Automated Assume-Guarantee Reasoning through Implicit Learning*

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

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

Abstract. We propose a purely implicit solution to the contextual assumption generation problem in assume-guarantee reasoning. Instead of improving the L^* algorithm — a learning algorithm for *finite automata*, our algorithm computes implicit representations of contextual assumptions by the CDNF algorithm — a learning algorithm for *Boolean functions*. We report three parametrized test cases where our solution outperforms the monolithic interpolation-based Model Checking algorithm.

1 Introduction

Assume-guarantee reasoning is a divide-and-conquer technique to alleviate the state explosion problem in formal verification. Let M be a transition system and π a predicate on states of M. We write $M \models \pi$ to denote that all reachable states of M satisfy the state predicate π. The composition of transition systems M and M' is denoted by $M \| M'$. Moreover, $M \preceq M'$ means that M is simulated by M'. Consider the following assume-guarantee reasoning rule:

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

In order to prove that the composition of M_0 and M_1 satisfies π, it suffices to find a transition system A such that the composition of M_0 and A satisfies the state

* 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, no. NSC99-2218-E-001-002-MY3, and iCAST under contract no. 1010717, Natural Sciences and Engineering Research Council of Canada NSERC Discovery Award, the FORMES Project within LIAMA Consortium, and the French ANR project SIVES ANR-08-BLAN-0326-01.

T. Touili, B. Cook, and P. Jackson (Eds.): CAV 2010, LNCS 6174, pp. 511–526, 2010.

predicate π, and that M_1 is simulated by A. Informally, the transition system A captures necessary assumptions about the context of M_0 to guarantee π. We thus call A a contextual assumption. The contextual assumption generation problem is to compute a contextual assumption in an assume-guarantee reasoning rule.

We address the contextual assumption generation problem in this paper. In [11], the problem is formulated as an automata learning problem. The authors apply the L^* algorithm [1] to generate a deterministic finite automaton as the contextual assumption. In contrast to previous works [14,13,7,5,23,20,11], our solution does not rely on the L^* algorithm. Instead, we use the CDNF algorithm [4] to generate Boolean functions that implicitly represent contextual assumptions in assume-guarantee reasoning. One can think of the relation between our approach and L^*-based techniques as very similar to the relation between implicit and explicit Model Checking. Succinct implicit representations give our algorithm advantages in generating contextual assumptions of a moderate size. They hence make our solution more scalable and applicable.

Our new technique directly computes implicit representations of contextual assumptions by applying the CDNF algorithm [4]. The CDNF algorithm is an exact learning algorithm for arbitrary Boolean functions. It assumes an active learning model similar to that in the L^* algorithm [1]. In its learning model, a membership query asks a teacher if a valuation satisfies the target Boolean function. An equivalence query asks if a conjecture is equivalent to the target Boolean function. If not, the teacher should give a counterexample so that the learning algorithm can refine the conjecture. The CDNF algorithm is a feasible learning algorithm. It infers any target Boolean function with a polynomial number of queries in the size of the target function and the number of variables [4].

In [11], all components and the contextual assumption were modeled as finite automata. The contextual assumption generation problem was solved by learning a deterministic finite automaton as the contextual assumption. In contrast, we view the problem as a Boolean function learning problem. In our setting, transition systems and hence contextual assumptions are implicitly represented by Boolean functions. The simulation relation $M_1 \preceq A$ in the assume-guarantee reasoning rule gives a simple characterization of the Boolean functions representing the transition system M_1 and a contextual assumption A. We thus exploit the information to resolve membership queries. Moreover, the premise $M_0 \| A \models \pi$ in the assume-guarantee reasoning rule further characterizes the Boolean functions representing the transition system M_0 and the contextual assumption A. This allows us to resolve equivalence queries in our algorithm.

It is important to note that our algorithm is not an optimization of the explicit L^* algorithm in any way. Instead, our algorithm simply generates contextual assumptions implicitly by employing an exact learning algorithm for Boolean functions. The most significant advantage of our solution is its scalability. This can be observed in two aspects. Recall that the L^* algorithm requires a polynomial number of queries in the *number of states* of the target finite automaton [1,21]. The CDNF algorithm, on the other hand, requires a polynomial number of queries in the *number of Boolean variables* of the target Boolean

function [4]. Since implicit representations obtained in our algorithm can be exponentially more succinct than explicit ones obtained in automata-theoretic algorithms, our solution can be exponentially better than explicit algorithms.

Comparing the qualities of generated contextual assumptions, our solution is also favorable. Most existing automata-theoretic algorithms are based on variants of the L^* algorithm [1,21], they inherently generate deterministic finite automata as contextual assumptions. In contrast, contextual assumptions generated by our algorithm are represented by general Boolean functions. In general, they are nondeterministic finite automata in an economic representation. Since nondeterministic finite automata can be exponentially more succinct than deterministic ones, our algorithm can generate contextual assumptions with exponentially less states than those generated by L^*-based algorithms. Even though implicit representations have been used in optimizing the L^* algorithm [23,13,20], our new implicit solution can still outperform these optimizations.

In [17], the CDNF algorithm is used to generate propositional loop invariants in sequential programs. The idea of using the L^* algorithm to learn contextual assumptions for assume-guarantee reasoning was first proposed in [11]. Following this work, there have been results for other assume-guarantee rules [2,20], symbolic implementations [20], various optimization techniques [6,23,15,7], performance evaluation [10], and extension to support liveness properties [12]. The common theme of these works is that they are all based on the L^* learning algorithm and hence always generate deterministic finite automata as contextual assumptions. To the best of our knowledge, the only exception is [3], which is essentially a modified version of the counterexample guided abstraction refinement technique [9]. Our solution is orthogonal to abstraction refinement; it can apply abstraction refinement techniques implemented in Model Checkers.

The paper is organized as follows. Section 2 gives the background of our presentation. We review the exact learning algorithm CDNF for Boolean functions in Section 3. It is followed by our solution to the contextual assumption generation problem (Section 4). Section 5 gives our preliminary experimental results. Finally, we conclude in Section 6.

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 more restrictive 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:*

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

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 CDNF Algorithm

For a fixed set of Boolean variables $\mathbf{x}$ and a Boolean function $\lambda(\mathbf{x})$ over $\mathbf{x}$, an exact learning algorithm for Boolean functions computes a representation of $\lambda(\mathbf{x})$

in a finite number of steps. The CDNF algorithm is an exact learning algorithm for Boolean functions [4]. Like the L^* algorithm [1], the CDNF algorithm uses an *active learning* model. In the model, it is assumed that *a teacher*, who knows the *target* Boolean formula $\lambda(\mathbf{x})$, provides the learning algorithm with answers to the following types of queries:

- *Membership query* $MEM(\nu)$ *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* $EQ(\theta)$ *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 $EQ(\theta)$.

Consider the following examples. Assume $\lambda(x, y) = (x \wedge \neg y) \vee (\neg x \wedge y)$ is the target Boolean function over x and y. The teacher answers *NO* to the query $MEM(\nu)$ where $\nu(x) = \nu(y) = \mathsf{F}$ (denoted by $\nu(xy) = \mathsf{FF}$), since $\lambda(\mathsf{F}, \mathsf{F}) = \mathsf{F}$. For a different valuation $\nu(xy) = \mathsf{TF}$, the teacher answers *YES*. As an example of equivalence queries, consider $EQ(x \vee y)$. The teacher provides the valuation $\nu(xy) = \mathsf{TT}$ 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*.

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 aforementioned active learning model, 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 $|\mathbf{x}|$ [4].

4 Learning a Contextual Assumption

Recall the following assume-guarantee reasoning rule (Lemma 1):

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

Our goal is to generate a contextual assumption $A = (\mathbf{x}_1, \iota_A(\mathbf{x}_1), \tau_A(\mathbf{x}_1, \mathbf{x}'_1))$ such that the premises $M_0 \| A \models \pi$ and $M_1 \preceq A$ hold. The contextual assumption consists of two parts: $\iota_A(\mathbf{x}_1)$ and $\tau_A(\mathbf{x}_1, \mathbf{x}'_1)$ which are Boolean functions over $\mathbf{x}_1$ and $\mathbf{x}_1 \cup \mathbf{x}'_1$ respectively. We naturally use the CDNF algorithm to learn both Boolean functions. Precisely, two instances of the CDNF algorithm are deployed: one for the initial predicate $\iota_A(\mathbf{x}_1)$, and the other for the transition relation $\tau_A(\mathbf{x}_1, \mathbf{x}'_1)$. Remember that the CDNF algorithm relies on a teacher, who knows the target Boolean function already, to answer queries from the learning algorithm. In this case, the target functions are unknown. We use the two premises of the assume-guarantee reasoning rule (Lemma 1) to simulate the role of a teacher. We explain in detail how this is done for the rest of Section 4.

There are four different types of queries (from the two instances of the CDNF algorithm) that need to be handled:

– 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 $\tau_A(\mathbf{x}_1, \mathbf{x}'_1)$.

In order to resolve membership queries, we exploit the fact that any contextual assumption must simulate M_1. The membership query $MEM(\mu)$ for the target $\iota_A(\mathbf{x}_1)$ is resolved by checking if μ satisfies $\iota_1(\mathbf{x}_1)$. If so, μ must also satisfy $\iota_A(\mathbf{x}_1)$ because M_1 is simulated by any contextual assumption A. The membership query $MEM(\mu, \mu')$ is resolved similarly.

For equivalence queries, we answer *YES* when a contextual assumption is found. Note that both conjectures $\iota(\mathbf{x}_1)$ and $\tau(\mathbf{x}_1, \mathbf{x}'_1)$ are needed to decide if they represent a contextual assumption. The two types of equivalence queries $EQ(\iota)$ and $EQ(\tau)$ hence cannot be resolved independently. In contrast to membership query resolution algorithms, there is only one equivalence query resolution algorithm for both types of equivalence queries.

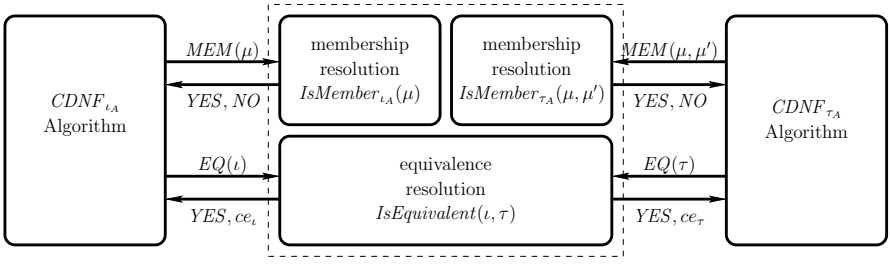

Fig. 1. Structure of Contextual Assumption Generator

Figure 1 shows the interaction between components in our contextual assumption generation algorithm. In the figure, two instances of the CDNF algorithm are shown on the sides. The instance $CDNF_{\iota_A}$ is intended to compute the initial predicate $\iota_A(\mathbf{x}_1)$ of an unknown contextual assumption A; the instance $CDNF_{\tau_A}$ is to compute the transition relation $\tau_A(\mathbf{x}_1, \mathbf{x}'_1)$ of A. The dashed box in the middle denotes the teachers. We design three query resolution algorithms to simulate the teachers for the two instances of the CDNF algorithm.

The membership query resolution algorithm $IsMember_{\iota_A}(\mu)$ resolves the membership query $MEM(\mu)$ for the target $\iota_A(\mathbf{x}_1)$. It receives queries and sends answers to the instance $CDNF_{\iota_A}$. Similarly, the membership query resolution algorithm $IsMember_{\tau_A}(\mu, \mu')$ communicates with the instance $CDNF_{\tau_A}$ solely. The equivalence query resolution algorithm $IsEquivalent(\iota, \tau)$, however, needs both conjectures from $CDNF_{\iota_A}$ and $CDNF_{\tau_A}$. It hence interacts with both instances.

4.1 Resolving Membership Queries

Let μ be a valuation over $\mathbf{x}_1$. The membership query $MEM(\mu)$ asks if μ is a satisfying valuation for the initial predicate $\iota_A(\mathbf{x}_1)$ of an unknown contextual

assumption A. We exploit the simulation relation in the assume-guarantee reasoning rule to resolve membership queries.

Input: $MEM(\mu)$: a membership query for the target $\iota_A(\mathbf{x}_1)$
Output: *YES* or *NO*
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*
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

Algorithm 1a shows the membership query resolution algorithm for $MEM(\mu)$. In order to understand the algorithm, recall the premise $M_1 \preceq A$ in the assume-guarantee reasoning rule (Lemma 1). The initial predicate $\iota_A(\mathbf{x}_1)$ for any contextual assumption A must satisfy $\forall \mathbf{x}_1.\iota_1(\mathbf{x}_1) \implies \iota_A(\mathbf{x}_1)$. On the given valuation μ over $\mathbf{x}_1$, we hence check if $\iota_1[\mu] = \mathsf{T}$. If so, we have $\iota_A[\mu] = \mathsf{T}$ by $M_1 \preceq A$ and return *YES*. Otherwise, we simply return *NO* for the sake of termination. Observe that the answers to membership queries for the target $\iota_A(\mathbf{x}_1)$ are consistent with $\iota_1(\mathbf{x}_1)$. Algorithm 1a effectively targets the initial predicate $\iota_1(\mathbf{x}_1)$ of M_1. Subsequently, $CDNF_{\iota_A}$ can infer $\iota_1(\mathbf{x}_1)$ of M_1 as the initial predicate $\iota_A(\mathbf{x}_1)$ of an unknown contextual assumption eventually. Of course, one expects that an initial predicate different from $\iota_1(\mathbf{x}_1)$ will be learned. Our experiments show that this is indeed the case in practice.

Resolving membership queries $MEM(\mu, \mu')$ for the transition relation $\tau_A(\mathbf{x}_1, \mathbf{x}'_1)$ of an unknown contextual assumption is almost identical (Algorithm 1b). Let μ and μ' be valuations over $\mathbf{x}_1$ and $\mathbf{x}'_1$ respectively. Similar to the case of initial predicate, the transition relation $\tau_A(\mathbf{x}_1, \mathbf{x}'_1)$ of any contextual assumption A must satisfy $\forall \mathbf{x}_1, \mathbf{x}'_1.\tau_1(\mathbf{x}_1, \mathbf{x}'_1) \implies \tau_A(\mathbf{x}_1, \mathbf{x}'_1)$ due to $M_1 \preceq A$. If $\tau_1[\mu, \mu'] = \mathsf{T}$, $\tau_A[\mu, \mu'] = \mathsf{T}$ and hence our membership resolution algorithm returns *YES*. Otherwise, Algorithm 1b returns *NO*. As in the membership query resolution algorithm for the initial predicate, these answers make sure that $CDNF_{\tau_A}$ can infer the transition relation $\tau_1(\mathbf{x}_1, \mathbf{x}'_1)$ of M_1 and terminate eventually.

4.2 Resolving Equivalence Queries

Our equivalence query resolution algorithm answers two different types of equivalence queries from the two instances of the CDNF algorithm. The equivalence query $EQ(\iota)$ from $CDNF_{\iota_A}$ asks if the Boolean function $\iota(\mathbf{x}_1)$ represents the initial predicate of an unknown contextual assumption; $EQ(\tau)$ from $CDNF_{\tau_A}$ asks if the Boolean function $\tau(\mathbf{x}_1, \mathbf{x}'_1)$ represents the transition relation of an unknown contextual assumption.

Let $\iota(\mathbf{x}_1)$ and $\tau(\mathbf{x}_1, \mathbf{x}_1')$ be conjectures. Consider the transition system $C = (\mathbf{x}_1, \iota(\mathbf{x}_1), \tau(\mathbf{x}_1, \mathbf{x}_1'))$. Our equivalence query resolution algorithm first checks if M_1 is simulated by C. If M_1 is not simulated by C, the equivalence query resolution algorithm returns a counterexample to either $CDNF_{\iota_A}$ or $CDNF_{\tau_A}$. Otherwise, it continues to check if C is in fact a contextual assumption by verifying $M_0 \| C \models \pi$ with a Model Checking algorithm. If the composition of M_0 and C satisfies π, the equivalence query resolution algorithm returns *YES*. We conclude that $M_0 \| M_1$ satisfies π. If the composition of M_0 and C does not satisfy π, the equivalence query resolution algorithm examines the witness returned by the Model Checking algorithm. If the witness is also a witness to $M_0 \| M_1 \not\models \pi$, we conclude that $M_0 \| M_1$ does not satisfy π. Otherwise, the equivalence query resolution algorithm returns a counterexample to either $CDNF_{\iota_A}$ or $CDNF_{\tau_A}$.

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)

Algorithm 2 gives details of our equivalence query resolution algorithm. Let C be the transition system $(\mathbf{x}_1, \iota(\mathbf{x}_1), \tau(\mathbf{x}_1, \mathbf{x}_1'))$. To verify that M_1 is simulated by C, the algorithm checks if $\iota_1(\mathbf{x}_1) \wedge \neg\iota(\mathbf{x}_1)$ is satisfiable. If $\iota_1(\mathbf{x}_1) \wedge \neg\iota(\mathbf{x}_1)$ is satisfied by a valuation μ, then $\forall \mathbf{x}_1 . \iota_1(\mathbf{x}_1) \implies \iota(\mathbf{x}_1)$ does not hold and hence $M_1 \not\preceq C$. The valuation μ is returned to $CDNF_{\iota_A}$ as a counterexample to the equivalence query $EQ(\iota)$. The equivalence query resolution algorithm then restarts after it receives another conjecture from $CDNF_{\iota_A}$. Similarly, if $\tau_1(\mathbf{x}_1, \mathbf{x}_1') \wedge \neg\tau(\mathbf{x}_1, \mathbf{x}_1')$ is satisfied by $\mu\mu'$, the valuation $\mu\mu'$ is returned to $CDNF_{\tau_A}$ as a counterexample to the equivalence query $EQ(\tau)$.

Now assume $M_1 \preceq C$. That is, the second premise of the assume-guarantee reasoning rule is fulfilled. It remains to verify $M_0 \| C \models \pi$. The equivalence query resolution algorithm uses Model Checking to verify if $M_0 \| C \models \pi$. If $M_0 \| C \models \pi$,

both premises of the assume-guarantee reasoning rule are fulfilled. The equivalence resolution algorithm concludes $M_0 \| M_1 \models \pi$. Otherwise, the Model Checking algorithm returns a witness α to $M_0 \| C \not\models \pi$. Recall that M_1 is simulated by C and hence $\mathit{Trace}(M_1) \subseteq \mathit{Trace}(C)$. A witness α to $M_0 \| C \not\models \pi$ is not necessary a witness to $M_0 \| M_1 \not\models \pi$ for $\alpha{\downarrow_{\mathbf{x}_1}}$ may not be a trace of M_1. We therefore check whether $\alpha{\downarrow_{\mathbf{x}_1}} \in \mathit{Trace}(M_1)$ by the witness analysis algorithm.

Analyzing Witnesses. Given a witness α to $M_0 \| C \not\models \pi$, the witness analysis algorithm $\mathit{IsWitness}(\alpha)$ inspects α to see if $\alpha{\downarrow_{\mathbf{x}_1}}$ is also a trace of M_1. If so, α is a witness to $M_0 \| M_1 \not\models \pi$. Otherwise, the transition system C deviates from M_1 at some point in $\alpha{\downarrow_{\mathbf{x}_1}}$. The deviation is returned to either CDNF_{ι_A} or CDNF_{τ_A} as a counterexample to $EQ(\iota)$ or $EQ(\tau)$ respectively (Algorithm 3).

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 $\mathit{IsEquivalent}(\iota', \tau)$;
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 $\mathit{IsEquivalent}(\iota, \tau')$;
end
report "$M_0 \| M_1 \not\models \pi$ is witnessed by α";

Algorithm 3. $\mathit{IsWitness}(\alpha)$

More concretely, let $\alpha{\downarrow_{\mathbf{x}_1}} = \mu^0 \mu^1 \cdots \mu^t$ be a sequence of valuations over $\mathbf{x}_1$. Algorithm 3 verifies whether μ^0 is an initial state of M_1. If not, μ^0 is a counterexample to the equivalence query $EQ(\iota)$. Otherwise, the witness analysis algorithm checks if each transition of $\alpha{\downarrow_{\mathbf{x}_1}}$ on C is also a transition on M_1. If the i-th transition of $\alpha{\downarrow_{\mathbf{x}_1}}$ is not a transition on M_1 (that is, $\tau_1[\mu^{i-1}, \mu^i] = \mathsf{F}$), the valuation $\mu^{i-1}\mu^i$ is returned as a counterexample to the equivalence query $EQ(\tau)$. If a counterexample to either $EQ(\iota)$ or $EQ(\tau)$ is found, the equivalence query resolution algorithm waits for a new conjecture and then restarts. Otherwise, every transition of $\alpha{\downarrow_{\mathbf{x}_1}}$ is also a transition on M_1, α is a witness to $M_0 \| M_1 \not\models \pi$.

4.3 Correctness

The correctness of our assumption generation algorithm is established in three steps: proving soundness, completeness, and 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 $\mathbf{x} = \mathbf{x}_0 \cup \mathbf{x}_1$. When the equivalence query resolution algorithm (Algorithm 2)

reports "$M_0 \| M_1 \models \pi$," it has verified that the composition of M_0 and C satisfies π, where $C = (\mathbf{x}_1, \iota(\mathbf{x}_1), \tau(\mathbf{x}_1, \mathbf{x}'_1))$ is the transition system corresponding to the conjectures $\iota(\mathbf{x}_1)$ and $\tau(\mathbf{x}_1, \mathbf{x}'_1)$. Moreover, we have $M_0 \preceq C$ because both $\iota_1(\mathbf{x}_1) \wedge \neg\iota(\mathbf{x}_1)$ and $\tau_1(\mathbf{x}_1, \mathbf{x}'_1) \wedge \neg\tau(\mathbf{x}_1, \mathbf{x}'_1)$ are not satisfiable. By the soundness of the assume-guarantee reasoning rule (Lemma 1), we have $M_0 \| M_1 \models \pi$.

On the other hand, when the witness analysis algorithm (Algorithm 3) reports "$M_0 \| M_1 \not\models \pi$ is witnessed by α," it has checked that $\alpha{\downarrow_{\mathbf{x}_1}}$ is a trace of M_1. Moreover, α is a witness to $M_0 \| C \not\models \pi$ and hence $\alpha{\downarrow_{\mathbf{x}_0}}$ is a trace of M_0. Since $\alpha{\downarrow_{\mathbf{x}_i}}$ is a trace of M_i for $i = 0, 1$, α is a trace of $M_0 \| M_1$ and thus a witness to $M_0 \| M_1 \not\models \pi$ as well. Our contextual assumption generation algorithm is sound.

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 $M_0 \| M_1 \not\models \pi$.*

If $M_0 \| M_1 \models \pi$, there is a transition system $C = (\mathbf{x}_1, \iota(\mathbf{x}_1), \tau(\mathbf{x}_1, \mathbf{x}'_1))$ such that $M_0 \| C \models \pi$ and $M_1 \preceq C$ by the invertibility of the assume-guarantee reasoning rule. Thus $\iota_1(\mathbf{x}_1) \wedge \neg\iota(\mathbf{x}_1)$ and $\tau_1(\mathbf{x}_1, \mathbf{x}'_1) \wedge \neg\tau(\mathbf{x}_1, \mathbf{x}'_1)$ are not satisfiable. Hence Algorithm 2 reports "$M_0 \| M_1 \models \pi$." On the other hand, assume α is a witness to $M_0 \| M_1 \not\models \pi$. Consider the transition system $C = (\mathbf{x}_1, \iota_{\mathsf{T}}(\mathbf{x}_1), \tau_{\mathsf{T}}(\mathbf{x}_1, \mathbf{x}'_1))$ where $\iota_{\mathsf{T}}(\mathbf{x}_1) = \mathsf{T}$ and $\tau_{\mathsf{T}}(\mathbf{x}_1, \mathbf{x}'_1) = \mathsf{T}$. Clearly $M_1 \preceq C$ and hence α is a witness to $M_0 \| C \not\models \pi$. Algorithm 3 reports "$M_0 \| M_1 \not\models \pi$ is witnessed by α." Our contextual assumption generation algorithm is complete.

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.*

It remains to show that our algorithm always reports "$M_0 \| M_1 \models \pi$" or "$M_0 \| M_1 \not\models \pi$ is witnessed by α." Observe that the answers given by our query resolution algorithms are consistent with $\iota_1(\mathbf{x}_1)$ and $\tau_1(\mathbf{x}_1, \mathbf{x}'_1)$. Hence the instance $CDNF_{\iota_A}$ will infer $\iota_1(\mathbf{x}_1)$ after a polynomial number of queries. Similarly, $CDNF_{\tau_A}$ will generate $\tau_1(\mathbf{x}_1, \mathbf{x}'_1)$ eventually. At this point, the corresponding transition system $C = (\mathbf{x}_1, \iota_1(\mathbf{x}_1), \tau_1(\mathbf{x}_1, \mathbf{x}'_1)) = M_1$. The equivalence query resolution algorithm can always decide whether $M_0 \| M_1 \models \pi$ or not. Our contextual assumption generation algorithm therefore always reports to the user after a polynomial number of queries.

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* $\mathbf{x} = \mathbf{x}_0 \cup \mathbf{x}_1$*. The contextual assumption generation algorithm reports "*$M_0 \| M_1 \models \pi$*" or "*$M_0 \| M_1 \not\models \pi$ *is witnessed by* α*" within a polynomial number of queries in* $|\iota_1(\mathbf{x}_1)|_{DNF}$*,* $|\iota_1(\mathbf{x}_1)|_{CNF}$*,* $|\tau_1(\mathbf{x}_1, \mathbf{x}_1')|_{DNF}$*,* $|\tau_1(\mathbf{x}_1, \mathbf{x}_1')|_{CNF}$*, and* $|\mathbf{x}_1|$*.*

5 Experiments

We have implemented a prototype of our contextual assumption generation algorithm in OCaml. Our current implementation uses the OCaml thread library for synchronization purposes. Each instance of the *CDNF* algorithm (that is, $CDNF_{\iota_A}$ or $CDNF_{\tau_A}$) is executed in a separate thread, and the equivalence query resolution algorithm is executed in a third thread.

We use MiniSat 2 (version 070721) in the membership query resolution algorithms (Algorithm 1) and the simulation checking in the equivalence query resolution algorithm (Algorithm 2). For monolithic Model Checking, we implement the interpolation-based algorithm in [19]. 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 three test cases in this section: the MSI cache coherence protocol [16], synchronous bus arbiters [18], and dining philosophers [22]. Each test case has experiments parametrized by the number of nodes. Let $M_1, \ldots, M_n$ be the nodes in an experiment with n nodes, and π a state predicate. We verify $M_1 \| \cdots \| M_n \models \pi$ in an experiment with n nodes.

Assume-guarantee reasoning is compared with monolithic interpolation-based Model Checking in each experiment. We explored several different partitions in each experiment. More precisely, an experiment with n nodes is divided into different partitions in n trials. In the i-th trial, we apply the following assume-guarantee reasoning rule:

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

Our contextual assumption algorithm generates a contextual assumption A to verify $M_1 \| \cdots \| M_n \models \pi$ in each trial. Since we do not address the decomposition problem in this paper, we choose the best result among the n trials and compare it with monolithic Model Checking. All experimental results are collected on a 3.2GHz Intel Xeon server with 2GB memory running Linux 2.4.20.

MSI Cache Coherence Protocol In the MSI cache coherence protocol, a memory is shared among n nodes. 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 [16]. Two properties are verified on the model derived from NuSMV [8]. We check that the first two nodes cannot own the bus simultaneously. Then we verify that

nodes	4	5	6	7	8	9	10	11	12
monolithic (sec)	2.6	4.1	4.9	5.3	6.0	7.9	7.6	9.3	9.6
assume-guarantee (sec)	1.5	1.9	4.0	2.7	3.6	6.3	7.1	7.6	8.6
improvement (%)	42.3	53.6	18.3	49.0	40.0	20.2	6.5	18.2	10.4
nodes	13	14	15	16	17	18	19	20	avg
monolithic (sec)	7.7	6.6	6.8	14.7	8.4	8.7	18.2	18.5	**8.6**
assume-guarantee (sec)	9.0	7.7	6.5	11.3	8.3	8.4	8.9	9.6	**6.6**
improvement (%)	-16.8	-16.6	4.4	23.1	1.1	3.4	51.0	48.1	**20.9**

(a) no contention for the first two nodes

nodes	4	5	6	7	8	9	10	11	12
monolithic	5s	15s	30s	42s	48s	1m43s	2m18s	5m8s	5m30s
assume-guarantee	3s	4s	30s	31s	31s	1m5s	42s	1m55s	1m33s
improvement (%)	40.0	73.3	0.0	26.1	35.4	36.8	69.5	62.6	71.8
nodes	13	14	15	16	17	18	19	20	avg
monolithic	2m37s	2m39s	3m14s	1m24s	6m38s	9m26s	9m26s	9m1s	**3m36s**
assume-guarantee	2m1s	2m20s	2m16s	1m28s	3m14s	4m5s	5m12s	9m11s	**2m9s**
improvement (%)	22.9	11.9	29.8	-4.7	51.2	56.7	44.8	-1.8	**36.8**

(b) no contention for all nodes

Fig. 2. Experimental Results for the MSI Protocol

any pair of nodes cannot own the bus at the same time. The former property involves only two nodes and is easier to verify than the latter. Figure 2 shows the results of experiments with 4 to 20 nodes.

In the figure, we show the verification time of the monolithic interpolation-based Model Checking (*monolithic*), the verification time of assume-guarantee reasoning (*assume-guarantee*), and the ratio of improvement (*improvement*). On the first property, monolithic Model Checking takes more than 14 seconds in the experiments with 16, 19, and 20 nodes. Assume-guarantee reasoning, on the other hand, finishes all but one experiments in 10 seconds. Assume-guarantee reasoning also performs significantly better than monolithic Model Checking on the second property. The verification time for assume-guarantee reasoning increases more stably than monolithic Model Checking (Figure 2b). The generated contextual assumptions improve assume-guarantee reasoning by 50% in 5 experiments with no less than 10 nodes. Given an experiment in this test case, one expects assume-guarantee reasoning to outperform monolithic Model Checking by 20.9% and 36.8% on the two properties respectively.

Synchronous Bus Arbiters. The synchronous bus arbiter is a bus arbitration protocol for synchronous circuits [18]. 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. We generalize the model in NuSMV [8] and verify two properties in this test case. We check that the first pair of nodes cannot

nodes	4	5	6	7	8	9	10	11	12
monolithic (sec)	5.1	7.6	11.1	16.6	25.5	42.4	58.9	81.1	123.7
assume-guarantee (sec)	4.2	6.4	10.5	14.5	22.9	36.4	41.3	45.8	108.2
improvement (%)	17.6	15.7	5.4	12.6	10.1	14.1	29.8	43.5	12.6
nodes	13	14	15	16	17	18	19	20	avg
monolithic (sec)	159.3	130.6	314.0	81.3	423.1	548.8	698.3	900.0	**213.3**
assume-guarantee (sec)	139.6	115.0	188.9	61.1	374.4	463.3	531.9	568.2	**160.7**
improvement (%)	12.3	11.9	39.8	24.8	11.5	15.5	23.8	36.8	**19.8**

(a) no simultaneous acknowledgment for the first two nodes

nodes	4	5	6	7	8	9	10	11	12
monolithic	3s	5s	5s	10s	34s	34s	1m45s	1m51s	4m32s
assume-guarantee	3s	5s	5s	10s	34s	50s	1m44s	1m59s	4m33s
improvement (%)	0	0	0	0	0	-47.0	0.9	-7.2	-0.3
nodes	13	14	15	16	17	18	19	20	avg
monolithic	7m9s	10m54s	12m27s	21m2s	30m22s	24m3s	33m38s	45m29s	**11m35s**
assume-guarantee	7m4s	8m43s	8m43s	12m39s	17m57s	24m0s	33m22s	45m20s	**9m43s**
improvement (%)	1.1	20.0	29.9	39.8	40.8	0.2	0.7	0.3	**4.6**

(b) no simultaneous acknowledgment for any pair of nodes

Fig. 3. Experimental Results for Synchronous Bus Arbiters

acknowledge the token simultaneously. Then we check that any pair of nodes cannot acknowledge the token at the same time. Figure 3 shows the results.

For the first property, assume-guarantee reasoning outperforms monolithic Model Checking consistently. Our algorithm computes a contextual assumption that improves the verification time by 19.8% on average. Assume-guarantee reasoning decisively outperforms monolithic Model Checking for experiments with 14 to 17 nodes on the second property. Among the experiments in all three cases, the experiments with 9 nodes is the only one where assume-guarantee reasoning is outperformed by more than 20%. Subsequently, assume-guarantee reasoning does not significantly improve the verification time on this property (4.6%).

Dining Philosophers. The dining philosophers problem illustrates a simple resource sharing problem in concurrent programs. In dining philosophers, n nodes are connected in a ring. Neighboring nodes share a resource. A node requires both resources shared with its neighbors to enter its working mode [22]. In this test case, we verify that a fixed pair of neighboring nodes cannot enter their working modes simultaneously (Figure 4).[1]

Our experiments show that the verification time of monolithic Model Checking varies drastically in this case. Assume-guarantee reasoning, on the other hand, performs more stably. Take the experiment with node 17 as an example.

[1] In fact, verifying that any neighboring nodes cannot enter the working mode in dining philosophers takes so much time that both monolithic Model Checking and assume-guarantee reasoning cannot finish in one hour in a setting with only 4 philosophers.

nodes	4	5	6	7	8	9	10	11	12
monolithic (sec)	15.8	16.6	823.7	141.1	22.7	56.1	32.0	34.7	64.3
assume-guarantee (sec)	13.1	11.3	33.3	15.1	10.9	19.6	32.2	23.6	32.1
improvement (%)	17.0	21.0	95.9	89.2	51.9	65.0	-0.6	31.9	50.0
nodes	13	14	15	16	17	18	19	20	avg
monolithic (sec)	1109.9	60.6	46.1	32.7	1741.1	91.1	2406.7	63.7	**397.5**
assume-guarantee (sec)	29.5	34.3	36.8	28.9	58.8	66.4	39.5	67.5	**32.5**
improvement (%)	97.3	43.3	20.1	11.6	96.6	27.1	98.3	-5.9	**47.6**

Fig. 4. Experimental Results for Dining Philosophers

Interpolation-based algorithm uses 180MB memory to compute 8 interpolants to conclude that the property is verified. Assume-guarantee reasoning only uses 104MB memory and 7 interpolants to reach the same conclusion. With our contextual assumption generation algorithm, assume-guarantee reasoning is expected to outperform monolithic Model Checking by 47.6% in this test case.

6 Conclusion

We introduced a new contextual assumption generation algorithm in this paper. The new algorithm computes implicit representations and is more scalable than explicit automata-theoretic algorithms. With the contextual assumptions generated by our algorithm, assume-guarantee reasoning can improve monolithic interpolation-based Model Checking in three parametrized test cases.

The initial predicate and the transition relation of the generated contextual assumption are different from those of a node. In all 1020 $(= (2+2+1) \times \sum_{n=4}^{20} n)$ trials, each generated contextual assumptions has different initial predicates and transition relations from those of its target node. Moreover, since the generated contextual assumption simulates its target, it is in fact an abstraction of the target node [9,3]. Although our contextual assumption generation algorithm can apply abstraction refinement techniques implemented in Model Checkers, it will be interesting to compare these two techniques.

Targeting one node is not the best decomposition we have in our test cases. In the MSI cache coherence protocol, targeting all nodes allows assume-guarantee reasoning to verify the experiment with 36 nodes in 4 minutes whereas monolithic Model Checking uses up all memory in 9 minutes and fails to verify. The challenge of how to best decompose a problem still remains. In summary, our experiments show that there is always *a decomposition* to make assume-guarantee reasoning outperform monolithic interpolation-based Model Checking in the three test cases. Finding such a decomposition will certainly be an important future work.

References

1. Angluin, D.: Learning regular sets from queries and counterexamples. Information and Computation 75(2), 87–106 (1987)
2. Barringer, H., Giannakopoulou, D., Păsăreanu, C.S.: Proof rules for automated compositional verification through learning. In: Workshop on Specification and Verification of Component-Based Systems, pp. 14–21 (2003)
3. Bobaru, M.G., Păsăreanu, C.S., Giannakopoulou, D.: Automated assume-guarantee reasoning by abstraction refinement. In: Gupta, A., Malik, S. (eds.) CAV 2008. LNCS, vol. 5123, pp. 135–148. Springer, Heidelberg (2008)
4. Bshouty, N.H.: Exact learning boolean function via the monotone theory. Information and Computation 123(1), 146–153 (1995)
5. Chaki, S., Clarke, E.M., Sinha, N., Thati, P.: Automated assume-guarantee reasoning for simulation conformance. In: Etessami, K., Rajamani, S.K. (eds.) CAV 2005. LNCS, vol. 3576, pp. 534–547. Springer, Heidelberg (2005)
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., 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 2009. LNCS, vol. 5505, pp. 31–45. Springer, Heidelberg (2009)
8. Cimatti, A., Clarke, E., Giunchiglia, F., Roveri, M.: NuSMV: a new Symbolic Model Verifier. In: Halbwachs, N., Peled, D. (eds.) CAV 1999. LNCS, vol. 1633, pp. 495–499. Springer, Heidelberg (1999)
9. Clarke, E.M., Grumberg, O., Jha, S., Lu, Y., Veith, H.: Counterexample-guided abstraction refinement for symbolic model checking. J. ACM 50(5), 752–794 (2003)
10. Cobleigh, J.M., Avrunin, G.S., Clarke, L.A.: Breaking up is hard to do: An evaluation of automated assume-guarantee reasoning. ACM Trans. Software Engineering Methodology 17(2) (2008)
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. Farzan, A., Chen, Y.F., Clarke, E.M., Tsay, Y.K., Wang, B.Y.: Extending automated compositional verification to the full class of omega-regular languages. In: Ramakrishnan, C., Rehof, J. (eds.) TACAS 2008. LNCS, vol. 4963, pp. 2–17. Springer, Heidelberg (2008)
13. Gheorghiu, M., Giannakopoulou, D., Păsăreanu, C.S.: Refining interface alphabets for compositional verification. In: Grumberg, O., Huth, M. (eds.) TACAS 2007. LNCS, vol. 4424, pp. 292–307. Springer, Heidelberg (2007)
14. Giannakopoulou, D., Păsăreanu, C.S.: Special issue on learning techniques for compositional reasoning. Formal Methods in System Design 32(3), 173–174 (2008)
15. Gupta, A., McMillan, K.L., Fu, Z.: Automated assumption generation for compositional verification. Formal Methods in System Design 32(3), 285–301 (2008)
16. Handy, J.: The Cache Memory Book. Academic Press, London (1998)
17. Jung, Y., Kong, S., Wang, B.Y., Yi, K.: Deriving invariants in propositional logic by algorithmic learning, decision procedure, and predicate abstraction. In: VMCAI. LNCS. Springer, Heidelberg (2010)
18. McMillan, K.L.: The SMV system, symbolic model checking - an approach. Technical Report CMU-CS-92-131, Carnegie Mellon University (1992)

19. McMillan, K.L.: Interpolation and SAT-based model checking. In: Hunt Jr., W.A.H., Somenzi, F. (eds.) CAV 2003. LNCS, vol. 2725, pp. 1–13. Springer, Heidelberg (2003)
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. Rivest, R.L., Schapire, R.E.: Inference of finite automata using homing sequences. Information and Computation 103(2), 299–347 (1993)
22. Silberschatz, A., Galvin, P.B., Gagne, G.: Operating System Concepts, 7th edn. John Wiley & Sons, Inc., Chichester (2004)
23. 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)