Learning-based Assume-Guarantee Regression Verification*

Fei He[1,2,3], Shu Mao[1,2,3], Bow-Yaw Wang[4]

[1] Tsinghua National Laboratory for Information Science and Technology (TNList)
[2] School of Software, Tsinghua University
[3] Key Laboratory for Information System Security, Ministry of Education, China
[4] Academia Sinica, Taiwan

Abstract. Due to enormous resource consumption, model checking each revision of evolving systems repeatedly is impractical. To reduce cost in checking every revision, contextual assumptions are reused from assume-guarantee reasoning. However, contextual assumptions are not always reusable. We propose a fine-grained learning technique to maximize the reuse of contextual assumptions. Based on fine-grained learning, we develop a regressional assume-guarantee verification approach for evolving systems. We have implemented a prototype of our approach and conducted extensive experiments (with 1018 verification tasks). The results suggest promising outlooks for our incremental technique.

1 Introduction

Software systems evolve throughout their life cycles. In order to add new features, many revisions are released over time. Since errors may be introduced with new releases, each revision needs to be formally verified. Formal verification however is still very time-consuming. Verifying every revision of an evolving system is impractical. A more effective technique to ensure correctness of evolving software systems is desired.

Model checking is a formal verification technique [4, 17]. In model checking, lots of internal information is computed during a verification run. Note that two consecutive revisions share many behaviors. When a revision is verified, internal information from model checking may still be useful to verifying the next revision. Regression verification expands this idea by reusing internal information to speed up the verification of later revisions [3, 6, 7, 10, 22, 27–29]. Various internal information has been proposed for reuse, including state space graphs [22, 29], constraint solving results [28], function summaries [3, 25], and abstract precisions [6].

Assume-guarantee reasoning [18] is a compositional technique to improve the scalability of model checking. In the compositional technique, contextual assumptions decompose verification tasks by summarizing component behaviors.

* This work was supported in part by the Chinese National 973 Plan (2010CB328003) and the NSF of China (61272001, 91218302).

Depending on compositional proof rules, contextual assumptions are required to fulfill different criteria for sound verification. Although they used to be constructed manually, contextual assumptions can be generated automatically by machine learning algorithms [13, 14, 18, 21].

Like internal information from model checking, contextual assumptions for the current revision may be reused for the next revision as well. Since contextual assumptions contains the most important information for verifying the current revision, they may immediately conclude the verification of the next revision. Contextual assumptions may be more suitable for regression verification. Compared to internal information from model checking, contextual assumptions are external information. They can be stored and reused without modifying model checking algorithms. In [26], contextual assumptions are exploited in regression verification. When the component summarized by contextual assumptions is not changed, the contextual assumptions are reused and modified to verify revised composed systems. If a system evolves into a new version, components may all be revised. Contextual assumptions thus can not be reused in regression verification. This can be a severe limitation.

Recall that system models are often represented by logic formulas in symbolic verification algorithms. A component may be represented by several logic formulas. Moreover, such logic formulas are further decomposed into more subformulas to attain the best performance. When a system with few components is updated, it is unlikely that all subformulas are revised. The chance of information reuse can be greatly improved if systems are decomposed into finer constituents. In our fine-grained learning framework, an instance of the learning algorithm [8, 19, 23] is deployed for each logic subformulas. When all instances infer their conjectures, a contextual assumption can be built from these conjectures and sent for assume-guarantee reasoning. We call this the *fine-grained learning-based verification*.

Using our fine-grained technique, we improve regression verification by *incremental assume-guarantee reasoning*. The word *incremental* means the previously-computed results are reused in later verification runs. Given a new revision of the system model represented as a number of logical formulas. We compare the previous revision and the new revision for each subformula. If they remain the same, the inferred conjecture in the previous verification for this subformula can be safely reused. Otherwise the conjecture is re-constructed. Since two revisions have similar behaviors, many of their subformulas remain unchanged. Previously inferred conjectures is likely to be reused.

We have implemented a prototype on top of NuSMV. We performed extensive experiments (with 1018 verification tasks) to evaluate the efficiency of our technique. Experimental results are very promising. If properties are satisfied before and after revisions, our new technique is about four times faster than conventional assume-guarantee reasoning. A similar speedup is also observed for unsatisfied properties before and after revisions. If properties are satisfied before but unsatisfied after revisions, incremental assume-guarantee reasoning also out-

performs but less significantly. Overall, we report more than three times speedup on more than a thousand verification tasks.

The remainder of this paper is organized as follows. Section 2 introduces necessary background. Section 3 explains our motivation. Fine-grained learning is discussed in Section 4. Our regression verification framework is presented in Section 5. Experimental results are reported in Section 6. Related work are discussed in Section 7. Finally Section 8 concludes this paper.

2 Background

Let $\mathbb{B}$ be the Boolean domain and X a finite set of Boolean *variables*. A *valuation* $s : X \to \mathbb{B}$ of X is a mapping from X to $\mathbb{B}$. A *predicate* $\phi(X)$ over X maps a valuation of X to $\mathbb{B}$. We may write ϕ if its variables are clear from the context.

Definition 1. *A* transition system $M = (X, \Lambda, \Gamma)$ *consists of a finite set of* variables X*, an* initial condition Λ *over* X*, and a* transition relation Γ *which is a predicate over* X *and* $X' = \{x' : x \in X\}$.

Definition 2. *Let* $M_i = \langle X_i, \Lambda_i, \Gamma_i \rangle$ *be transition systems for* $i = 0, 1$ *(X_i's are not necessarily disjoint), the composition* $M_0 \| M_1 = \langle X, \Lambda, \Gamma \rangle$ *is a transition system where* $X = X_0 \cup X_1$*,* $\Lambda(X) = \Lambda_0(X_0) \wedge \Lambda_1(X_1)$*, and* $\Gamma(X) = \Gamma_0(X_0) \wedge \Gamma_1(X_1)$.

Let $M = (X, \Lambda, \Gamma)$ be a transition system. A *state* s of M is a valuation over X. A *trace* σ of M is a sequence of states $s_0, s_1, \cdots, s_n$, such that s_0 is an initial state, and there is a transition from s_i to s_{i+1} for $i = 0, \ldots, n-1$. For any predicate ϕ, a sequence σ of states $s_0, s_1, \ldots, s_n$ *satisfies* ϕ (written $\sigma \models \phi$) if $s_i \models \phi$ for $i = 0, \ldots, n$. We say M *satisfies* ϕ (written $M \models \phi$) if $\sigma \models \phi$ for all traces of M. Given a transition system M and a predicate ϕ, the *invariant checking* problem is to decide whether M satisfies ϕ.

2.1 Learning-based Assume-Guarantee Verification

Assume-guarantee reasoning aims to mitigate the state explosion problem by divide-and-conquer strategy. It uses assumptions to summarize components. Since details of components can be ignored in assumptions, the compositional technique can be more effective than monolithic verification.

Definition 3. *Let* $M_i = \langle X, \Lambda_i, \Gamma_i \rangle$ *be transition systems for* $i = 0, 1$*,* M_1 simulates M_0 *(written* $M_0 \preceq M_1$*) if* $\Lambda_0 \Rightarrow \Lambda_1$ *and* $\Gamma_0 \Rightarrow \Gamma_1$.

Note that the above simulation relation is defined over first-order representation of models. Informally, $M_0 \preceq M_1$ if M_1 simulates all behaviors of M_0.

Theorem 1 ([14]). *Let* $M_i = \langle X_i, \Lambda_i, \Gamma_i \rangle$ *be transition systems for* $i = 0, 1$*,* $X = X_0 \cup X_1$*, and* $\phi(X)$ *a predicate, the following assume-guarantee reasoning rule is sound and invertible:*

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

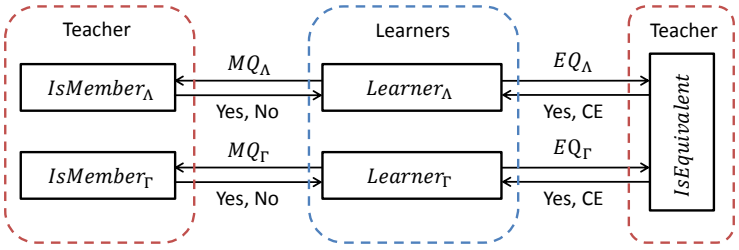


Fig. 1. The learning-based verification framework

A 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. In the proof rule (1), the transition system A is called a *contextual assumption* (for short, *assumption*) of M_0. A contextual assumption is *valid* if it either satisfies both premises of above rule, or is able to reveal a counterexample to $M_0 \| M_1 \models \phi$.

Active learning algorithms have been deployed to automatically learn the assumptions for compositional verification [1, 13, 14, 18, 20, 21]. Let U be an *unknown* predicate. A learning algorithm infers a Boolean formula characterizing U by making queries. It assumes a teacher who knows the target predicate U and answers the following two types of queries:

- On a *membership query* $MQ(s)$ with a valuation s, the teacher answers *YES* if $U(s)$ holds, and *NO* otherwise.
- On a *equivalence query* $EQ(H)$ with a *hypothesis* Boolean formula H, the teacher answers *YES* if H is semantically equal to U. Otherwise, she returns a valuation t on which H and U evaluate to different Boolean values as a counterexample.

Figure 1 shows the learning-based verification framework [13, 14, 21]. In the framework, a mechanical teacher is designed to answer queries from the learner. For simplicity of illustration, the mechanical teacher in the figure is divided into two parts, each answering one type of queries. Let $M_0 = \langle X_0, \Lambda_0, \Gamma_0 \rangle$ be a transition system. The mechanical teacher knows Λ_0 and Γ_0, and guides *Learner* to infer an assumption $A = \langle X_0, \Lambda_A, \Gamma_A \rangle$ fulfilling the premises of the proof rule (1). Two learning algorithms are instantiated: one for the initial condition Λ_A, the other for the transition relation Γ_A. For instance, consider the learning algorithm for Γ_A. For a membership query $MQ_\Gamma(s, t)$ from $Learner_\Gamma$, the mechanical teacher checks if $\langle s, t \rangle$ satisfies Γ_0. If so, the mechanical teacher answers *YES*. Otherwise, she answers *NO*. Conceptually, the mechanical teacher uses Γ_0 as the target predicate. In the worst case, the mechanical teacher infers Γ_0 as Γ_A.

The equivalence queries of the two learning algorithms need be synchronized. Let $\overline{\Lambda}_A$ and $\overline{\Gamma}_A$ be the current purported representations of Λ_A and Γ_A, respectively, the mechanical teacher first constructs $\overline{A} = \langle X_0, \overline{\Lambda}_A, \overline{\Gamma}_A \rangle$, then it checks

if the purported conjecture of $\overline{A}$ satisfies both premises of the assume-guarantee reasoning rule. If it does, the verification terminates and returns "safe". Otherwise, the premises checker returns a counterexample. The teacher then proceeds to check whether this counterexample is real or not. If it is a real counterexample, the verification algorithm terminates and reports "unsafe". Otherwise, the teacher returns this counterexample to *Learner*. *Learner* will use this counterexample to refine its purported formulas. This process repeats until a valid assumption is inferred.

2.2 Regression Verification

Computer systems evolve during their life time. Since the current version of a system has different behaviors from its previous versions, properties must be re-verified against the current version. In regression verification, we consider the invariant checking problem on two versions of a system. We would like to exploit any information from the previous verification in the current verification.

Definition 4. *Let $M = (X, \Lambda, \Gamma)$ and $M' = (X, \Lambda', \Gamma')$ be transition systems and $\phi(X)$ a specification. The* regression verification *problem is to check whether $M' \models \phi$ after the verification of $M \models \phi$.*

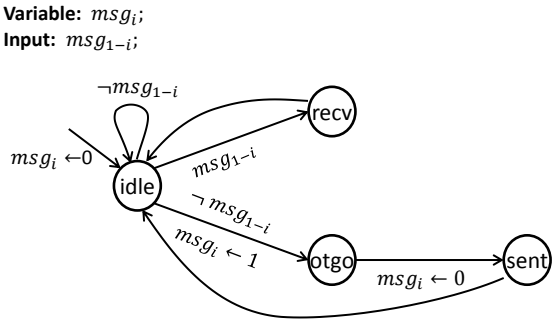
Note that Definition 4 does not assume whether the previous version M satisfies the property ϕ or not. We would like to re-use any information from the previous verification regardless of whether $M \models \phi$ holds or not.

3 Motivation

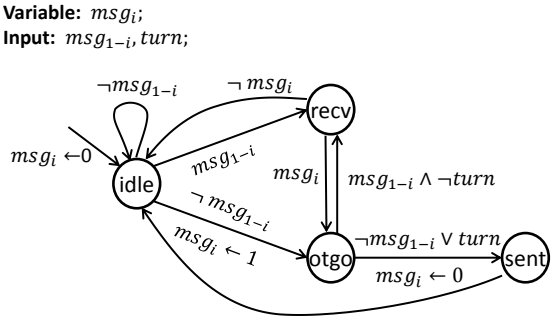
Let M_0 and M_1 be two components of a system, and A^* a valid contextual assumption. To perform regression verification on updated components M_0' and M_1', a natural idea is to reuse the contextual assumption A^*. However, it is shown in [26] that A^* as a whole can only be reused if $M_0' = M_0$ and M_1' simulates M_1. This can be a severe limitation.

3.1 An Example

Consider an email system composed of two clients c_i $(i = 0, 1)$. The client c_i is shown in Figure 2(a). Each c_i is associated with a data variable msg_i, whose value being `true` indicates that c_i is sending a message. When c_i sends a message, c_{1-i} will be informed and vice versa. The client c_i has four states: the idle state ("`idle`"), the receiving state ("`recv`"), the outgoing state ("`otgo`"), and the sent state ("`sent`"). Initially, c_i is at the `idle` state. If a message arrives (that is, msg_{1-i} = `true`), c_i transits to the receiving state `recv`. Otherwise, it nondeterministically transits to the outgoing state `otgo` and sets msg_i to `true`, or remains at the idle state `idle`. After the message is sent, the client transits to its `sent` state. Denote M_{c_i} the model of c_i.



(a) The email client $c_i (i = 0, 1)$



(b) The updated email client $c_i' (i = 0, 1)$

Fig. 2. The email client

The requirement ϕ_{es} is that all sent emails are well received. Formally, $\phi_{es} := (state_0 = \texttt{sent}) \leftrightarrow (state_1 = \texttt{recv})$. Apparently, ϕ_{es} is not satisfied by the model. Assume that both clients transit from their `idle` states to their `otgo` states simultaneously, representing both are going to send a message. The only next state for both of them is the `sent` state, which means that both clients have sent their messages, but none of them was well received.

The original model needs be revised to satisfy the requirement. Let $c_i' (i = 0, 1)$ be the updated client, shown in Fig. 2(b). In the new model, sending out a message is granted for a client if another client does not require sending at the same time. If both clients simultaneously want to send their messages, a new variable, called "*turn*", is introduced to assign priority to one of them.

Let us consider the regression verification of $M_{c_0'} \| M_{c_1'}$. Apparently, $M_{c_0'} \neq M_{c_0}$ and $M_{c_1'} \neq M_{c_1}$. According to [26], the contextual assumptions inferred in the verification of $M_{c_0} \| M_{c_1}$ cannot be reused in the verification of $M_{c_0'} \| M_{c_1'}$. However, if we take a look at the symbolic representations of these two revisions of the system, many commonalities can be identified.

Denote $M_{c_i} = \langle X_{c_i}, \Lambda_{c_i}, \Gamma_{c_i} \rangle$ for $i = 0, 1$, where X_{c_i} is a set including a state variable $state_i \in \{\texttt{idle}, \texttt{recv}, \texttt{otgo}, \texttt{sent}\}$ and a data variable msg_i. The model

M_{c_i} can be specified in a way that specifies for each variable x its initial values $init(x)$ and its next-state values $next(x)$. (for example, in NuSMV language [16]):

$$
\begin{array}{l}
init(state_i) := \texttt{idle}, init(msg_i) := \mathsf{false}, \\
next(state_i) := \\
\quad case \\
\quad\quad (state_i = \texttt{idle}) \wedge msg_{1-i} : \texttt{recv}; \\
\quad\quad (state_i = \texttt{idle}) \wedge \neg msg_{1-i} : \{\texttt{idle, otgo}\}; \\
\quad\quad (state_i = \texttt{otgo}) : \texttt{sent}; \\
\quad\quad (state_i = \texttt{recv}) : \texttt{idle}; \\
\quad\quad (state_i = \texttt{sent}) : \texttt{idle}; \\
\quad esac \\
next(msg_i) := \\
\quad case \\
\quad\quad (state_i = \texttt{idle}) \wedge (next(state_i) = \texttt{otgo}) : \mathsf{true}; \\
\quad\quad (state_i = \texttt{otgo}) \wedge (next(state_i) = \texttt{sent}) : \mathsf{false}; \\
\quad\quad \mathsf{true} : msg_i; \\
\quad esac
\end{array}
$$

The "*case* ... *esac*" expression in above formulas returns the first expression on the right hand side of ":", such that the corresponding condition on the left hand side evaluates to true [16]. For short, we write λ_x for the logic formula $x = init(x)$ and γ_x for the formula $x' = tran(x)$. Then Λ_{c_i} and Γ_{c_i} can be represented as:

$$\Lambda_{c_i} = \lambda_{state_i} \wedge \lambda_{msg_i}, \qquad\qquad \Gamma_{c_i} = \gamma_{state_i} \wedge \gamma_{msg_i}.$$

The formulas $init(state_i)$, $init(msg_i)$ and $next(msg_i)$ in the new model are identical to those in the old model. The only difference lies in the formula $next(state_i)$, which in the new model is:

$$
\begin{array}{l}
next(state_i) := \\
\quad case \\
\quad\quad (state_i = \texttt{idle}) \wedge msg_{1-i} : \texttt{recv}; \\
\quad\quad (state_i = \texttt{idle}) \wedge \neg msg_{1-i} : \{\texttt{idle, otgo}\}; \\
\quad\quad (state_i = \texttt{otgo}) \wedge msg_{1-i} \wedge \neg turn : \texttt{recv}; \\
\quad\quad (state_i = \texttt{otgo}) \wedge (\neg msg_{1-i} \vee turn) : \texttt{sent}; \\
\quad\quad (state_i = \texttt{recv}) \wedge msg_i : \texttt{otgo}; \\
\quad\quad (state_i = \texttt{recv}) \wedge \neg msg_i : \texttt{idle}; \\
\quad\quad (state_i = \texttt{sent}) : \texttt{idle}; \\
\quad esac
\end{array}
$$

3.2 Our Solutions

To take full advantage of commonalities between revisions, we propose to learn the contextual assumptions in a fine-grained fashion. Recall that M_{c_i} in the email system is represented using four predicate formulas, i.e., λ_{state_i}, γ_{state_i}, λ_{msg_i} and γ_{msg_i}. Instead of inferring the contextual assumption as a whole model [26], we suggest to learn it as these four formulas. Note that the former three formulas are identical in the updated model, the inferred conjectures for these three formulas can be safely reused. In this way, the chance of assumption reuse is improved.

We intend to learn the contextual assumptions also in a symbolic fashion. In [26], the contextual assumptions are represented as deterministic finite automata (DFA's). However, the DFA is not a compact representation of a model. A Boolean formula representable by a BDD having n nodes may need mn nodes even in its most compact DFA representation [23], where m is the number of variables in the formula. Learning models via their DFA representations is thus not an efficient approach. We utilize the learning technique in [21] to learn the BDD representation of contextual assumptions. The benefits are multiple folds. Firstly, the symbolic representation of a model is more compact. Recording and reusing the contextual assumption in its symbolic representation is thus more memory-efficient. Secondly, symbolic assumptions can be better adapted to the symbolic model checking. Finally, with the symbolic representations, the equivalence checking of models can be performed in a much more efficient way.

4 Fine-Grained Learning Technique

In this section, we propose a fine-grained learning technique for assume-guarantee verification. Let $M_U = \langle X, \Lambda, \Gamma \rangle$ be the *unknown* target model. Its initial condition Λ and transition relation Γ can oftentimes be represented as a set of logical formulas. Instead of inferring M_U as a DFA, or as two big logical formulas (i.e. Λ and Γ), we propose to infer it as a set of small logical formulas. Fine-grained learning technique will give us more chances to reuse the inferred results.

Without loss of generality, we assume Λ and Γ are decomposed into n predicate formulas: $\varphi_1, \varphi_2, \cdots, \varphi_n$. Define *templates* to be constructed inductively by logical operators and subscripted square parentheses ($[\bullet]_k$). Let ζ_Λ and ζ_Γ be two templates. With ζ_Λ and ζ_Γ, we can construct a contextual assumption from the purported formulas. For example, consider the templates $\zeta_\Lambda[\bullet]_1[\bullet]_2 = [\bullet]_1 \wedge [\bullet]_2$ and $\zeta_\Gamma[\bullet]_1[\bullet]_2 = [\bullet]_1 \wedge [\bullet]_2$ in the email system. Suppose $\overline{\lambda}_{state_0}$, $\overline{\lambda}_{msg_0}$, $\overline{\gamma}_{state_0}$ and $\overline{\gamma}_{msg_0}$ are the current purported formulas. The initial condition and transition relation of the contextual assumption can be constructed as $\zeta_\Lambda[\overline{\lambda}_{state_0}]_1[\overline{\lambda}_{msg_0}]_2 = \overline{\lambda}_{state_0} \wedge \overline{\lambda}_{msg_0}$, and $\zeta_\Gamma[\overline{\gamma}_{state_0}]_1[\overline{\gamma}_{msg_0}]_2 = \overline{\gamma}_{state_0} \wedge \overline{\gamma}_{msg_0}$, respectively.

The fine-grained learning model is shown in Fig. 3. For each subformula φ_i ($1 \leq i \leq n$), one instance of the learning algorithm is deployed. All learners make membership and equivalence queries to a mechanical teacher. Similar to the learning-based framework in Section 2.1, equivalence queries need be synchronized. When all learners get a conjecture, the mechanical teacher constructs

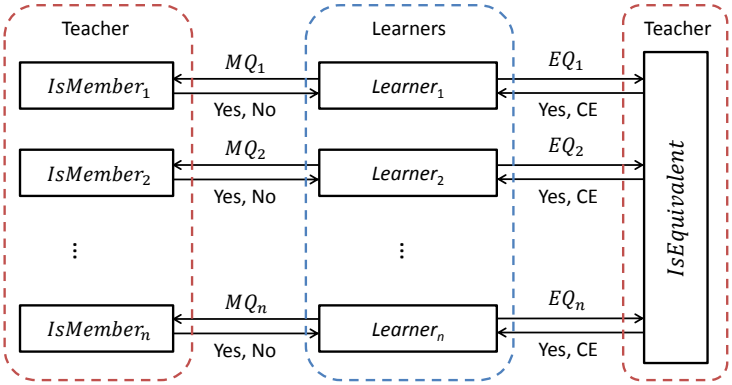


Fig. 3. The fine-grained learning framework

a contextual assumption (using ζ_A and ζ_Γ). If the constructed assumption fulfills both premises of the assume-guarantee reasoning rule (1), the verification is finished. Otherwise, the mechanical teacher helps the learners refine their conjectures by providing counterexamples.

Note that our fine-grained technique is not limited to the NuSMV language, and the target model is not necessary to be decomposed by variables (as in the email system example). To see an example, consider the ELTS (extended labelled transition systems with variables) model that is usually specified by transitions. Let k be the number of transitions in an ELTS model. Encoding each transition as a logical formula, the transition relation of the ELTS model is the disjunction of all transition formulas, and the template $\zeta_\Gamma = [\bullet]_1 \vee [\bullet]_2 \vee \cdots \vee [\bullet]_k$. Generally, we follow the syntactic structure to decompose the symbolic representation of the target model.

5 Assume-Guarantee Regression Verification

In this section, we discuss the data structures of contextual assumptions, propose our regression verification framework, and finally prove the correctness of our technique.

5.1 Data Structures of Contextual Assumptions

Our framework employs Nakamura's algorithm [23] to infer the BDD representation of contextual assumptions. Nakamura's algorithm is an instance of the active learning algorithm. Its basic procedure follows that discussed in Section. 2.1. When we say assumption reusing, we actually mean reusing the data structure of the learning algorithm. We thus discuss in the following the data structures used in Nakamura's algorithm.

Let D be the target (*reduced* and *ordered*) BDD with m variables. A BDD is a directed acyclic graph with one root node and two sink nodes. Each sink node is labeled with 0 or 1, and each non-sink node is labeled with a variable. A BDD can be regarded as a DFA. For any node of D, an *access string* u is a string that leads the BDD from its initial node to that node. Each node of D can be represented by its access string. In the following, we abuse the notation of u (and v) to represent both a node and its access string. For any two distinct nodes u and v, a *distinguishing string* w is a string such that uw reaches the terminal 1 and vw reach the terminal 0, or vice versa. Denote $nodes(D)$ the set of strings $a_1a_2\cdots a_k$ such that $k = m$ or the assignment of $x_1 \leftarrow a_1, x_2 \leftarrow a_2, \cdots, x_k \leftarrow a_k$ leads to a node labeled x_{k+1} in D. Let v be a string of length m, denote $D(v)$ the sink label that v reaches in D.

Two data structures are maintained in the BDD learning algorithm: a BDD with access strings (for short, BDDAS) S, and a set $T = \{T_1, T_2, \cdots, T_m\}$ of classification trees. A *BDDAS* is different from an ordinary BDD mainly in the following points: it may have a dummy root node; each of its nodes has an access string; each of its edges is labeled with a binary string. Denote $nodes_i^S(S)$ the set of access strings possessed by the non-dummy nodes in S whose length is i. Let $nodes^S(S) = \bigcup_{i=0}^{m} nodes_i^S(S)$. Let v be a string of length m, denote $S(v)$ the sink label that v reaches in S.

A *classification tree* T_i $(1 \leq i \leq m)$ decides which node in S a given string of length i will reach. It is composed of internal nodes and leaf nodes. Each internal node is labeled with a distinguishing string of length $m - i$, and each leaf node is labeled with either a special symbol μ, or an access string of length i that is possessed by a node of S. Any string α of length i is classified by T_i into one of its leaf nodes. Denote $T_i(\alpha)$ the leaf label into which α is classified. A string classified into a leaf node labeled with μ means that this string cannot reach any node in the corresponding OBDDAS.

A BDD can be obtained from a BDDAS. The obtained BDD is sent to the teacher for equivalence checking. If it passes the equivalence checking, we are done. Otherwise, a string is returned by the teacher as a counterexample. With this counterexample, the learner updates its BDDAS and classification trees. During the updating, the teacher's answers to membership queries are stored in classification trees. After updating, the cardinality of S (i.e. the number of nodes in S) increases by one. The target BDD is restored when the cardinality of S equals the number of nodes in the target BDD [23].

5.2 Regression Verification Framework

Our assume-guarantee regression verification algorithm is depicted in Algorithm 1. Before the new round of verification starts, an initialization step is performed, which attempts to reuse the contextual assumption inferred in the previous round of verification.

Let M_0 and M_1 be two components of a system. Recall that M_0 is the learning target. Assume M_0 is represented as n logical formulas: φ_1, φ_2, $\cdots$, φ_n. Let φ_i' $(1 \leq i \leq n)$ be the updated form of φ_i in M_0'. In the regression verification,

the algorithm checks for each i ($1 \leq i \leq n$) if φ'_i is equivalent to φ_i. If it is, the data structures (the BDDAS and classification trees) of the previous learner $Learner_{\varphi_i}$ is restored and used to initialize $Learner_{\varphi'_i}$. Otherwise, $Learner_{\varphi'_i}$ starts with empty data structures.

```
1 for 1 ≤ i ≤ m do
2 |   if φ'_i ≡ φ_i then
3 |   |   Learner_{φ'_i} ← Learner_{φ_i}
4 |   else
5 |   |   Initialize Learner_{φ'_i} with empty data structures
6 |   end
7 end
8 Use the technique in Section 4 to verify M'_0 ‖ M'_1 ⊨ φ ;
```

Algorithm 1: *IncrementalAG*(M'_0, M'_1, ϕ)

5.3 Correctness

We prove the correctness of our assume-guarantee regression verification framework in this subsection.

Let α_1, α_2 be two binary strings, we use $|\alpha_1|$ to denote the length of α_1, $pre(\alpha_1, i)$ the prefix string of α_1 with length i, and $\alpha_1 \cdot \alpha_2$ the concatenation of α_1 and α_2.

Definition 5. *A BDDAS S and a set $T = \{T_1, \cdots, T_m\}$ of classification trees are said* valid *for the target BDD D, if the following conditions are satisfied [23]:*

1. $nodes^S(S) \subseteq nodes(D)$;
2. $\forall v \in nodes^S_m(S)$, $S(v) = D(v)$;
3. $\forall v_1, v_2 \in nodes^S(S)$, *if* v_1 *and* v_2 *lead to the same node in* D, *there must be* $v_1 = v_2$;
4. $\forall v \in nodes^S(S)$, $T_{|v|}(v) = v$;
5. *for any binary string* α *of length* $i (1 \leq i \leq m)$, $\alpha \notin nodes(D) \Rightarrow T_i(\alpha) = \mu$;
6. *for any edge in* S *that is from* u *to* v *and labeled with* l,
   - $T_{|v|}(u \cdot l) = v$, *and*
   - $|u| < \forall j < |v|$, $T_j(u \cdot pre(l, j - |u|)) = \mu$.

Lemma 1 ([23]). *The Nakamura's learning algorithm terminates with a correct result starting from any BDDAS S and classification trees T_i for $i = 1, \cdots, m$ that are valid for the target BDD D.*

Theorem 2. *Given two BDD's D_1 and D_2, if $D_1 \equiv D_2$, the BDDAS S and classification trees T_i for $i = 1, \cdots, m$ generated by the learner of D_1 are valid for D_2.*

Recall that in our verification framwork, only results of equivalent formulas are reused. Theorem 2 is thus applicable. The correctness of our assume-guarantee regression verification framework (Algorithm 1) follows from Lemma 1 and Theorem 2.

Theorem 3 (Correctness). *The assume-guarantee regression verification algorithm (Algorithm 1) always terminates with a correct result.*

Note that our regression verification framework is not limited to Nakamura's algorithm [23]. Conceptually, any active learning algorithm can apply, such as the L^* algorithm for regular languages [2]. However, to be better suited for the fine-grained learning technique, an implicit learning algorithm is preferred. Alternatively, one can also use the CDNF learning algorithm [8] that infers Boolean functions.

6 Evaluation

A prototype of our regressional assume-guarantee verification technique was implemented on top of NuSMV 2.4.3 [16]. We have performed extensive experiments (in total, 1018 verification tasks from 108 revisions of 7 examples) to evaluate the efficiency of our technique. All experiments were conducted on a machine with 3.06GHz CPU and 2G RAM, running the Ubuntu 12.04 operation system.

A verification task is specified by a base model, an update to the base model, and a specification. It consists of two rounds of verifications. The contextual assumption inferred in the first round of verification (on the base model) can be optionally reused in the second round of verification (on the updated model). We compare the performance of the second round of verification with and without assumption reuse. The maximal run time is set to 3 hours.

The experiments are performed on seven examples, where *Gigamax* models a cache coherence protocol for the Gigamax multiprocess, *MSI* models a cache coherence protocol for consistence ensuring between processors and main memory, *Guidance* models the Shuttle Digital Autopilot engines out (3E/O) contingency guidance requirements, *SyncArb* models a synchronous bus arbiter, *Philo* models the dining philosophers problem [12], *Phone* models a simple phone system with four terminals [24], and *Lift* models the lift system in [5]. The former four examples are obtained from the NuSMV website[5], while the latter two are obtained from literatures. Each example model contains a number of interacting components. Our tool selects one component as M_0 and the composition of others as M_1.

We consider different degrees of changing a model: small changes (using mutations) and significant changes (with significant difference in the functionalities).

Two performance metrics are used in our experiments: (a) the run time (*Time*) for each verification run; (b) the number of membership queries ($|MQ|$)

[5] `http://nusmv.fbk.eu/examples/examples.html`

and the number of equivalence queries ($|EQ|$) raised in each verification run. Recall that answering learners' queries is the most costly operation in the learning-based verification framework, these two metrics are related to each other.

6.1 Results for Small Changes

Model changes are often small. We realized a program to randomly produce a number of mutations to a model either by introducing new variables or by changing the initial condition or transition relation of an existing variable.

This experiment was performed on five examples: *Gigamax*, *MSI*, *Guidance*, *SyncArb* and *Philo*. Results are shown in the upper part of Table 1. The columns $|Update|$, $|Spec.|$ and $|Task|$ list for each example the numbers of updates, specifications, and verification tasks, respectively. The following two column show the performance of the regression verification with and without assumption reuse respectively. All performance results (including the number of membership queries $|MQ|$, the number of equivalence queries $|EQ|$, and the run time) are given in average values over all tasks per example. The last column compares these two approaches. More experiment details of the highlighted example *SyncArb* will be discussed in Section 6.3. The experiment analysis is deferred to the next subsection. We will combine other examples' results and give a combined analysis.

6.2 Results for Significant Changes

During the evolution of a system, new features can be added to improve the original design. This kind of updates involves significant changes to the original model.

The second experiment was performed on two examples: *Phone* and *Lift*. These two examples were obtained from the software product-line engineering community [5,24]. For each example, there are a base model and a set of features. Each feature is considered as a significant change to the base model. Results of this experiment are shown in the bottom part of Table 1. The last *Total* row gives the average of respective values over all examples, including the examples mentioned in the former experiment and those in this experiment.

From Table 1, we observe an impressive improvement of our incremental approach with assumption reuse. Depending on examples, the average speed up of assumption reuse is between 1.26 to 3.79. Over all examples (with 1018 verification tasks in total), the average speed up is 3.47. We also find that the number of queries made by the incremental approach is greatly reduced compared to those without reuse. Over all examples, the average number of membership queries $|MQ|$ is reduced by a ratio of 2.89, and the average number of equivalence queries $|EQ|$ is reduced by a ratio of 3.44. Recall that answering learners' queries is the most costly operation in the learning-based assume-guarantee verification, these results conforms to those about run time.

There is no significant difference for the performance improvement of our incremental approach between the examples with small changes and others with

significant changes. This observation supports that our incremental approach is applicable to both degrees of model changes.

Table 1. Results for all examples: time in seconds

Example	|Update|	|Spec.|	|Task|	*with* Reuse			*without* Reuse			*without* / *with*		
				$\|MQ\|$	$\|EQ\|$	Time	$\|MQ\|$	$\|EQ\|$	Time	$\|MQ\|$	$\|EQ\|$	Time
Gigamax	35	6	210	657	12	58.41	2962	69	221.57	4.51	5.66	3.79
MSI	23	14	322	196	6	2.78	2695	97	5.99	13.74	16.26	2.15
Guidance	19	15	285	188	12	53.99	1197	82	199.15	6.37	6.76	3.69
SyncArb	10	2	19	1151	62	23.49	5336	425	81.22	4.64	6.85	3.46
Philo	7	1	7	4848	207	57.50	5834	245	72.37	1.20	1.18	1.26
Phone	7	17	119	9176	85	25.38	25917	328	51.66	2.82	3.86	2.04
Lift	7	8	56	40855	783	11.32	100455	1864	21.06	2.46	2.38	1.86
Total			1018	3625	63	32.47	10494	218	112.56	2.89	3.44	3.47

6.3 Results for a Single Example

Detailed results for *SyncArb* example are shown in Table 2. The *Sat.* column shows a pair of Boolean values ("T" for true, "F" for false), representing the satisfiability of the specification on the base model and the updated model, respectively. The term "**max**" in the last column denotes a divided-by-zero value. The bottom two rows report the sum and the average of the respective values over all verification tasks.

With assumption reuse, the numbers of membership queries $|MQ|$ and the number of equivalence queries $|EQ|$ are 0's in 15 out of 19 tasks. In other words, the reused assumptions immediately conclude the second round of verification in these tasks. This observation further witnesses the usability of assumption reuse to regression verification.

6.4 Impact of the Satisfiability Results to the Performance

Recall that when models change, the previously established (or falsified) specifications may become unsatisfied (or satisfied). We test in this experiment the impact of the satisifiability results to the efficiency of our incremental approach.

We group verification tasks of each example by their satisfiability results. In total, there are four types of groups: both true (denoted as (T, T)), true on the base model and false on the updated model (denoted as (T, F)), false on the base model and true on the updated model (denoted as (F, T)), and both false (denoted as (F, F)). Results are shown in Table 3, where $|Task|$ column lists the number of verification tasks in each group. Empty groups (with $|Task| = 0$) are omitted from the table.

We got very interesting findings from these results. The last column of Table 3 shows that the regression verification is most likely to be improved by

Table 2. Results for *SyncArb*: time in seconds

Spec.	Update	Sat.	*with* Reuse			*without* Reuse			*without* / *with*		
			$\|MQ\|$	$\|EQ\|$	Time	$\|MQ\|$	$\|EQ\|$	Time	$\|MQ\|$	$\|EQ\|$	Time
1	1	(T, F)	17	1	22.10	6561	551	81.24	385.94	551.00	3.68
	2	(T, F)	10083	584	85.45	14312	974	167.75	1.42	1.67	1.96
	4	(T, T)	0	0	10.80	6525	550	108.21	**max**	**max**	10.02
	5	(T, F)	7152	358	28.86	17017	1030	531.28	2.38	2.88	18.41
	6	(T, F)	0	0	184.56	6525	550	243.95	**max**	**max**	1.32
	7	(T, T)	0	0	5.77	6521	550	85.47	**max**	**max**	14.81
	8	(T, T)	0	0	7.79	6525	550	85.84	**max**	**max**	11.02
	9	(T, T)	0	0	8.43	6510	550	91.17	**max**	**max**	10.82
	10	(T, F)	4618	236	64.83	10693	746	114.12	2.32	3.16	1.76
2	1	(F, F)	0	0	2.64	1972	200	3.28	**max**	**max**	1.24
	2	(F, F)	0	0	2.99	1978	200	3.58	**max**	**max**	1.20
	3	(F, F)	0	0	2.47	1964	200	3.07	**max**	**max**	1.24
	4	(F, F)	0	0	2.85	1974	200	3.61	**max**	**max**	1.27
	5	(F, F)	0	0	2.74	1964	200	3.38	**max**	**max**	1.23
	6	(F, F)	0	0	2.71	1974	200	3.41	**max**	**max**	1.26
	7	(F, F)	0	0	2.74	1974	200	3.35	**max**	**max**	1.22
	8	(F, F)	0	0	2.73	1974	200	3.42	**max**	**max**	1.25
	9	(F, F)	0	0	3.05	1980	200	3.62	**max**	**max**	1.19
	10	(F, F)	0	0	2.76	2440	230	3.36	**max**	**max**	1.21
Sum			21870	1179	446.28	101383	8081	1543.11	4.64	6.85	3.46
Average			1151	62	23.49	5336	425	81.22	4.64	6.85	3.46

the assumption reuse if the specification was previously satisfied. There are two (F, T) groups (*Gigamax*, *Phone*) and two (F, F) groups (*Guidance*, *Philo*) on which the assumption reuse leads to notably performance degeneration. In contrast, the performance of the regression verification is always improved (or nearly improved) by assumption reuse in all (T, T) and (T, F) groups. We speculate the reasons as follows. Recall the assume-guarantee reasoning rule (1). If the specification is satisfied by the system, we need to find a contextual assumption to prove both premises in the rule. In contrast, if the specification is dissatisfied by the system, we need only an assumption that reveals a counterexample to the specification. Finding a counterexample is always much easier than proving the correctness. From the viewpoint of reuse, the assumption revealing a counterexample is certainly less useful than the one proving the correctness of the model.

The *Total* row in Table 3 gives that the average speedup of the incremental technique over all examples for (T, T), (T, F), (F, T) and (F, F) groups are 4.12, 1.75, 0.59, and 4.29, respectively. It further shows that the incremental technique tends to gets the best performance when the staisfiability of the specification are the same on both models. This phenomenon is also reasonable. Given that many behaviors are shared between these two models, the previously found proof (or

Table 3. Results grouped by the satisfiability results on the base and the updated models: time in seconds

Model	Sat.	|Task|	*with* Reuse			*without* Reuse			*without* / *with*		
			$\|MQ\|$	$\|EQ\|$	Time	$\|MQ\|$	$\|EQ\|$	Time	$\|MQ\|$	$\|EQ\|$	Time
Gigamax	(T, T)	139	0	0	37.12	0	0	55.47	-	-	1.49
	(T, F)	1	474	31	0.06	474	31	0.08	1.00	1.00	1.36
	(F, T)	23	447	10	60.41	1702	40	26.61	3.81	4.05	0.44
	(F, F)	47	2706	49	121.60	12392	286	812.92	4.58	5.89	6.69
MSI	(T, T)	180	77	2	4.88	3648	124	10.14	47.54	56.12	2.08
	(T, F)	4	56	5	0.04	216	18	0.07	3.89	4.00	1.51
	(F, T)	1	2808	83	4.97	4369	169	15.99	1.56	2.04	3.22
	(F, F)	137	338	10	0.09	1502	64	0.64	4.44	6.14	7.41
Guidance	(T, T)	211	0	0	27.86	968	66	211.70	**max**	**max**	7.60
	(T, F)	34	1425	93	176.80	2559	172	299.43	1.80	1.86	1.69
	(F, F)	40	127	7	87.47	1250	88	47.71	9.85	11.76	0.55
SyncArb	(T, T)	4	0	0	8.20	6520	550	92.67	**max**	**max**	11.31
	(T, F)	5	4374	236	77.16	11022	770	227.67	2.52	3.27	2.95
	(F, F)	10	0	0	2.77	2019	203	3.41	**max**	**max**	1.23
Philo	(T, T)	1	0	0	152.92	0	0	212.77	-	-	1.39
	(T, F)	5	6666	282	38.09	6666	282	57.24	1.00	1.00	1.50
	(F, F)	1	608	40	59.15	7511	300	7.58	12.35	7.50	0.13
Phone	(T, T)	77	11141	96	35.57	30783	367	76.63	2.76	3.82	2.15
	(F, T)	4	15747	159	62.96	20361	252	41.11	1.29	1.58	0.65
	(F, F)	38	4502	55	0.78	16643	257	2.17	3.70	4.69	2.77
Lift	(T, T)	10	497	8	0.37	185932	3229	51.60	374.41	419.35	138.71
	(T, F)	2	172319	3195	44.94	175354	3288	44.31	1.02	1.03	0.99
	(F, T)	3	444478	8530	152.80	451580	8780	149.01	1.02	1.03	0.98
	(F, F)	41	14752	287	2.00	50262	955	3.11	3.41	3.32	1.56
Total	(T, T)	623	1407	13	23.83	8213	159	98.24	5.84	12.57	4.12
	(T, F)	51	8804	239	130.93	10343	349	229.30	1.17	1.46	1.75
	(F, T)	31	45468	856	67.89	47732	918	39.99	1.05	1.07	0.59
	(F, F)	314	3042	57	30.03	11335	245	128.84	3.73	4.30	4.29
	1018		3621	63	32.44	10484	218	112.45	2.89	3.44	3.47

counterexample) is very likely to be a valid proof (or a valid counterexample) for the updated model.

7 Related Work

The first technique on learning-based assume-guarantee reasoning was proposed in [18], where the L^* algorithm [2] was adopted to learn the DFA representation of contextual assumptions. The L^*-based assume-guarantee reasoning was further optimized in different directions by many researchers, including [1,11,15,30]. An implicit learning framework for assume-guarantee reasoning was proposed in [14], where contextual assumptions are inferred in their symbolic representations. Both the BDD learning algorithm [23] and *CDNF* learning algorithm [8] have been adapted to this framework. Moreover, the techinque in [21] improves the implicit learning framework by a progressive witness analysis algorithm. In [20], the learning-based assume-guarantee reasoning was fruther applied to probabilistic model checking. Our technique contributes in assume-guarantee reasoning by providing a new fine-grained learning technique.

Regression verification was investigated mainly in two directions, the equivalence analysis, and the reuse of previously computed results. In the latter direction, a variety of information have been proposed for reuse in regression verification. In [22,29], the state-space graphs are recorded for reuse in latter verification runs. In [28], the intermediate results of a constraint solver are stored and reused. In [7], the abstraction precision used for performing predicate abstraction on previous program is reused. Note that the precision reuse technique is orthogonal to ours. Our technique contributes in this area by integrating regression verification and automated assume-guarantee reasoning.

The most relevant work to ours are [9,26]. They used the idea of assumption reuse to solve the dynamic component substitutability problem. Their technique requires $M_0' = M_0$ and M_1' simulates M_1. This is surely a severe limitation. We removed this limitation by fine-grained learning technique. With our technique, the assume-guarantee regression verification is enabled.

8 Conclusions and Future Work

We presented in this paper a learning-based assume-guarantee regression verification technique. With this technique, contextual assumptions of the previous round of verification can be efficiently reused in the current verification. Correctness of this techniques is established. Experimental results (with 1018 verification tasks) show significant improvements of our technique.

Currently, we implemented a prototype of our technique on top of NuSMV. We are considering to extend this technique to a component-based modeling language that allows hierarchical components and sophisticated interactions. We are also planning to integrate our technique with predicate abstraction, and then apply it to program verification.

References

1. Alur, R., Madhusudan, P., Nam, W.: Symbolic compositional verification by learning assumptions. In Etessami, K., Rajamani, S.K., eds.: Computer Aided Verification. Volume 3576 of Lecture Notes in Computer Science., Springer (2005) 548–562
2. Angluin, D.: Learning regular sets from queries and counterexamples. Information and Computation **75**(2) (1987) 87–106
3. Backes, J., Person, S., Rungta, N., Tkachuk, O.: Regression verification using impact summaries. In: Model Checking Software. Springer (2013) 99–116
4. Baier, C., Katoen, J.P.: Principles of model checking. MIT Press (2008)
5. Berry, M.: Proving properties of the lift system. Master's thesis, School of Computer Science, University of Birmingham **199**(6) (1996)
6. Beyer, D., Löwe, S., Novikov, E., Stahlbauer, A., Wendler, P.: Precision reuse for efficient regression verification. In: Proceedings of the 2013 9th Joint Meeting on Foundations of Software Engineering, ACM (2013) 389–399
7. Beyer, D., Wendler, P.: Reuse of verification results. In: Model Checking Software. Springer (2013) 1–17
8. Bshouty, N.H.: Exact learning Boolean function via the monotone theory. Information and Computation **123**(1) (1995) 146–153
9. Chaki, S., Clarke, E., Sharygina, N., Sinha, N.: Verification of evolving software via component substitutability analysis. Formal Methods in System Design **32**(3) (2008) 235–266
10. Chaki, S., Gurfinkel, A., Strichman, O.: Regression verification for multi-threaded programs. In: Verification, Model Checking, and Abstract Interpretation, Springer (2012) 119–135
11. Chaki, S., Strichman, O.: Optimized L^*-based assume-guarantee reasoning. In Grumberg, O., Huth, M., eds.: Tools and Algorithms for the Construction and Analysis of Systems. Volume 4424 of Lecture Notes in Computer Science., Springer (2007) 276–291
12. Chaum, D.: The dining cryptographers problem: Unconditional sender and recipient untraceability. Journal of Cryptology **1**(1) (1988) 65–75
13. Chen, Y.F., Clarke, E.M., Farzan, A., He, F., Tsai, M.H., Tsay, Y.K., Wang, B.Y., Zhu, L.: Comparing learning algorithms in automated assume-guarantee reasoning. In: Leveraging Applications of Formal Methods, Verification and Validation (1). Volume 6415 of Lecture Notes in Computer Science., Springer (2010) 643–657
14. 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.: Computer Aided Verification. Volume 6174 of Lecture Notes in Computer Science., Springer (2010) 511–526
15. 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.: Tools and Algorithms for the Construction and Analysis of Systems. Volume 5505 of Lecture Notes in Computer Science., Springer (2009) 31–45
16. Cimatti, A., Clarke, E., Giunchiglia, E., Giunchiglia, F., Pistore, M., Roveri, M., Sebastiani, R., Tacchella, A.: NuSMV 2: An opensource tool for symbolic model checking. In: Computer Aided Verification, Springer (2002) 359–364
17. Clarke, E.M., Grumberg, O., Peled, D.: Model checking. MIT press (1999)
18. Cobleigh, J.M., Giannakopoulou, D., Păsăreanu, C.S.: Learning assumptions for compositional verification. In Garavel, H., Hatcliff, J., eds.: Tools and Algorithms for the Construction and Analysis of Systems. Volume 2619 of Lecture Notes in Computer Science., Springer (2003) 331–346

19. Gavaldà, R., Guijarro, D.: Learning ordered binary decision diagrams. In Jantke, K.P., Shinohara, T., Zeugmann, T., eds.: Algorithmic Learning Theory. Volume 997 of Lecture Notes in Computer Science., Springer (1995) 228–238
20. He, F., Gao, X., Wang, B.Y., Zhang, L.: Leveraging weighted automata in compositional reasoning about concurrent probabilistic systems. In: Proceedings of the 42nd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, ACM (2015) 503–514
21. He, F., Wang, B.Y., Yin, L., Zhu, L.: Symbolic assume-guarantee reasoning through BDD learning. In: ICSE, ACM (2014) 1071–1082
22. Lauterburg, S., Sobeih, A., Marinov, D., Viswanathan, M.: Incremental state-space exploration for programs with dynamically allocated data. In: Proceedings of the 30th international conference on Software engineering, ACM (2008) 291–300
23. Nakamura, A.: An efficient query learning algorithm for ordered binary decision diagrams. Information and Computation **201**(2) (2005) 178–198
24. Plath, M., Ryan, M.: Feature integration using a feature construct. Science of Computer Programming **41**(1) (2001) 53–84
25. Sery, O., Fedyukovich, G., Sharygina, N.: Incremental upgrade checking by means of interpolation-based function summaries. In: Formal Methods in Computer-Aided Design (FMCAD), 2012, IEEE (2012) 114–121
26. Sharygina, N., Chaki, S., Clarke, E., Sinha, N.: Dynamic component substitutability analysis. In: FM 2005: Formal Methods. Springer (2005) 512–528
27. Strichman, O., Godlin, B.: Regression verification-a practical way to verify programs. In: Verified Software: Theories, Tools, Experiments. Springer (2008) 496–501
28. Visser, W., Geldenhuys, J., Dwyer, M.B.: Green: reducing, reusing and recycling constraints in program analysis. In: Proceedings of the ACM SIGSOFT 20th International Symposium on the Foundations of Software Engineering, ACM (2012) 58
29. Yang, G., Dwyer, M.B., Rothermel, G.: Regression model checking. In: Software Maintenance, 2009. ICSM 2009. IEEE International Conference on, IEEE (2009) 115–124
30. Zhu, H., He, F., Hung, W.N., Song, X., Gu, M.: Data mining based decomposition for assume-guarantee reasoning. In: FMCAD, IEEE (2009) 116–119