VNU Journal of Science: Comp. Science & Com. Eng, Vol. 34, No. 2 (2018) 16–32

VNU Journal of Science:
Computer Science and Communication Engineering

Journal homepage: http://www.jcsce.vnu.edu.vn/index.php/jcsce

On Locally Strongest Assumption Generation Method for Component-Based Software Verification

Hoang-Viet Tran*, Pham Ngoc Hung

Faculty of Information Technology, VNU University of Engineering and Technology, No. 144 Xuan Thuy Street, Dich Vong Ward, Cau Giay District, Hanoi, Vietnam

Abstract

Assume-guarantee reasoning, a well-known approach in component-based software (CBS) verification, is in fact a language containment problem whose computational cost depends on the sizes of languages of the software components under checking and the assumption to be generated. Therefore, the smaller language assumptions, the more computational cost we can reduce in software verification. Moreover, strong assumptions are more important in CBS verification in the context of software evolution because they can be reused many times in the verification process. For this reason, this paper presents a method for generating locally strongest assumptions with locally smallest languages during CBS verification. The key idea of this method is to create a variant technique for answering membership queries of the *Teacher* when responding to the *Learner* in the L^*–based assumption learning process. This variant technique is then integrated into an algorithm in order to generate locally strongest assumptions. These assumptions will effectively reduce the computational cost when verifying CBS, especially for large–scale and evolving ones. The correctness proof, experimental results, and some discussions about the proposed method are also presented.

Received 14 June 2018, Revised 18 September 2018, Accepted 15 October 2018

Keywords: Assume-guarantee reasoning, Model checking, Component-based software verification, Locally strongest assumptions, Locally smallest language assumptions.

1. Introduction

The assume-guarantee verification proposed in [1–5] has been recognized as a promising, incremental, and fully automatic method for modular verification of CBS by model checking [6]. This method decomposes a verification target about a CBS into smaller parts corresponding to the individual components such that we can model check each of them separately. Thus, the method has a potential to deal with the state explosion problem in model checking. The key idea of this method is to generate an assumption such that the assumption is strong enough for the component to satisfy a required property and weak enough to be discharged by the rest of the software. The most common rule that is used in assume-guarantee verification is the non-circular rule as shown in formula 1. Given a CBS $M = M_1 \parallel M_2$, and a predefined property p, we need to find an assumption A so

* Corresponding author. Email.: vietth2004@gmail.com
https://doi.org/10.25073/2588-1086/vnucsce.209

that formula 1 holds.

$$\frac{\begin{array}{c} M_1 \parallel A \models p \\ M_2 \models A \end{array}}{M_1 \parallel M_2 \models p} \tag{1}$$

This is actually the language containment problem of the two couples of components ($M_1 \parallel A$, p), and (M_2, A), i.e., to decide if $L(M_1 \parallel A)_{\uparrow\Sigma_p} \subseteq L(p)$, and $L(M_2)_{\uparrow\Sigma_A} \subseteq L(A)$, where $\parallel$ is the parallel composition operator defined in Definition 4, $\models$ and $\uparrow \Sigma$ is the satisfiability and projection operator defined in Definition 6, respectively. Therefore, the stronger the assumption (i.e., an assumption with smaller language) is, the more computational cost can be reduced, especially when model checking large-scale CBSs. Furthermore, when a component is evolved in the context of the software evolution, we can recheck the evolved CBS effectively by reusing the generated stronger assumptions. As a result, generating assumptions with as small as possible languages is of primary importance for assume-guarantee verification of CBSs.

Although the assumption generation method proposed in [2] has already tried to generate stronger assumptions than those generated by the method proposed in [1], it has not been able to generate strongest assumptions. This is because the method proposed in [2] uses a learning algorithm called L^* [7, 8] for learning regular languages. In fact, L^* algorithm depends on a *minimally adequate Teacher* for being able to generate the strongest assumptions (i.e., the assumptions with minimal languages). Therefore, the algorithms that implement $Teacher$ will affect the languages of the generated assumptions. On the other hand, in the context of software compositional verification, depends on the implementation of $Teacher$, L^* learning algorithm always terminates and returns the first assumption that satisfies the assume-guarantee rules before reaching the strongest assumptions. As a result, the assumptions generated by the assume-guarantee verification method proposed in [2] are not the strongest ones. In addition, in fact, there exist many candidate assumptions satisfying the assume-guarantee rules. Section 4 shows a counterexample that there exists another assumption (denoted by A_{LS}) which is stronger than the assumption A generated by the L^*–based assumption generation method proposed in [2] (i.e., $L(A_{LS})_{\uparrow\Sigma_A} \subseteq L(A)$). The problem is how to find the strongest assumptions (i.e., assumptions with smallest languages) in the space of candidate assumptions.

Recently, there are many researches that have been proposed in improvement of the L^*–based assumption generation method proposed in [2]. In the series of papers presented in [9–11], Hung et al. proposes a method that can generate the state minimal assumptions (i.e., assumptions with the smallest number of states) using the depth-limited search. However, this does not guarantee that these assumptions have the smallest languages. In 2007, Chaki and Strichman proposed three optimizations to the L^*–based assumption generation method in which they proposed a method to minimize the alphabet used by the assumption that allows us to reduce the sizes of the generated assumptions [12]. Nonetheless, in [12], the size of languages of the generated assumptions is not guaranteed to be smaller than the size of those generated by the L^*–based assumption generation method proposed in [2]. In [13], Gupta et al. proposed a method to compute an exact minimal automaton to act as an intermediate assertion in assume-guarantee reasoning, using a sampling approach and a Boolean satisfiability solver. However, this automaton is not the stronger assumption with smaller language and this method is suitable for hardware verification. Therefore, from the above researches, we can see that although generating stronger assumptions is a very important problem, there is no research into this so far.

For the above reasons, the purpose of this paper is to generate the strongest assumptions for compositional verification. However, for some reasons which will be explained in more details in Section 4, the proposed method can only generate the locally strongest ones. The method is based on an observation that the technique to answer membership queries from $Learner$ of $Teacher$ uses the language of the weakest assumption, denoted by $L(A_W)$, to decide whether to return $true$ or $false$ to $Learner$ [2]. If a trace s belongs to $L(A_W)$, it returns $true$ even if s may not belongs to the language of the assumption to be generated. For this reason, the key idea of the proposed technique for answering membership queries is that $Teacher$ will not directly return $true$ to the query. It will return "?" to $Learner$ whenever the trace s belongs to $L(A_W)$. Otherwise, it will return $false$. After that, this technique is integrated into an improved L^*–based algorithm that tries every possibility that a trace belongs to language of the assumption A to be generated. For this purpose, at the i^{th} iteration of the learning process, when the observation table (S, E, T) is closed with n "?" results, we have the corresponding candidate assumption A_i where all "?" results are considered as $true$. We decide if (S, E, T) is closed with the consideration that all "?" results are $true$, this is the same as the assumption generation method proposed in [2]. The algorithm tries every k–combination of n "?" results and considers those "?" results as $false$ (i.e., the corresponding traces do not belong to $L(A)$), where k is from n (all "?" results are $false$) to 1 (one "?" result is $false$). If none of these k–combinations is corresponding to a satisfied assumption, the algorithm will turn all "?" results into $true$ (all corresponding traces belong to $L(A)$) and generate corresponding candidate assumption A_i then ask an equivalence query for A_i. After that, the algorithm continues the learning process again for the next iteration. The algorithm terminates as soon as it reaches a conclusive result. Consequently, with this method of assumption generation, the generated assumptions, if exists, will be the locally strongest assumptions.

The rest of this paper is organized as follows. Section 2 presents background concepts which will be used in this paper. Next, Section 3 reviews the L^*–based assumption generation method for compositional verification. After that, Section 4 describes the proposed method to generate locally strongest assumptions. We prove the correctness of the proposed method in Section 5. Experimental results and discussions are presented in Section 6. Related works to the paper are also analyzed in Section 7. Finally, we conclude the paper in Section 8.

2. Background

In this section, we present some basic concepts which will be used in this work.

LTSs. This research uses *Labeled Transition Systems* (LTSs) to model behaviors of components. Let $\mathcal{A}ct$ be the universal set of observable actions and let τ denote a local action unobservable to a component environment. We use π to denote a special error state. An LTS is defined as follows.

Definition 1. *(LTS). An LTS M is a quadruple $\langle Q, \Sigma, \delta, q_0 \rangle$, where:*

- *Q is a non-empty set of states,*
- *$\Sigma \subseteq \mathcal{A}ct$ is a finite set of observable actions called the alphabet of M,*
- *$\delta \subseteq Q \times \Sigma \cup \{\tau\} \times Q$ is a transition relation, and*
- *$q_0 \in Q$ is the initial state.*

Definition 2. *(Trace). A trace σ of an LTS $M = \langle Q, \Sigma, \delta, q_0 \rangle$ is a finite sequence of actions $a_1 a_2 ... a_n$, such that there exists a sequence of states starting at the initial state (i.e., $q_0 q_1 ... q_n$) such that for $1 \leq i \leq n$, $(q_{i-1}, a_i, q_i) \in \delta$, $q_i \in Q$.*

Definition 3. *(Concatenation operator). Given two sets of event sequences P and Q, $P.Q = \{pq \mid p \in P, q \in Q\}$, where pq presents the concatenation of the event sequences p and q.*

Note 1. *The set of all traces of M is called the language of M, denoted by $L(M)$. Let $\sigma = a_1 a_2 ... a_n$ be a finite trace of an LTS M. We use*

$[\sigma]$ to denote the LTS $M_\sigma = \langle Q, \Sigma, \delta, q_0\rangle$ with $Q = \{q_0, q_1, ..., q_n\}$, and $\delta = \{(q_{i-1}, a_i, q_i)\}$, where $1 \leq i \leq n$.

Parallel Composition. The parallel composition operator $\|$ is a commutative and associative operator up-to language equivalence that combines the behavior of two models by synchronizing the common actions to their alphabets and interleaving the remaining actions.

Definition 4. *(Parallel composition operator). The parallel composition between $M_1 = \langle Q_1, \Sigma_{M_1}, \delta_1, q_0^1\rangle$ and $M_2 = \langle Q_2, \Sigma_{M_2}, \delta_2, q_0^2\rangle$, denoted by $M_1\|M_2$, is defined as follows. $M_1\|M_2$ is equivalent to $\prod$ if either M_1 or M_2 is equivalent to $\prod$, where $\prod$ denotes the LTS $\langle\{\pi\}, Act, , \pi\rangle$. Otherwise, $M_1\|M_2$ is an LTS $M = \langle Q, \Sigma, \delta, q_0\rangle$ where $Q = Q_1 \times Q_2$, $\Sigma = \Sigma_{M_1} \cup \Sigma_{M_2}$, $q_0 = (q_0^1, q_0^2)$, and the transition relation δ is given by the following rules:*

$$\text{(i)}\frac{\alpha \in \Sigma_{M_1} \cap \Sigma_{M_2}, (p, \alpha, p') \in \delta_1, (q, \alpha, q') \in \delta_2}{((p, q), \alpha, (p', q')) \in \delta} \tag{2}$$

$$\text{(ii)}\frac{\alpha \in \Sigma_{M_1} \backslash \Sigma_{M_2}, (p, \alpha, p') \in \delta_1}{((p, q), \alpha, (p', q)) \in \delta} \tag{3}$$

$$\text{(iii)}\frac{\alpha \in \Sigma_{M_2} \backslash \Sigma_{M_1}, (q, \alpha, q') \in \delta_2}{((p, q), \alpha, (p, q')) \in \delta} \tag{4}$$

Safety LTSs, Safety Property, Satisfiability and Error LTSs.

Definition 5. *(Safety LTS). A safety LTS is a deterministic LTS that contains no state that is equivalent to π state.*

Note 2. *A safety property asserts that nothing bad happens for all time. A safety property p is specified as a safety LTS $p = \langle Q, \Sigma_p, \delta, q_0\rangle$ whose language $L(p)$ defines the set of acceptable behaviors over Σ_p.*

Definition 6. *(Satisfiability). An LTS M satisfies p, denoted by $M \models p$, if and only if $\forall\sigma \in L(M)$: $(\sigma_{\uparrow\Sigma_p}) \in L(p)$, where $\sigma_{\uparrow\Sigma_p}$ denotes the trace obtained by removing from σ all occurrences of actions $a \notin \Sigma_p$.*

Note 3. *When we check whether an LTS M satisfies a required property p, an* error LTS, *denoted by p_{err}, is created which traps possible violations with the π state. p_{err} is defined as follows:*

Definition 7. *(Error LTS). An* error LTS *of a property $p = \langle Q, \Sigma_p, \delta, q_0\rangle$ is $p_{err} = \langle Q \cup \{\pi\}, \Sigma_p, \delta', q_0\rangle$, where $\delta' = \delta \cup \{(q, a, \pi) \mid a \in \Sigma_p$ and $\nexists q' \in Q : (q, a, q') \in \delta\}$.*

Remark 1. *The error LTS is complete, meaning each state other than the error state has outgoing transitions for every action in the alphabet. In order to verify a component M satisfying a property p, both M and p are represented by safety LTSs, the parallel compositional system $M\|p_{err}$ is then computed. If some states (q, π) are reachable in the compositional system, M violates p. Otherwise, it satisfies p.*

Definition 8. *(Deterministic finite state automata) (DFA). A DFA D is a five tuple $\langle Q, \Sigma, \delta, q_0, F\rangle$, where:*

- *Q, Σ, δ, q_0 are defined as for deterministic LTSs, and*
- *$F \subseteq Q$ is a set of accepting states.*

Note 4. *Let D be a DFA and σ be a string over Σ. We use $\delta(q, \sigma)$ to denote the state that D will be in after reading σ starting from the state q. A string σ is accepted by a DFA $D = \langle Q, \Sigma, \delta, q_0, F\rangle$ if $\delta(q_0, \sigma) \in F$. The set of all string σ accepted by D is called the language of D (denoted by $L(D)$). Formally, we have $L(D) = \{\sigma \mid \delta(q_0, \sigma) \in F\}$.*

Definition 9. *(Assume-Guarantee Reasoning). Let M be a system which consists of two components M_1 and M_2, p be a safety property, and A be an assumption about M_1's environment. The assume-guarantee rules are described as following formula [2].*

$$\frac{\begin{array}{c}(step\,1)\ \langle A\rangle\ M_1\ \langle p\rangle \\ (step\,2)\ \langle true\rangle\ M_2\ \langle A\rangle\end{array}}{\langle true\rangle\ M_1||M_2\ \langle p\rangle}$$

Note 5. *We use the formula* $\langle true\rangle\ M\ \langle A\rangle$ *to represent the compositional formula* $M\|A_{err}$*. The formula* $\langle A\rangle\ M\ \langle p\rangle$ *is* $true$ *if whenever* M *is part of a system satisfying* A*, then the system must also guarantee* p*. In order to check the formula, where both* A *and* p *are safety LTSs, we compute the compositional formula* $A\|M\|p_{err}$ *and check if the error state* π *is reachable in the composition. If it is, the formula is violated. Otherwise it is satisfied.*

Definition 10. *(Weakest Assumption) [1]. The weakest assumption* A_W *describes exactly those traces over the alphabet* $\Sigma = (\Sigma_{M_1} \cup \Sigma_p) \cap \Sigma_{M_2}$ *which, the error state* π *is not reachable in the compositional system* $M_1\|p_{err}$*. The weakest assumption* A_W *means that for any environment component* E*,* $M_1\|E \models p$ *if and only if* $E \models A_W$*.*

Definition 11. *(Strongest Assumption). Let* A_S *be an assumption that satisfies the assume-guarantee rules in Definition 9. If for all* A *satisfying the assume-guarantee rules in Definition 9:* $L(A_S)_{\uparrow\Sigma_A} \subseteq L(A)$*, we call* A_S *the strongest assumption.*

Note 6. *Let* $\mathfrak{A}$ *be a subset of assumptions that satisfy the assume-guarantee rules in Definition 9 and* $A_{LS} \in \mathfrak{A}$*. If for all* $A \in \mathfrak{A}$*:* $L(A_{LS})_{\uparrow\Sigma_A} \subseteq L(A)$*, we call* A_{LS} *the locally strongest assumption.*

Definition 12. *(Observation table). Given a set of alphabet symbols* Σ*, an observation table is a 3-tuple* (S, E, T)*, where:*

- $S \in \Sigma^*$ *is a set of prefixes,*
- $E \in \Sigma^*$ *is a set of suffixes, and*
- $T : (S \cup S.\Sigma).E \rightarrow \{true, false\}$*. With a string* $s \in \Sigma^*$*,* $T(s) = true$ *means* $s \in L(A)$*, otherwise* $s \notin L(A)$*, where* A *is the corresponding assumption to* (S, E, T)*.*

An observation table is closed if $\forall s \in S, \forall a \in \Sigma, \exists s' \in S, \forall e \in E : T(sae) = T(s'e)$. In this case, s' presents the next state from s after seeing a, sa is indistinguishable from s' by any of suffixes. Intuitively, an observation table (S, E, T) is closed means that every row $sa \in S.\Sigma$ has a matching row s' in S.

When an observation table (S, E, T) over an alphabet Σ is closed, we define the corresponding DFA that accepts the associated language as follows [7]. $M = \langle Q, \Sigma_M, \delta, q_0, F\rangle$, where

- $Q = \{row(s) : s \in S\}$,
- $q_0 = row(\lambda)$,
- $F = \{row(s) : s \in S$ and $T(s) = 1\}$,
- $\Sigma_M = \Sigma$, and
- $\delta(row(s), a) = row(s.a)$.

From this way of constructing DFA from an observation table (S, E, T), we can see that each states of the DFA which is being created is corresponding to one row in S. Therefore, from now on, we sometimes call the rows in (S, E, T) its states.

Remark 2. *The DFAs generated from observation table in this context are complete, minimal, and prefix-closed (an automaton* D *is prefix-closed if* $L(D)$ *is prefix-closed, i.e., for every* $\sigma \in L(D)$*, every prefix of* σ *is also in* $L(D)$*). Therefore, these DFAs contain a single non-accepting state (denoted by* nas*) [2]. Consider a DFA* $D = \langle Q \cup \{nas\}, \Sigma, \delta, q_0, Q\rangle$ *in this context, we can calculate the corresponding safety LTS* A *by removing the non-accepting state* nas *and all of its ingoing transitions. Formally, we have* $A = \langle Q, \Sigma, \delta \cap (Q \times \Sigma \times \{nas\}), q_0\rangle$*.*

3. L^*–based assumption generation method

3.1. The L^* algorithm

L^* algorithm [7] is an incremental learning algorithm that is developed by Angluin and later improved by Rivest and Schapire [8]. L^* can learn an unknown regular language and generate a deterministic finite automata (DFA) that accepts it. The key idea of L^* learning algorithm is based on the "*Myhill Nerode Theorem*" [14] in the formal languages theory. It said that for every regular set $U \subseteq \Sigma^*$,

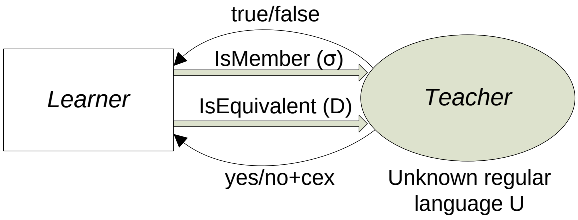

Figure 1. The interaction between L^* $Learner$ and $Teacher$.

there exists a ***unique, minimal deterministic automaton*** whose states are isomorphic to the set of equivalence classes of the following relation: $w \approx w'$ if and only if $\forall u \in \Sigma^* : wu \in U \Leftrightarrow w'u \in U$. Therefore, the main idea of L^* is to learn equivalence classes, i.e., two prefixes are not in the same class if and only if there is a distinguishing suffix u.

Let U be an unknown regular language over some alphabet Σ. L^* will produce a DFA D such that $L(D) = U$. In this learning model, the learning process is performed by the interaction between the two objects $Learner$ (i.e., L^*) and $Teacher$. The interaction is shown in Figure 1 [17]. $Teacher$ is an oracle that must be able to answer the following two types of queries from $Learner$.

- Membership queries: These queries consist of a string $\sigma \in \Sigma^*$ (i.e., "is $\sigma \in U$?"). The answer is $true$ if $\sigma \in U$, and $false$ otherwise.
- Equivalence queries: These queries consist of a candidate DFA D whose language the algorithm believes to be identical to U ("is $L(D) = U$?"). The answer is YES if $L(D) = U$. Otherwise $Teacher$ returns NO and a counterexample cex which is a string in the symmetric difference of $L(D)$ and U.

3.2. Generating assumption using L^* algorithm

Given a CBS M that consists of two components M_1 and M_2 and a safety property p. The L^*–based assumption generation algorithm proposed in [2, 17] generates a contextual assumption using the L^* algorithm [7]. The details of this algorithm are shown in Algorithm 1. In order to learn an assumption

Algorithm 1: L^*–based assumption generation algorithm

1 **begin**
2 Let $S = E = \{\lambda\}$
3 **while** $true$ **do**
4 Update T using membership queries
5 **while** (S, E, T) *is not closed* **do**
6 Add sa to S to make (S, E, T) closed where $s \in S$ and $a \in \Sigma$
7 Update T using membership queries
8 **end**
9 Construct candidate DFA D from (S, E, T)
10 Make the $conjecture$ C from D
11 $equiResult \leftarrow$ Ask an equivalence query for the $conjecture$ C
12 **if** $equiResult.Key$ *is* YES **then**
13 **return** C
14 **else if** $equiResult.Key$ *is* $UNSAT$ **then**
15 **return** $UNSAT + cex$
16 **else**
/* $Teacher$ returns $NO + cex$ */
17 Add $e \in \Sigma^*$ that witnesses the counterexample to E
18 **end**
19 **end**
20 **end**

A, Algorithm 1 maintains an observation table (S, E, T). The algorithm starts by initializing S and E with the empty string λ (line 2). After that, the algorithm updates (S, E, T) by using membership queries (line 4). While the observation table is not closed, the algorithm continues adding sa to S and updating the observation table to make it closed (from line 5 to line 8). When the observation table is closed, the algorithm creates a conjecture C from the closed table (S, E, T) and asks an *equivalence query* to $Teacher$ (from line 9 to line 11). The algorithm then stores the result of candidate query to $equiResult$. An equivalence query result contains two properties: $Key \in \{YES, NO, UNSAT\}$ (i.e., YES means the corresponding assumption satisfies the assume-guarantee rules in Definition 9; NO means the corresponding assumption does not satisfy assume-guarantee rules in Definition 9, however, at this point, we could not decide if the given system M does not satisfy p yet, we can use the corresponding counterexample cex to generate a new candidate assumption; $UNSAT$ means the given system M does not satisfy p and the counterexample is cex); the other property is an assumption when Key is YES or a counterexample cex when Key is NO or $UNSAT$. If $equiResult.Key$ is YES (i.e., C is the needed assumption), the algorithm stops and returns C (line 13). If $equiResult.Key$ is $UNSAT$, the algorithm will stops and returns $UNSAT$ and cex is the corresponding counterexample. Otherwise, if $equiResult.Key$ is NO, it analyzes the returned counterexample cex to find a suitable suffixes e. This suffix e must be such that adding it to E will cause the next assumption candidate to reflect the difference and keep the set of suffix E closed. The method to find e is not in the scope of this paper, please find more details in [8]. It then adds e to E (line 17) and continues the learning process again from line 4. The incremental composition verification during the iteration i^{th} is shown in Figure 2 [2, 17].

In order to answer a membership query whether a trace $\sigma = a_1 a_2 ... a_n$ belongs to $L(A)$ or not, we create an LTS $[\sigma] = \langle Q, \Sigma, \delta, q_0 \rangle$ with $Q = \{q_0, q_1, ..., q_n\}$, and $\delta = \{(q_{i-1}, a_i, q_i)\}$, where $1 \leq i \leq n$. $Teacher$ then checks the formula $\langle [\sigma] \rangle M_1 \langle p \rangle$ by computing compositional system $[\sigma] || M_1 || p_{err}$. If the error state π is unreachable, $Teacher$ returns yes (i.e., $\sigma \in L(A)$). Otherwise, $Teacher$ returns no (i.e., $\sigma \notin L(A)$).

In regards to dealing with equivalence queries, as mentioned above in Section 3.1, these queries are handled in $Teacher$ by comparing $L(A) = U$. However, in case of assume-guarantee reasoning, we have not known what is U yet. The only thing we know is that the assumption A to be generated must satisfy the assume-guarantee rules in Definition 9. Therefore, instead of checking $L(A) = U$, we check if A satisfies the assume-guarantee rules in Definition 9.

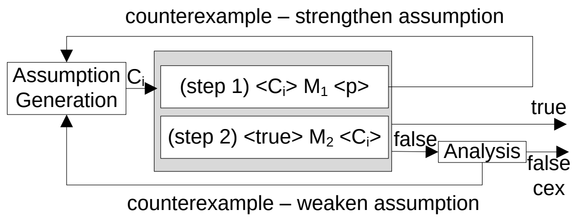

Figure 2. Incremental compositional verification during iteration i^{th}.

4. Learning locally strongest assumptions

As mentioned in Section 1, the assumptions generated by the L^*–based assumption generation method proposed in [2] are not strongest. In the counterexample shown in Figure 3, given two component models M_1, M_2, and a required safety property p, the L^*–based assumption generation method proposed in [2] generates the assumption A. However, there exists a stronger assumption A_{LS} with $L(A_{LS})_{\uparrow\Sigma_A} \subseteq L(A)$ as shown in Figure 3. We have checked $L(A_{LS})_{\uparrow\Sigma_A} \subseteq L(A)$ by using the tool named LTSA [15, 16]. For this purpose, we described A as a *property* and checked if $A_{LS} \models A$ using LTSA. The result is correct. This means that $L(A_{LS})_{\uparrow\Sigma_A} \subseteq L(A)$.

The original purpose of this research is to generate the strongest assumptions for assume-guarantee reasoning verification of CBS. However, in the space of assumptions that satisfy the assume-guarantee reasoning rule in Definition 9, there can be a lot of assumptions. Moreover, we cannot compare the languages of two arbitrary assumptions in general. This is because given two arbitrary assumptions A_1 and A_2, we can have a scenario that $L(A_1) \not\subseteq L(A_2)$ and $L(A_2) \not\subseteq L(A_1)$ but $L(A_1) \cap L(A_2) \neq \emptyset$ and $L(A_1) \cap L(A_2)$ is not an assumption. In this scenario, we cannot decide if A_1 is stronger than

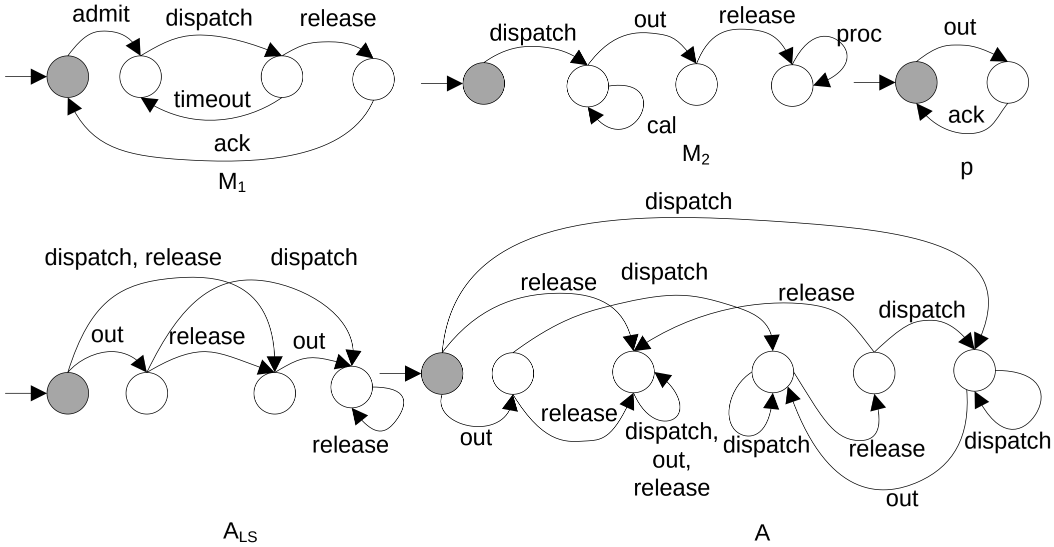

Figure 3. A counterexample proves that the assumptions generated in [2] are not strongest.

A_2 or vice versa. Another situation is that there exist two assumptions A_3 and A_4 which are the locally strongest assumptions in two specific subsets $\mathfrak{A}_3$ and $\mathfrak{A}_4$, but we also cannot decide if A_3 is stronger than A_4 or vice versa. Besides, we may even have a situation where there are two incomparable locally strongest assumptions in a single set of assumptions $\mathfrak{A}$. Furthermore, there exist many methods to improve the L^*–based assumption generation method to generate locally strongest assumptions. However, with the consideration of time complexity, we choose a method that can generate locally strongest assumptions in an acceptable time complexity.

We do this by creating a variant technique for answering membership queries of $Teacher$. This technique is then integrated into Algorithm 3 to generate locally strongest assumptions. We prove the correctness of the proposed method in Section 5.

4.1. A variant of the technique for answering membership queries

In Algorithm 1, $Learner$ updates the observation table during the learning process by asking $Teacher$ a membership query if a trace s belongs to the language of an assumption A that satisfies the assume-guarantee rules (i.e., $s \in L(A)$?).

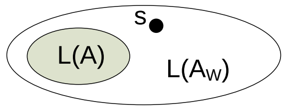

Figure 4. The relationship between $L(A)$ and $L(A_W)$.

Algorithm 2: An algorithm for answering membership queries

---

input : A trace $s = a_0a_1...a_n$
output : If $s \in L(A_W)$ then "?", otherwise $false$

```
begin
    if ⟨[s]⟩M1⟨p⟩ then
        return "?"
    else
        return false
    end
end
```

In order to answer this query, the algorithm in [2] bases on the language of the *weakest assumption* ($L(A_W)$) to consider if the given trace belongs to $L(A)$. If $s \in L(A_W)$, the algorithm returns $true$, otherwise, it returns

$false$. However, when the algorithm returns $true$, it has not known whether s really belongs to $L(A)$. This is because $\forall A : L(A) \subseteq L(A_W)$. The relationship between $L(A)$ and $L(A_W)$ is shown in Figure 4 [17]. For this reason, we use the same variant technique as proposed in [9–11, 17] for answering the membership queries described in Algorithm 2. In this variant algorithm when $Teacher$ receives a membership query for a trace $s = a_0 a_1 ... a_n \in \Sigma^*$, it first builds an LTS $[s]$. It then model checks $\langle [s] \rangle M_1 \langle p \rangle$. If $true$ is returned (i.e., $s \in L(A_W)$), $Teacher$ returns "?" (line 3). Otherwise, $Teacher$ returns $false$ (line 5). The "?" result is then used in $Learner$ to learn the locally strongest assumptions.

4.2. Generating the locally strongest assumptions

In order to employ the variant technique for answering membership queries proposed in Algorithm 2 to generate assumption while doing component-based software verification, we use the improved L^*–based algorithm shown in Algorithm 3. Given a CBS M that consists of two components M_1 and M_2 and a safety property p. The key idea of this algorithm bases on an observation that at each step of the learning process where the observation table is closed (OT_i), we can generate one candidate assumption (A_i). OT_i can have many "?" membership query results (for example, n results). When we try to take the combination of k "?" results out of n "?" results (where k is from n to 1) and consider all of these "?" results as $false$ (all of the corresponding traces do not belong to the language of the assumption to be generated) while we consider other "?" results as $true$, there are many cases that the corresponding observation table (OT_{kj}) is closed. Therefore, we can consider the corresponding candidate C_{kj} as a new candidate and ask an equivalence query for C_{kj}. In case both of A_i and C_{kj} satisfy the assume-guarantee rules in Definition 9, we always have $L(C_{kj}) \subseteq L(A_i)$. We will prove that the assumptions generated by Algorithm 3 are the locally strongest assumptions later in this paper. The details of the improved L^*–based algorithm are shown in Algorithm 3.

The algorithm starts by initializing S and E with the empty string (λ) (line 2). After that, the algorithm updates the observation (S, E, T) by using membership queries (line 4). The algorithm then tries to make (S, E, T) closed (from line 5 to line 8). We decide if (S, E, T) is closed with the consideration that all "?" results are $true$, this is the same as the assumption generation method proposed in [2]. When the observation table (S, E, T) closed, the algorithm updates those "?" results in rows of (S, E, T) which are corresponding to not final states to $true$ (line 9). This is because we want to reduce the number of "?" results in the observation table (S, E, T) so that the number of combinations in the next step will be smaller. The algorithm then checks the candidates that are corresponding to k-combinations of n "?" results which are considered as $false$ (line from 10 to 20). This step is performed in some smaller steps: For each k from n to 1 (line 10), the algorithm gets a k–combination of n "?" results (line 11); Turn all "?" results in the k–combination to $false$, the other "?" results will be turned to $true$ (line 12); If the corresponding observation table (S, E, T) is closed (line 13), the algorithm calculates a candidate C_{ikj} (line 14). After that, the algorithm asks $Teacher$ an equivalence query (line 15) and stores result in $result$. An equivalence query result contains two properties: $Key \in \{YES, NO, UNSAT\}$ (i.e., YES means the corresponding assumption satisfies the assume-guarantee rules in Definition 9; NO means the corresponding assumption does not satisfy assume-guarantee rules in Definition 9, however, at this point, we could not decide if the given system M does not satisfy p yet, we can use the corresponding counterexample cex to generate a new candidate assumption; $UNSAT$ means the given system M does not satisfy p and the counterexample is cex); the other property is an assumption when Key is YES or a counterexample cex when Key is NO or $UNSAT$. If $result.Key$ is YES, the algorithm stops and returns the assumption associated with $result$ (line 17).

Algorithm 3: Learning locally strongest assumptions algorithm

1 **begin**
2 Let $S = E = \{\lambda\}$
3 **while** $true$ **do**
4 Update T using membership queries
5 **while** (S, E, T) *is not closed* **do**
6 Add sa to S to make (S, E, T) closed where $s \in S$ and $a \in \Sigma$
7 Update T using membership queries
8 **end**
9 Update "?" results to $true$ in rows in (S, E, T) which are not corresponding to final states
10 **for** *each* k *from* n *to* 1 **do**
11 Get k–combination of n "?" results.
12 Turn all those "?" results to $false$, other "?" results are turned to $true$.
13 **if** *The corresponding observation table* (S, E, T) *is closed* **then**
14 Create a candidate assumption C_{ikj}.
15 $result \leftarrow$ Ask an equivalence query for C_{ikj}.
16 **if** $result.Key$ *is* YES **then**
17 **return** $result.Assumption$
18 **end**
19 **end**
20 **end**
21 Turn all "?" results in (S, E, T) to $true$
22 Construct candidate DFA D from (S, E, T)
23 Make the $conjecture$ A_i from D
24 $equiResult \leftarrow$ ask an equivalence query for A_i
25 **if** $equiResult.Key$ *is* YES **then**
26 **return** A_i
27 **else if** $equiResult.Key$ *is* $UNSAT$ **then**
28 **return** $UNSAT + cex$
29 **else**
/* $Teacher$ returns $NO + cex$ */
30 Add $e \in \Sigma^*$ that witnesses the counterexample to E
31 **end**
32 **end**
33 **end**

In this case, we have the locally strongest assumption generated. When the algorithm runs into line 21, it means that no stronger assumption can be found in this iteration of the learning progress, the algorithm turns all "?" results of (S, E, T) to $true$ and generates the corresponding candidate assumption A_i (lines from 21 to 23). The algorithm then asks an equivalence query for A_i (line 24). If the equivalence query result $equiResult.Key$ is YES, the algorithm stops and returns A_i as the needed assumption (line 26). If $equiResult.Key$ is $UNSAT$, the algorithm returns $UNSAT$ and the corresponding counterexample cex (line 28). This means that the given system M violates property p with the counterexample cex. Otherwise, the $equiResult.Key$ is NO and a counterexample cex. The algorithm will analyze the counterexample cex to find a suitable suffix e. This suffix e must be such that adding it to E will cause the next assumption candidate to reflect the difference and keep the set of suffixes E closed. The method to find e is not in the scope of this paper, please find more details in [8]. The algorithm then adds it to E in order to have a better candidate assumption in the next iteration (line 30). The algorithm then continues the learning process again from line 4 until it reaches a conclusive result.

5. Correctness

The correctness of our assumption generation method is proved through three steps: proving its soundness, completeness, and termination. The correctness of the proposed algorithm is proved based on the correctness of the assumption generation algorithm proposed in [2].

Lemma 1. *(Soundness). Let* $M_i = \langle Q_{M_i}, \Sigma_{M_i}, \delta_{M_i}, q_0^i \rangle$ *be LTSs, where* $i = 1, 2$ *and* p *be a safety property.*

1. *If Algorithm 3 reports "YES and an associated assumption A", then $M_1 || M_2 \models p$ and A is the satisfied assumption.*
2. *If Algorithm 3 reports "$UNSAT$ and a witness cex", then cex is the witness to $M_1 || M_2 \not\models p$.*

Proof. 1. When Algorithm 3 reports "YES", it has asked $Teacher$ an equivalence query at line 15 or line 24 and get the result "YES". When returning YES, $Teacher$ has verified that the candidate A actually satisfied the assume-guarantee rules in Definition 9 using the proposed algorithm in [2]. Therefore, $M_1||M_2 \models p$ and A is the required assumption thanks to the correctness of the learning algorithm proposed in [2].

2. On the other hand, when Algorithm 3 reports "$UNSAT$" and a counterexample cex, all of the candidate assumptions that have been asked to $Teacher$ in line 15 did not satisfy the assume-guarantee rules in Definition 9. The equivalence query in line 24 has the result $UNSAT$ and cex. When returning $UNSAT$ and cex, $Teacher$ has checked that M actually violates property p and cex is the witness. Therefore, thanks to the correctness of the learning algorithm proposed in [2], $M_1||M2 \not\models p$ and cex is the witness.

□

Lemma 2. *(Completeness). Let $M_i = \langle Q_{M_i}, \Sigma_{M_i}, \delta_{M_i}, q_0^i \rangle$ be LTSs, where $i = 1, 2$ and p be a safety property.*

1. *If $M_1||M_2 \models p$, then Algorithm 3 reports "YES" and the associated assumption A is the required assumption.*
2. *If $M_1||M_2 \not\models p$, then Algorithm 3 reports "$UNSAT$" and the associated counterexample cex is the witness to $M_1||M_2 \not\models p$.*

Proof. 1. Compare Algorithm 1 and Algorithm 3, we can see that Algorithm 3 is different from Algorithm 1 at lines from 9 to 21. On the other hand, these steps are finite steps asking $Teacher$ some more equivalence queries. Therefore, in the worst case, we cannot find out any satisfied assumption from these steps, the algorithm is equivalent to Algorithm 1. Therefore, if $M_1||M_2 \models p$, then in the worst case, Algorithm 3 returns YES and the corresponding assumption A thanks to the correctness of the learning algorithm proposed in [2].

2. The same as the above description, in the worst case, where no satisfied assumption can be found in Algorithm 3 from line 9 to line 21, Algorithm 3 is equivalent to Algorithm 1. Therefore, if $M_1||M_2 \not\models p$, then Algorithm 3 will return $UNSAT$ and the associated cex is the counterexample thanks to the correctness of the learning algorithm proposed in [2].

□

Lemma 3. *(Termination). Let $M_i = \langle Q_{M_i}, \Sigma_{M_i}, \delta_{M_i}, q_0^i \rangle$ be LTSs, where $i = 1, 2$ and p be a safety property. Algorithm 3 terminates in a finite number of learning steps.*

Proof. The termination of Algorithm 3 follows directly from the above Lemma 1 and 2. □

Lemma 4. *(Locally strongest assumption). Let $M_i = \langle Q_{M_i}, \Sigma_{M_i}, \delta_{M_i}, q_0^i \rangle$ be LTSs, where $i = 1, 2$ and p be a safety property. Let's assume that $M_1||M_2 \models p$ and Algorithm 3 does not return the assumption immediately after getting the first satisfied assumption (line 17). It continues running to find all possible assumptions until all of the question results are turned into "$true$" results in the corresponding observation table. Let $\mathfrak{A}$ be the set of those assumptions and A be the first generated assumption. A is the locally strongest assumption in $\mathfrak{A}$.*

Proof. The key idea of Algorithm 3 is shown in Figure 5. In this learning process, at the

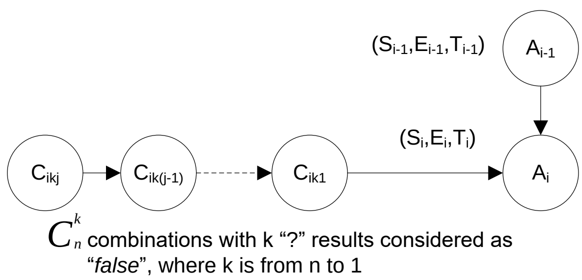

Figure 5. The key idea of the improved L^*–based assumption generation method.

iteration i^{th}, we have a closed table (S_i, E_i, T_i) and the corresponding candidate assumption A_i in which all "?" results are considered as $true$. This means all of the associated traces with those "?" results are considered *in* the language of the assumption to be generated. If we have n "?" results in (S_i, E_i, T_i), the algorithm will start this iteration by trying to get k–combinations of n "?" results and consider all "?" results in those k–combinations as $false$, where k is from n to 1. This means that the algorithm will try to consider those corresponding traces as *not* in the language of the assumption to be generated. By doing this, the algorithm has tried every possibility that a trace does not belong to the language of the assumption to be generated. This is because $k = n$ means no trace corresponding to "?" belongs to the language of the assumption to be generated. $k = n - 1$ means only one trace corresponding to "?" results belongs to the language of the assumption to be generated, and so on. On the other hand, Algorithm 3 stops learning right after reaching a conclusive result. Therefore, in the worst case, where all of "?" results are considered as $true$, Algorithm 3 is equivalent to Algorithm 1. In other cases where there is a candidate assumption $C_{ikj} \neq A_i$ that satisfies the assume-guarantee rules in Definition 9, obviously, we have $L(C_{ikj}) \subset L(A_i)$ because there are k "?" results in (S_i, E_i, T_i) are considered as $false$. This means k traces that belong to $L(A_i)$ but do not belong to $L(C_{ikj})$.

In case C_{ikj} exists, C_{ikj} is the *locally strongest assumption* because the algorithm has tried all possibilities that $n, n-1, ..., k+1$ "?" results do not belong to the language of the assumption to be generated but it has not been successful yet. This way, the algorithm has tries the strongest candidate assumption first, then weaker candidate assumptions later. On the other hand, with one value of k, we have many k–combinations of n "?" results which can be considered as $false$. Each of the k–combination is corresponding to one C_{ikj}, where $1 \leq j \leq C_n^k$. However, we cannot compare $L(C_{ikj})$ and $L(C_{ikt})$, where $1 \leq j, t \leq C_n^k$. Therefore, Algorithm 3 stops right after reaching the conclusive result and does not check all other C_{ikj} with the same value of k. As a result, the generated assumption must be the *locally strongest assumption* in the same iteration of the learning process.

We can remove line 21 from Algorithm 3. At that time, Algorithm 3 can generate stronger assumptions than those generated by Algorithm 1. However, it will not have the list of candidate assumptions of Algorithm 1 which plays a guideline role during the learning process. As a result, the algorithm will become much less efficient. □

Lemma 5. *(Complexity). Assume that Algorithm 1 takes m_{equi} equivalence queries and m_{mem} membership queries. Assume that at the iteration i^{th}, there are n_i "?" results. In the worst case where we have one candidate assumption for every k–combination of "?", it will takes $\Sigma_{k=1}^{n_i} C_{n_i}^k$ equivalence queries, but no more membership queries. Therefore, in total and in the worst case, Algorithm 3 takes $\Sigma_{i=1}^{m_{equi}} \Sigma_{k=1}^{n_i} C_{n_i}^k$ equivalence queries and m_{mem} membership queries. As a result, the complexity of the proposed algorithm at iteration i^{th} is $O(2^{n_i})$. For the target of reducing this complexity to a polynomial one, we have plan to another research that is based on the baseline candidate assumption A_i itself, not on its corresponding observation table (S_i, E_i, T_i) anymore.*

6. Experiment and discussion

This section presents our implemented tool for the improved L^*–based assumption generation method, Algorithm 3, and experimental results by applying the tool for some test systems. We also discuss the advantages and disadvantages of the proposed method.

6.1. Experiment

We have implemented Algorithm 3 in a tool called Locally Strongest Assumption Generation Tool (LSAG Tool[1]) in order to compare L^*–based assumption generation algorithm proposed in [2] with Algorithm 3. The tool is implemented using Microsoft Visual Studio 2017 Community. The test is carried out with some artificial test cases on a machine with the following system information: Processor: Intel(R) Core(TM) i5-3230M; CPU: @2.60GHz, 2601 Mhz, 2 Core(s), 4 Logical Processor(s); OS Name: Microsoft Windows 10 Enterprise. The experimental results are shown in Table 1. In this table, the sizes of M_1, M_2, and p are shown in columns $|M_1|$, $|M_2|$, and $|p|$, respectively. Column "Is stronger" shows if the assumptions generated by Algorithm 3 is stronger than those generated by L^*–based assumption generation method. "yes" means that the assumption generated by Algorithm 3 is stronger than the one generated by L^*–based assumption generation method while "no" indicates that the assumption generated by Algorithm 3 is actually the same as the one generated by L^*–based assumption generation method. When they are not the same (i.e., $A_{LS} \not\equiv A_{org}$), in order to check if the assumption generated by Algorithm 3 (A_{LS}) is stronger than the one generated by the L^*–based assumption generation method (A_{org}), we use a tool called LTSA [15, 16]. For this purpose, we describe A_{org} as a *property* and check if $A_{LS} \models A_{org}$. If the error state cannot be reached in LTSA tool (i.e., $L(A_{LS}) \subset L(A_{org})$), then the corresponding value in column "Is stronger" will be "yes". Otherwise, we have $A_{LS} \equiv A$ and the value in column "Is stronger" will be "no". Columns "AG Time(ms)" and "LSAG Time(ms)" show the time required to generate assumptions for L^*–based assumption generation method and Algorithm 3, respectively. Columns "M_{AG}", "EQ_{AG}" and "M_{LS}", "EQ_{LS}" show the corresponding number of membership queries and equivalence queries needed when generating assumptions using L^*–based assumption generation method and Algorithm 3. From the above experimental results, we have the following observations:

- For some systems (test case 1, 2, 3, and 4), Algorithm 3 can generate the same assumptions as the ones generated by L^*–based assumption generation method. For other systems (test case 5, 6, 7, and 8), Algorithm 3 can generate stronger assumptions than the ones generated by L^*–based assumption generation method.
- Algorithm 3 requires more time to generate assumptions than L^*–based assumption generation method.
- In test case 6 and 8, the number of membership queries needed to generate locally strongest assumption M_{LS} is less than the number of membership queries needed to generate original assumption. This is because, in this case, we can find a satisfied locally strongest assumption at a step prior to the step where the original assumption generation method can generate the satisfied assumption.

6.2. Discussion

In regards to the importances of the generated locally strongest assumptions when verifying CBS, there are several interesting points as follows:

- Modular verification for CBS is done by model checking the assume-guarantee rules with the generated assumption as one of its components. This is actually a problem of language containment of the languages of components of the system under checking and the assumption to be generated. For this reason, the computational cost of this checking is affected by the assumption language. Therefore, the stronger assumption we have, the more reduction we gain for the computational cost of the verification.

[1] http://www.tranhoangviet.name.vn/p/lsagtools.html

Table 1. Experimental results

No.	TestCase	$\|M1\|$	$\|M2\|$	$\|p\|$	Is stronger	M_{AG}	EQ_{AG}	AG Time (ms)	M_{LSAG}	EQ_{LSAG}	LSAG Time (ms)
1	TestCase1	3	3	2	no	17	2	51	17	11	106
2	TestCase2	43	5	3	no	161	5	1391	161	14	1601
3	TestCase3	3	5	3	no	254	6	147	254	51	1184
4	TestCase4	3	3	2	no	49	4	23	49	15	184
5	TestCase5	5	4	2	yes	38	3	19	38	17	57
6	TestCase6	4	4	2	yes	79	4	51	38	12	76
7	TestCase7	24	4	2	yes	112	4	732	101	79	1871
8	TestCase8	33	4	2	yes	145	4	2817	129	782	112932

- The key idea of this work is to consider that all possible combinations of traces which are not in the language of the assumption A to be generated. We do that by considering from the possibility that no trace belongs to $L(A)$ to the possibility that all traces belong to $L(A)$. Besides, the algorithm terminates as soon as it reaches a conclusive result. Because of this, the returned assumptions will be the local strongest ones.

- When a component is evolved after adapting some refinements in the context of software evolution, the whole evolved CBS needs to be rechecked. In this case, we can reduce the cost of rechecking the evolved system by using the locally strongest assumptions.

- Time complexity of Algorithm 3 is high in comparison to that of Algorithm 1 when generating the first assumption. However, this assumption can be used several times during software development life cycle. The more times we can reuse this assumption, the more computational cost we save for software verification. Further more, we are working on a method to reduce this time complexity of Algorithm 3.

- Locally strongest assumptions mean less complex behavior so this assumption is easier for human to understand. This is interesting for checking large–scale systems.

- The key point when implementing Algorithm 3 is how to keep the observation table *closed* and *consistent* so that the language of the corresponding assumption candidate can be consistent with the observation table. This can be done with a suitable algorithm to choose suffix e when adding new item to suffix list E of the observation table in line 30. This algorithm is not in the scope of this paper. Please refer to [8] for more details.

Despite the advantages mentioned above, the algorithm needs to try every possible combinations of “?” results to see if a trace can be in the language of $L(A)$, the complexity of the Algorithm 3 is clearly higher than the complexity of Algorithm 1.

The most complex step in Algorithm 3 is the step from line 10 to line 20 where the algorithm tries every possible k–combination of n “?” question results and consider them as $false$. Therefore, the complexity of Algorithm 3 depends on the number of “?” results in each steps of the learning process. For this reason, in Algorithm 3, we introduce an extra step in line 9 to reduce the number of “?” results that need to be processed. This is based on an observation that those traces that are associated to not final states in the DFA which is corresponding to the observation table do not have much value in the assumption to be generated. This is because those states will be removed when generating the candidate assumption from a closed observation table.

In the general case, not all of the cases that Algorithm 3 requires more time to generate assumption than the L^*–based assumption generation method. For example, if running Algorithm 1, it takes m_{equi} steps to reach the satisfied assumption. However, there may be a step i before m_{equi} where a combination of "?" results considered as $false$ results in a satisfied assumption. In this case, the time required to generate locally strongest assumption will be less than the time to generate assumption using L^*–based assumption generation method.

You may notice that Algorithm 3 bases on Algorithm 1 for making the observation table (S, E, T) closed, creating local candidate assumptions in the i^{th} iteration of the learning process. We can apply the method that considers "?" results as $false$ first when making the observation table (S, E, T) closed, if the corresponding candidate assumption does not satisfy the assume-guarantee rules in Definition 9, we can go one step back to consider one by one "?" results as $true$ until we find out the satisfied candidate assumption. However, this method of finding candidate assumption has a very much greater time complexity. We chose the method that bases on the L^*–based assumption generation method as a framework for providing baseline candidate assumptions during the learning process. We only generate local strongest candidate assumptions based on those baseline candidate assumptions. This method of learning can effectively generate locally strongest assumptions in an acceptable time complexity.

7. Related works

There are many researches related to improving the compositional verification for CBS. Consider only the most current works, we can refer to [2, 9–13, 17].

Tran et al. proposed a method to generate strongest assumption for verification of CBS [17]. However, this method has not considered assumptions that cannot be found by the algorithm. Therefore, the method can only find out locally strongest assumptions. Although the method presented by Tran et al. uses the same variant membership queries answering technique as proposed by Hung et al. [9–11], it has not considered using candidate assumptions generated by the method of Cobleigh et al. [2] as baseline candidates. As a result, the cost for verification is very high. Sharing the same idea of using the variant membership queries answering technique, we take the baseline candidate assumptions generated by the method of Cobleigh et al. into account when trying to find the satisfied assumptions. This results in an acceptable assumption generation time. In the meantime, the generated assumptions are also locally strongest assumptions.

The framework proposed in [2] by Cobleigh et al. can generate assumptions for compositional verification of CBS. However, because the algorithm is based on the language of the weakest assumption ($L(A_W)$), the generated assumptions are not strongest. By observing this, we focus on improving the method so that the algorithm can generate locally strongest assumptions which can reduce the computational cost when verifying large–scale CBS.

In [13], Gupta et al. proposed a method to compute an exact minimal automaton to act as an intermediate assertion in assume-guarantee reasoning, using a sampling approach and a Boolean satisfiability solver. This is an approach which is suitable to compute minimal separating assumptions for assume-guarantee reasoning for hardware verification. Our work focuses on generating the locally strongest assumptions when verifying CBS by improving the L^*–based assumption generation algorithm proposed in [2].

In a series of papers of [9–11], Hung et al. proposed a method for generating minimal assumptions, improving, and optimizing that method to generate those assumptions for compositional verification. However, the generated minimal assumptions in these works mean to have a minimal number of states. Our work shares the same observation that a trace s that belongs to $L(A_W)$ does not always belong to the generated assumption language $L(A)$. Besides, the satisfiability problem is actually the problem of language containment. Therefore, our work will effectively reduce the computational cost when verifying CBS.

Chaki and Strichman proposed three

optimizations in [12] to the L^*–based automated assume-guarantee reasoning algorithm for the compositional verification of concurrent systems. Among those three optimizations, the most important one is to develop a method for minimizing the alphabet used by the assumptions, which reduces the size of the assumptions and the number of queries required to construct them. However, the method does not generate the locally strongest assumptions as the proposed method in this paper.

8. Conclusion

We have presented a method to generate locally strongest assumptions for assume-guarantee verification of CBS. The key idea of this method is to develop a variant technique for answering membership queries from $Learner$ of $Teacher$. This technique is then integrated into an improved L^*–based algorithm for trying every possible combination that a trace belongs to the language of the assumption to be generated. Because the algorithm terminates as soon as it reaches the conclusive result, the generated assumptions are the locally strongest ones. These assumptions can effectively reduce the computational cost when doing verification for CBS, especially for large-scale and evolving ones.

Although the proposed method can generate locally strongest assumptions for compositional verification, it still has an exponential time complexity. On the other hand, there are many other methods that can generate other locally strongest assumptions. We are in progress of researching a method which can generate other locally strongest assumptions that are stronger than those generated by the proposed method in this paper but has a polynomial time complexity. Besides, we are also applying the proposed method for software in practice to prove its effectiveness. Moreover, we are investigating how to generalize the method for larger systems, i.e., systems contain more than two components. On the other hand, the current work is only for safety properties, we are going to extend our proposed method for checking other properties such as liveness properties and apply the proposed method for general systems, e.g., hardware systems, real-time systems, and evolving ones.

Acknowledgments

This work was funded by the Vietnam National Foundation for Science and Technology Development (NAFOSTED) under grant number 102.03-2015.25.

References

[1] D. Giannakopoulou, C. S. Păsăreanu, H. Barringer, Assumption generation for software component verification, in: Proceedings of the 17th IEEE International Conference on Automated Software Engineering, ASE '02, IEEE Computer Society, Washington, DC, USA, 2002, pp. 3–12.

[2] J. M. Cobleigh, D. Giannakopoulou, C. S. Păsăreanu, Learning assumptions for compositional verification, in: Proceedings of the 9th International Conference on Tools and Algorithms for the Construction and Analysis of Systems, TACAS'03, Springer-Verlag, Berlin, Heidelberg, 2003, pp. 331–346.

[3] E. Clarke, D. Long, K. McMillan, Compositional model checking, in: Proceedings of the Fourth Annual Symposium on Logic in Computer Science, IEEE Press, Piscataway, NJ, USA, 1989, pp. 353–362.

[4] O. Grumberg, D. E. Long, Model checking and modular verification, ACM Trans. Program. Lang. Syst. 16 (3) (1994) 843–871.

[5] A. Pnueli, In transition from global to modular temporal reasoning about programs, in: K. R. Apt (Ed.), Logics and Models of Concurrent Systems, Springer-Verlag New York, Inc., New York, NY, USA, 1985, Ch. In Transition from Global to Modular Temporal Reasoning About Programs, pp. 123–144.

[6] E. M. Clarke, Jr., O. Grumberg, D. A. Peled, Model Checking, MIT Press, Cambridge, MA, USA, 1999.

[7] D. Angluin, Learning regular sets from queries

and counterexamples, Inf. Comput. 75 (2) (1987) 87–106.

[8] R. L. Rivest, R. E. Schapire, Inference of finite automata using homing sequences, in: Proceedings of the Twenty-first Annual ACM Symposium on Theory of Computing, STOC ’89, ACM, New York, NY, USA, 1989, pp. 411–420.

[9] P. Ngoc Hung, T. Aoki, T. Katayama, Theoretical Aspects of Computing - ICTAC 2009: 6th International Colloquium, Kuala Lumpur, Malaysia, August 16-20, 2009. Proceedings, Springer Berlin Heidelberg, Berlin, Heidelberg, 2009, Ch. A Minimized Assumption Generation Method for Component-Based Software Verification, pp. 277–291.

[10] P. N. Hung, V.-H. Nguyen, T. Aoki, T. Katayama, An improvement of minimized assumption generation method for component-based software verification, in: Computing and Communication Technologies, Research, Innovation, and Vision for the Future (RIVF), 2012 IEEE RIVF International Conference on, 2012, pp. 1–6.

[11] P. N. Hung, V. H. Nguyen, T. Aoki, T. Katayama, On optimization of minimized assumption generation method for component-based software verification, IEICE Transactions 95-A (9) (2012) 1451–1460.

[12] S. Chaki, O. Strichman, Tools and Algorithms for the Construction and Analysis of Systems: 13th International Conference, TACAS 2007, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2007 Braga, Portugal, March 24 - April 1, 2007. Proceedings, Springer Berlin Heidelberg, Berlin, Heidelberg, 2007, Ch. Optimized L*-Based Assume-Guarantee Reasoning, pp. 276–291.

[13] A. Gupta, K. L. Mcmillan, Z. Fu, Automated assumption generation for compositional verification, Form. Methods Syst. Des. 32 (3) (2008) 285–301.

[14] A. Nerode, Linear automaton transformations, Proceedings of the American Mathematical Society 9 (4) (1958) 541–544.

[15] J. Magee, J. Kramer, Labelled transition system analyser v3.0, https://www.doc.ic.ac.uk/ltsa/.

[16] J. Magee, J. Kramer, D. Giannakopoulou, Behaviour Analysis of Software Architectures, Springer US, Boston, MA, 1999, pp. 35–49.

[17] H.-V. Tran, C. L. Le, P. N. Hung, A strongest assumption generation method for component-based software verification, in: Computing and Communication Technologies, Research, Innovation, and Vision for the Future, IEEE–RIVF International Conference, 2016.