing $M_0 \| C \not\models \pi$. The equivalence query resolution algorithm then invokes $IsWitness(\alpha)$ to analyze the trace α.

Input: $EQ(\iota)$: an equivalence query for the target $\iota_A(\mathbf{x}_1)$; $EQ(\tau)$: an equivalence query for the target $\tau_A(\mathbf{x}_1, \mathbf{x}'_1)$
Output: *YES*, a counterexample to $EQ(\iota)$, or a counterexample to $EQ(\tau)$
let C be the transition system $(\mathbf{x}_1, \iota(\mathbf{x}_1), \tau(\mathbf{x}_1, \mathbf{x}'_1))$;
if $\iota_1(\mathbf{x}_1) \wedge \neg\iota(\mathbf{x}_1)$ *is satisfied by* μ **then**
  answer $EQ(\iota)$ with the counterexample μ;
  receive another equivalence query $EQ(\iota')$;
  call *IsEquivalent*(ι', τ);
if $\tau_1(\mathbf{x}_1, \mathbf{x}'_1) \wedge \neg\tau(\mathbf{x}_1, \mathbf{x}'_1)$ *is satisfied by* $\mu\mu'$ **then**
  answer $EQ(\tau)$ with the counterexample $\mu\mu'$;
  receive another equivalence query $EQ(\tau')$;
  call *IsEquivalent*(ι, τ');
if $M_0 \| C \models \pi$ **then**
  answer $EQ(\iota)$ with *YES*;
  answer $EQ(\tau)$ with *YES*;
  report "$M_0 \| M_1 \models \pi$";
else
  let α be a witness to $M_0 \| C \not\models \pi$;
  call *IsWitness*(α);
end

Algorithm 2. *IsEquivalent*(ι, τ)

The witness analysis algorithm *IsWitness*(α) checks if the restriction $\alpha{\downarrow_{\mathbf{x}_1}}$ is also a trace of M_1 (Algorithm 3). If so, α is in fact a witness to $M_0 \| M_1 \not\models \pi$. Otherwise, the restriction $\alpha{\downarrow_{\mathbf{x}_1}}$ must deviate from the initial predicate or the transition relation of M_1. The witness analysis algorithm therefore returns the deviation as a counterexample to either $EQ(\iota)$ or $EQ(\tau)$. It then waits for a new equivalence query and restarts the equivalence query resolution algorithm.

The correctness of the algorithm is established by the following properties [7].

Lemma 2 (soundness). *Let* $M_i = (\mathbf{x}_i, \iota_i(\mathbf{x}_i), \tau_i(\mathbf{x}_i, \mathbf{x}'_i))$ *be transition systems for* $i = 0, 1$, *and* $\pi(\mathbf{x})$ *a state predicate over* $\mathbf{x} = \mathbf{x}_0 \cup \mathbf{x}_1$.

1. *Let* $\iota(\mathbf{x}_1)$ *and* $\tau(\mathbf{x}_1, \mathbf{x}'_1)$ *be Boolean functions over* $\mathbf{x}_1$ *and* $\mathbf{x}_1 \cup \mathbf{x}'_1$ *respectively. If IsEquivalent*(ι, τ) *reports "*$M_0 \| M_1 \models \pi$*," then* $M_0 \| M_1 \models \pi$*;*
2. *Let* $\iota(\mathbf{x}_1)$ *and* $\tau(\mathbf{x}_1, \mathbf{x}'_1)$ *be Boolean functions over* $\mathbf{x}_1$ *and* $\mathbf{x}_1 \cup \mathbf{x}'_1$ *respectively. If IsEquivalent*(ι, τ) *reports "*$M_0 \| M_1 \not\models \pi$ *is witnessed by* α*," then* α *is a witness to* $M_0 \| M_1 \not\models \pi$.

Lemma 3 (completeness). *Let* $M_i = (\mathbf{x}_i, \iota_i(\mathbf{x}_i), \tau_i(\mathbf{x}_i, \mathbf{x}'_i))$ *be transition systems for* $i = 0, 1$, *and* $\pi(\mathbf{x})$ *a state predicate over* $\mathbf{x} = \mathbf{x}_0 \cup \mathbf{x}_1$.

1. *If* $M_0 \| M_1 \models \pi$, *then IsEquivalent*(ι, τ) *reports "*$M_0 \| M_1 \models \pi$*" for some Boolean functions* $\iota(\mathbf{x}_1)$ *and* $\tau(\mathbf{x}_1, \mathbf{x}'_1)$ *over* $\mathbf{x}_1$ *and* $\mathbf{x}_1 \cup \mathbf{x}'_1$ *respectively.*
2. *If* α *is a witness to* $M_0 \| M_1 \not\models \pi$, *then IsEquivalent*$(\iota, \tau)$ *reports "*$M_0 \| M_1 \not\models \pi$ *is witnessed by* α*" for some Boolean functions* $\iota(\mathbf{x}_1)$ *and* $\tau(\mathbf{x}_1, \mathbf{x}'_1)$ *over* $\mathbf{x}_1$ *and* $\mathbf{x}_1 \cup \mathbf{x}'_1$ *respectively.*

Input: α is a witness to $M_0 \| C \not\models \pi$
Output: a counterexample to $EQ(\iota)$, or a counterexample to $EQ(\tau)$
let $\alpha{\downarrow_{\mathbf{x}_1}} = \mu^0\mu^1 \cdots \mu^t$;
if $\iota_1[\mu^0] = \mathsf{F}$ **then**
    answer $EQ(\iota)$ with the counterexample μ^0;
    receive another equivalence query $EQ(\iota')$;
    call *IsEquivalent*(ι', τ);
for $i := 1$ **to** t **do**
    if $\tau_1[\mu^{i-1}, \mu^i] = \mathsf{F}$ **then**
        answer $EQ(\tau)$ with the counterexample $\mu^{i-1}\mu^i$;
        receive another equivalence query $EQ(\tau')$;
        call *IsEquivalent*(ι, τ');
end
report "$M_0 \| M_1 \not\models \pi$ is witnessed by α";

Algorithm 3. *IsWitness*(α)

Lemma 4 (termination). *Let $M_i = (\mathbf{x}_i, \iota_i(\mathbf{x}_i), \tau_i(\mathbf{x}_i, \mathbf{x}_i'))$ be transition systems for $i = 0, 1$, and $\pi(\mathbf{x})$ a state predicate over $\mathbf{x} = \mathbf{x}_0 \cup \mathbf{x}_1$. Suppose the learning algorithm infers an unknown target Boolean formula f with $t(|f|)$ queries. The mechanical teacher reports "$M_0 \| M_1 \models \pi$" or "$M_0 \| M_1 \not\models \pi$ is witnessed by α" with at most $t(|\iota_1|) + t(|\tau_1|)$ queries.*

5 Exact Learning Algorithms for Boolean Functions

Thanks to the interface defined in the learning model, the mechanical teacher presented in Section 4 is independent of the underlying learning algorithm. As long as a learning algorithm uses the same learning model, it can be instantiated as $Learner_{\iota_A}$ and $Learner_{\tau_A}$ to infer contextual assumptions. There are in fact several learning algorithms for Boolean functions [3,12,19]. Here, we are most interested in the CDNF algorithm [3] and the OBDD learning algorithm [12].

5.1 The CDNF Algorithm

The CDNF algorithm is an exact learning algorithm for Boolean functions [3]. It represents any unknown target Boolean function in the conjunctive-disjunctive normal form.[1] The CDNF algorithm works iteratively. In each iteration, it first uses membership queries to construct a conjecture in CDNF. After a conjecture is built, the CDNF algorithm poses an equivalence query to check if it has inferred the unknown target. If not, the current conjecture is refined by the counterexample. Depending on whether the conjecture need be weakened or strengthened, the learning algorithm either modifies DNF formulae of the current conjecture, or adds a DNF formula to the conjecture respectively.

Initially, the CDNF algorithm starts with the degenerated conjecture T, and makes the equivalence query $EQ(\mathsf{T})$. If T is not the target, the CDNF algorithm

[1] A Boolean formula is in the *conjunctive-disjunctive normal form (CDNF)* if it is a conjunction of DNF formulae.

adds the first conjunct and therefore strengthens the initial conjecture. More generally, let us say that a counterexample of an equivalence query is *positive* if the target evaluates to T under the counterexample; it is *negative* if the target evaluates to F. When the CDNF algorithm obtains a positive counterexample, it weakens conjuncts of the current conjecture by adding a conjunctive clause to DNF formulae in the conjecture. When the CDNF algorithm obtains a negative counterexample, it strengthens the current conjecture by adding a DNF formula to the conjecture. The first equivalence query simply initiates the iterations by strengthening the degenerated conjunction.

Let $\lambda(\mathbf{x})$ be a Boolean function over $\mathbf{x}$, $|\lambda(\mathbf{x})|_{DNF}$ and $|\lambda(\mathbf{x})|_{CNF}$ denote the sizes of $\lambda(\mathbf{x})$ in minimal disjunctive and conjunctive normal forms respectively. Under the learning model in Section 3, the CDNF algorithm computes a representation for any target Boolean function $\lambda(\mathbf{x})$ with a polynomial number of queries in $|\lambda(\mathbf{x})|_{DNF}$, $|\lambda(\mathbf{x})|_{CNF}$, and $|\mathbf{x}|$ [3].

5.2 A Learning Algorithm for Ordered Binary Decision Diagrams

Fix a variable ordering on Boolean variables $\mathbf{x}$. A valuation over $\mathbf{x}$ can be represented by a string in $\{0,1\}^{|\mathbf{x}|}$. For any Boolean function $\lambda(\mathbf{x})$, its satisfying valuations hence correspond to a finite language. Moreover, an OBDD for $\lambda(\mathbf{x})$ can be seen as a recognizer for the finite language of satisfying valuations. Observe that the structure of the OBDD for $\lambda(\mathbf{x})$ is in fact similar to the minimal deterministic finite automaton for the language of satisfying valuations [16]. Subsequently, one may infer an unknown OBDD by the L^* algorithm. This idea has been explored in an exact learning algorithm for OBDD [12]. Under the learning model in Section 3, the OBDD learning algorithm computes any target OBDD d with a polynomial number of queries in the size of d. By Shannon's expansion, we obtain another exact learning algorithm for Boolean functions.

The OBDD learning algorithm behaves very differently from the CDNF algorithm. As described above, the CDNF algorithm starts with the equivalence query $EQ(\mathsf{T})$. On the other hand, the OBDD learning algorithm almost always starts with the equivalence query $EQ(\mathsf{F})$. Starting from the empty valuation, the L^* algorithm builds its first conjecture by making membership queries on extensions of the empty valuation. If all valuations of length less than two are rejected, the L^* algorithm will build the minimal finite automaton recognizing the empty language as its conjecture. Recall that any satisfying valuation for Boolean functions over n variables must have length n, and that the empty set of satisfying valuations corresponds to the Boolean function F. The OBDD learning algorithm subsequently always starts with the equivalence query $EQ(\mathsf{F})$ when there is more than one Boolean variable.

In our settings, starting with the equivalence query $EQ(\mathsf{F})$ may impede the performance. After $Learner_{\iota_A}$ and $Learner_{\tau_A}$ make their first equivalence queries $EQ(\mathsf{F})$, the equivalence query resolution algorithm has the transition system $C_\perp = (\mathbf{x}_1, \mathsf{F}, \mathsf{F})$. Since $M_1 \not\preceq C_\perp$, it will ask the learning algorithm to weaken both conjectures. In fact, the OBDD learning algorithm sometimes weakens too conservatively and infers M_1 as the contextual assumption.

Input: $MEM(\mu)$: a membership query for the target $\lambda(\mathbf{x})$
Output: *YES* or *NO*
if *teacher's answer to* $MEM(\mu)$ *is YES* **then return** *NO* **else return** *YES*

(a) Inverted Membership Query

Input: $EQ(\theta)$: a membership query for the target $\lambda(\mathbf{x})$
Output: *YES* or a counterexample to $EQ(\theta)$
return *teacher's answer to* $EQ(\neg\theta)$

(b) Inverted Equivalence Query

Algorithm 4. Inverted Queries

One simple way to avoid this problem is to invert queries from the OBDD learning algorithm (Algorithm 4). The main idea is to let the learning algorithm infer the negation of the target Boolean function. When the OBDD learning algorithm makes a membership query, we return *NO* if the teacher answers *YES*. Otherwise, we return *YES*. When the OBDD learning algorithm makes an equivalence query, we ask the teacher if the negation of the conjecture is the target. If not, we forward teacher's counterexample to the learning algorithm. With this simple translation, the first equivalence query $EQ(\mathsf{F})$ from the OBDD learning algorithm is converted to the equivalence query $EQ(\mathsf{T})$ as desired.

6 Experiments

We have implemented a prototype of the mechanical teacher in OCaml. Our implementation uses the OCaml thread library. Each instance of the learning algorithm is executed in a separate thread, and the equivalence query resolution algorithm is executed in a third thread. MiniSat 2 (version 070721) is used to evaluate Boolean functions in Algorithm 1, and check the simulation relation in Algorithm 2. For monolithic Model Checking, we implement the interpolation-based algorithm in [18]. Interpolants are computed by instrumenting MiniSat 2. The interpolation-based Model Checking algorithm is also used in the equivalence query resolution algorithm (Algorithm 2).

We report the following five test cases: the MSI cache coherence protocol [5], the PCI bus protocol [4], a simple bus control protocol [10], the Gigamax cache coherence protocol [9], and synchronous bus arbiters [17] . Each test case has experiments parametrized by the number of nodes. Let $M_1, \ldots, M_n$ be nodes and π a state predicate. We verify $M_1 \| \cdots \| M_n \models \pi$ in an experiment with n nodes. An experiment with n nodes is divided into different partitions in n trials. We apply the following assume-guarantee reasoning rule in the i-th trial:

$$\frac{(M_1 \| \cdots \| M_{i-1} \| M_{i+1} \| \cdots M_n) \| A \models \pi \qquad M_i \preceq A}{(M_1 \| \cdots \| M_{i-1} \| M_{i+1} \| \cdots M_n) \| M_i \models \pi}$$

In each trial, we use the CDNF algorithm and the OBDD learning algorithm to generate a contextual assumption A to verify $M_1 \| \cdots \| M_n \models \pi$. We choose the best result among the n trials and compare it with monolithic Model Checking. All experimental results are collected on a server with 8 Intel Xeon 2.0GHz processors. Each experiment is carried out on a dedicated core with 4GB memory.

MSI Cache Coherence Protocol. In the MSI cache coherence protocol, a memory is shared among n nodes [5]. Each node has a cache. A bus connects the memory and caches of the nodes. When a node accesses a memory cell, it reads the cell from the bus and keeps a copy in its cache. Several copies of the same memory cell can be kept in different nodes. The MSI protocol ensures data coherence by keeping each cache in one of the three states: Modified, Shared, and Invalid [14]. Two properties are verified in the experiments with 4 to 20 nodes (Figure 2). The property `master1` specifies that at most one node can be the bus master. The other property `m0s1m1` states that if the data at a memory location is modified by a node, it cannot be shared or modified by another node at the same time.

nodes	4	5	6	7	8	9	10	11	12
monolithic	3s	5s	9s	13s	15s	33s	44s	1m41s	1m48s
cdnf	**0s**	**0s**	**2s**	**1s**	**3s**	**6s**	**4s**	**7s**	**7s**
bdd	**0s**	**0s**	**2s**	2s	**3s**	**6s**	5s	**7s**	8s

nodes	13	14	15	16	17	18	19	20	avg
monolithic	54s	1m30s	1m51s	50s	3m54s	5m38s	5m32s	5m16s	1m49s
cdnf	**11s**	**18s**	**7s**	**16s**	**48s**	**28s**	**12s**	1m12s	**14s**
bdd	**11s**	**18s**	8s	**16s**	56s	**28s**	14s	**1m10s**	15s

(a) Results for the Property `master1`

nodes	4	5	6	7	8	9	10	11	12
monolithic	**55s**	5m54s	46s	4m21s	22m19s	2m7s	2m37s	2m29s	9m49s
cdnf	2m14s	48s	54s	1m38s	1m18s	1m17s	2m43s	2m35s	2m43s
bdd	2m10s	**22s**	**44s**	**1m35s**	**1m14s**	**1m14s**	**2m41s**	**46s**	**2m38s**

nodes	13	14	15	16	17	18	19	20	avg
monolithic	6m15s	2m44s	11m35s	4m7s	18m55s	9m33s	10m12s	9m31s	7m18s
cdnf	2m17s	2m9s	3m4s	2m44s	2m51s	2m35s	3m45s	4m36s	2m22s
bdd	**2m14s**	**2m8s**	**3m2s**	**2m29s**	**2m50s**	**2m32s**	**3m26s**	**4m13s**	**2m8s**

(b) Results for the Property `m0s1m1`

Fig. 2. Experimental Results for the MSI Protocol

In both figures, we show the verification time of the monolithic interpolation-based Model Checking (*monolithic*), the verification time of assume-guarantee reasoning using the CDNF algorithm (*cdnf*), and those using the OBDD learning algorithm (*bdd*). We also identify the best algorithm in each experiment.

For both properties, assume-guarantee reasoning outperforms monolithic Model Checking consistently. Between the two learning algorithms used in assume-guarantee reasoning, their performances are almost indistinguishable for the property `master1`. The OBDD learning algorithm wins the CDNF algorithm in all experiments in the property `m0s1m1`. In the experiment with 11 nodes, the contextual assumption generated by the OBDD learning algorithm concludes the verification in less a minutes. The CDNF algorithm, on the other hand, takes more than two and a half minutes to verify the same property. It is in fact slower than monolithic Model Checking in this experiment.

PCI. This example models the PCI bus protocol with two levels of arbiters controlling data transmission [4]. For $2n$ PCI devices, we create a first-level arbiter and n second-level arbiters. The first-level arbiter connects all second-level arbiters. Each second-level arbiter connects two devices. When a device wants to start a transaction, it first requests the permission from its second-level arbiter. The second-level arbiter then selects a request between its devices. The first-level arbiter in turn grants the permission to the selected request from one of the second-level arbiters. We check that the first two nodes do not consider the bus to be idle at the same time. Figure 3 gives the experimental results. Assume-guarantee reasoning significantly outperforms monolithic Model Checking in this case. The difference between both learning algorithms is however negligible except for the experiment with 17 nodes. The CDNF algorithm proves the property in 37 seconds and wins the OBDD learning algorithm by 8 seconds. Monolithic Model Checking is unable to conclude the verification in two and a half minutes.

nodes	4	5	6	7	8	9	10	11	12
monolithic	15s	15s	25s	34s	46s	49s	1m2s	1m20s	1m17s
cdnf	**6s**	**10s**	**11s**	**14s**	**16s**	**17s**	**22s**	**24s**	28s
bdd	**6s**	**10s**	**11s**	**14s**	**16s**	**17s**	**22s**	27s	**24s**
nodes	13	14	15	16	17	18	19	20	avg
monolithic	1m25s	1m40s	1m48s	1m54s	2m45s	2m24s	2m34s	3m17s	1m26s
cdnf	**30s**	**33s**	**27s**	30s	**37s**	**40s**	**54s**	**50s**	**26s**
bdd	31s	34s	28s	**29s**	45s	41s	55s	**50s**	27s

Fig. 3. Experimental Results for PCI

Bus Control Protocol. In this bus control protocol, several nodes are attached to the bus [10]. Each node is assigned to a unique priority. A counter is used to decide the ownership of the bus. The node with priority p can send data when the counter has value p. If a node sends data, the counter is reset. Otherwise, the counter is incremented by one. We check that the first node cannot send data on the bus together with any other node at the same time. Figure 4 gives the results for the experiments with 34 to 50 nodes. Assume-guarantee reasoning clearly outperforms monolithic Model Checking. The CDNF algorithm and the OBDD learning algorithm win 10 and 5 of the experiments respectively. They are tied at the first place for the remaining 2 experiments.

nodes	34	35	36	37	38	39	40	41	42
monolithic	32s	33s	32s	33s	49s	2m25s	41s	40s	1m28s
cdnf	**21s**	**18s**	**21s**	**22s**	**21s**	**25s**	34s	**28s**	32s
bdd	**21s**	22s	25s	24s	25s	26s	**28s**	31s	**25s**

nodes	43	44	45	46	47	48	49	50	avg
monolithic	52s	52s	1m0s	54s	54s	1m12s	47s	15m42s	2m33s
cdnf	34s	**31s**	**33s**	**38s**	38s	**39s**	50s	**28s**	**43s**
bdd	**31s**	36s	**33s**	43s	**26s**	40s	**42s**	42s	44s

Fig. 4. Experimental Results for the Bus Control Protocol

nodes	21	22	23	24	25	26	27	28	29
monolithic (sec)	4.827	5.471	5.885	6.236	7.030	8.020	8.380	9.562	10.329
cdnf (sec)	**4.656**	**5.150**	**5.517**	**6.099**	6.789	**7.064**	8.213	8.958	**9.838**
bdd (sec)	4.857	5.328	5.741	6.339	**6.696**	7.693	**8.085**	**8.814**	10.086

nodes	30	31	32	33	34	35	36	37	avg
monolithic (sec)	11.503	12.107	13.175	14.145	14.890	15.791	17.009	18.518	14.524
cdnf (sec)	**10.803**	**11.579**	**12.904**	**13.977**	**14.689**	**15.658**	**16.962**	**17.966**	**14.065**
bdd (sec)	11.399	11.946	13.010	14.003	15.267	16.091	17.256	18.284	14.324

Fig. 5. Experimental Results for the Gigamax Cache Coherence Protocol

Gigamax. In the Gigamax cache coherence protocol, several processors and a memory is attached to a bus [9]. Each processor has a local cache. A local cache can be in the Invalid, Shared, or the Owned state. Among the memory and processors, at most one can be the bus master and issues commands on the bus. When a processor is the bus master and issues a ReadOwned command, its local cache state becomes Owned. A processor can write to the bus if its local cache state is Owned. For this test case, we consider the experiments with 21 to 37 nodes and verify that the memory is written by at most one processor. The results are shown in Figure 5. The CDNF algorithm outperforms the other two in 14 experiments whereas the OBDD learning algorithm wins the remaining 3 experiments with a small margin. The OBDD learning algorithm performs just slightly better than monolithic Model Checking on average.

Synchronous Bus Arbiters. The synchronous bus arbiter is a bus arbitration protocol for synchronous circuits [17]. In this protocol, n nodes are connected in a ring. A token is passed around the nodes. A node can request and acknowledge the token from the node next to it. The node having the token has the exclusive right to access the bus. For this test case, we check that there is at most one node can access to the bus. Figure 6 shows the results. Assume-guarantee reasoning and monolithic Model Checking perform almost identically on smaller experiments. For experiments with more than 12 nodes, assume-guarantee reasoning is slightly better. Monolithic Model Checking however is able to finish the experiment with 20 nodes by 40 seconds. Subsequently, monolithic Model Checking is comparable to assume-guarantee reasoning on average.

nodes	4	5	6	7	8	9	10	11	12
monolithic	**0s**	**1s**	**3s**	**5s**	**10s**	**18s**	33s	**54s**	1m38s
cdnf	**0s**	**1s**	**3s**	**5s**	**10s**	**18s**	**32s**	**54s**	**1m24s**
bdd	**0s**	**1s**	**3s**	**5s**	**10s**	**18s**	33s	55s	1m30s
nodes	13	14	15	16	17	18	19	20	avg
monolithic	**2m11s**	3m33s	5m9s	7m18s	10m11s	14m0s	19m13s	**25m17s**	5m20s
cdnf	2m13s	**3m20s**	**4m54s**	**7m8s**	**10m10s**	13m52s	**19m5s**	25m57s	**5m18s**
bdd	2m16s	3m27s	5m3s	7m21s	**10m10s**	**13m50s**	19m11s	26m4s	5m21s

Fig. 6. Experimental Results for Synchronous Bus Arbiters

7 Conclusion

Using two exact learning algorithms for Boolean functions, the performance of assume-guarantee reasoning is compared against the monolithic interpolation-based Model Checking algorithm in this paper. Our experiments show that automated assume-guarantee reasoning through implicit learning can outperform monolithic Model Checking in some parametrized hardware test cases. For the OBDD learning algorithm, we demonstrate a simple technique to invert queries from the learning algorithms. The technique improves the performance of the OBDD learning algorithm in assume-guarantee reasoning.

In three of the five test cases, assume-guarantee reasoning significantly improves the average verification time. The compositional technique slightly outperforms monolithic Model Checking in the remaining two test cases. Between the two learning algorithms, the difference however is marginal except the property `m0s1m1` in the MSI protocol. The OBDD learning algorithm clearly dominates the CDNF algorithm in this case. On the other hand, the CDNF algorithm performs slightly better than the OBDD learning algorithm in all other cases.

The simple modification of the OBDD learning algorithm shows that learning algorithms may not be trivially applied in the learning-based approach to assume-guarantee reasoning. Further optimizations are needed in this application domain. Particularly, the performances of the CDNF algorithm and the OBDD learning algorithm are sensitive to variable orderings. More research on this regard may further improve the scalability of assume-guarantee reasoning.

References

1. Angluin, D.: Learning regular sets from queries and counterexamples. Information and Computation 75(2), 87–106 (1987)
2. Bryant, R.: Graph-based algorithms for Boolean-function manipulation. IEEE Transaction on Computers C-35(8) (1986)
3. Bshouty, N.H.: Exact learning boolean function via the monotone theory. Information and Computation 123(1), 146–153 (1995)
4. Campos, S.V.A., Clarke, E.M., Marrero, W.R., Minea, M.: Verifying the performance of the PCI local bus using symbolic techniques. In: ICCD, pp. 72–78 (1995)

5. Cantin, J.F., Lipasti, M.H., Smith, J.E.: Dynamic verification of cache coherence protocols. In: Workshop on Memory Performance Issues (2001)
6. Chaki, S., Strichman, O.: Optimized L^*-based assume-guarantee reasoning. In: Grumberg, O., Huth, M. (eds.) TACAS 2007. LNCS, vol. 4424, pp. 276–291. Springer, Heidelberg (2007)
7. Chen, Y.F., Clarke, E.M., Farzan, A., Tsai, M.H., Tsay, Y.K., Wang, B.Y.: Automated assume-guarantee reasoning through implicit learning. In: Touili, T., Cook, B., Jackson, P. (eds.) CAV 2010. LNCS, vol. 6174. Springer, Heidelberg (2010)
8. Chen, Y.F., Farzan, A., Clarke, E.M., Tsay, Y.K., Wang, B.Y.: Learning minimal separating DFA's for compositional verification. In: Kowalewski, S., Philippou, A. (eds.) TACAS. LNCS, vol. 5505, pp. 31–45. Springer, Heidelberg (2009)
9. Cimatti, A., Clarke, E., Giunchiglia, F., Roveri, M.: NuSMV: a new Symbolic Model Verifier. In: Halbwachs, N., Peled, D.A. (eds.) CAV 1999. LNCS, vol. 1633, pp. 495–499. Springer, Heidelberg (1999)
10. Clarke, E.M., Kröning, D.: SMV example: Bus protocol, PowerPoint file (2002)
11. Cobleigh, J.M., Giannakopoulou, D.: Păsăreanu, C.S.: Learning assumptions for compositional verification. In: Garavel, H., Hatcliff, J. (eds.) TACAS 2003. LNCS, vol. 2619, pp. 331–346. Springer, Heidelberg (2003)
12. Gavaldà, R., Guijarro, D.: Learning ordered binary decision diagrams. In: Zeugmann, T., Shinohara, T., Jantke, K.P. (eds.) ALT 1995. LNCS, vol. 997, pp. 228–238. Springer, Heidelberg (1995)
13. Gupta, A., McMillan, K.L., Fu, Z.: Automated assumption generation for compositional verification. Formal Methods in System Design 32(3), 285–301 (2008)
14. Handy, J.: The Cache Memory Book. Academic Press, London (1998)
15. Jung, Y., Kong, S., Wang, B.Y., Yi, K.: Deriving invariants in propositional logic by algorithmic learning, decision procedure, and predicate abstraction. In: Barthe, G., Hermenegildo, M. (eds.) VMCAI 2010. LNCS, vol. 5944, pp. 180–196. Springer, Heidelberg (2010)
16. Kimura, S., Clarke, E.M.: A parallel algorithm for constructing binary decision diagrams. In: ICCD, pp. 220–223 (1990)
17. McMillan, K.L.: The SMV system, symbolic model checking - an approach. Technical Report CMU-CS-92-131, Carnegie Mellon University (1992)
18. McMillan, K.L.: Interpolation and SAT-based model checking. In: Hunt Jr., W.A., Somenzi, F. (eds.) CAV 2003. LNCS, vol. 2725, pp. 1–13. Springer, Heidelberg (2003)
19. Nakamura, A.: An efficient query learning algorithm for ordered binary decision diagrams. Information and Computation 201(2), 178–198 (2005)
20. Nam, W., Madhusudan, P., Alur, R.: Automatic symbolic compositional verification by learning assumptions. Formal Methods in System Design 32(3), 207–234 (2008)
21. Sinha, N., Clarke, E.M.: SAT-based compositional verification using lazy learning. In: Damm, W., Hermanns, H. (eds.) CAV 2007. LNCS, vol. 4590, pp. 39–54. Springer, Heidelberg (2007)