Assume-Guarantee Reasoning with Local Specifications

Alessio Lomuscio[1], Ben Strulo[2], Nigel Walker[2], and Peng Wu[3]

[1] Department of Computing, Imperial College London, UK
a.lomuscio@imperial.ac.uk
[2] BT Innovate, Adastral Park, UK
{ben.strulo,nigel.g.walker}@bt.com
[3] State Key Laboratory of Computer Science, Institute of Software, Chinese Academy of Sciences, China
wp@ios.ac.cn

Abstract. We investigate assume-guarantee reasoning for global specifications consisting of conjunctions of local specifications. We present a sound and complete assume-guarantee rule that permits reasoning about individual modules for local specifications and draws conclusions on global specifications. We illustrate our approach with an example from the field of network congestion control, where different agents are responsible for controlling packet flow across a shared infrastructure. In this context, we derive an assume-guarantee rule for system stability, and show that this rule is valuable to reason about any number of agents, any initial flow configuration, and any topology of bounded degree.

1 Introduction

Assume-Guarantee reasoning [21, 12, 5] is one of the key techniques to alleviate state explosion in model checking. In a system composed of a number of reactive modules each module can be regarded as interacting with an abstract environment representing the rest of the system. Properties are then verified with the aid of assumptions characterising the environment of each module. General assume-guarantee rules have been proposed for safety and liveness properties over the last decade [11, 4, 7, 8]. However, large assumptions can still cause scalability issues. The motivation of this paper is to investigate possible ways to reduce the size of the assumptions to be identified and to reuse them for compositional verification.

Our starting point is the observation that a module in a system typically reacts directly with relatively few modules in its environment. However, under general assume-guarantee rules, the assumptions generated from a system property do not exploit this neighbourhood dependency. Consequently, assumptions for a module may contain redundant information about parts of the system that the module does not directly interact with. Moreover, any new modules added to the system can contribute with further redundancy in the assumptions.

In this paper we show that, for a system property that can be represented as the conjunction of local specifications on individual modules, these scalability issues can be resolved by generating assumptions from local specifications. Our main contribution is a new presentation of the assume-guarantee rules to permit reasoning about individual modules for local specifications, yet drawing conclusions on properties of the system as a whole.

Firstly, we present a simple assume-guarantee rule $\mathbf{R_1}$ that we prove to be sound for local specifications. Through a counterexample, we show that this simple rule is not complete because it exploits only the direct dependency between modules.

We then extend rule $\mathbf{R_1}$ towards completeness. This leads to a bounded assume-guarantee rule $\mathbf{R}^\pi$ that we prove to be sound and complete for local specifications. It explores the neighbourhood around each module up to the depth π of the dependency closure of the system. We use this rule to propose a bounded assume-guarantee reasoning approach, in which the dependency between modules is exploited incrementally.

We evaluate the approach through a case study of an optimisation-based congestion control system proposed by Kelly and Voice [15]. The optimisation approach allows a distributed solution for network congestion control. The fact that a congestion control system is stable means that each source in the system reaches an equilibrium flow configuration on the routes available to the source. We analyse the stability of the system by reasoning about local stability of its individual sources. The case study shows that an instantiation of rule $\mathbf{R}^\pi$ for system stability can be applied for reasoning about any number of sources, any initial flow configuration, and any topology of bounded degree. To the best of our knowledge, previous work on model checking of networked systems focused on verifying network protocols under given topologies. The assume-guarantee framework developed in this paper supports verification of network-wide objectives *irrespective of the underlying network topologies*.

Related Work. The history of compositional verification of concurrent systems dates back to late 70s and 80s with the pioneering works by Francez and Pnueli [9], Jones [14] and Misra and Chandy [21]. Since then, considerable effort was devoted to studying the soundness of circular assume-guarantee reasoning. Maier [20] showed that compositional circular assume-guarantee rules cannot be both sound and complete. Kupferman and Vardi [16] presented an automata-theoretic approach to model checking assume-guarantee assertions.

More recently, Giannakopoulou, Păsăreanu et al. [11, 4, 7, 10, 24] proposed sound and complete non-circular assume-guarantee rules for safety properties, with support of learning based assumption generation. Nam, Alur et al. [22, 23] proposed a symbolic approach to learning-based assume-guarantee reasoning. Farzan, Chen et al. [8] extended the assume-guarantee rules to liveness properties, based on the fact that ω-regular languages preserve the essential closure properties of regular languages.

The idea of reasoning about local specifications has appeared in early works on compositional verification [1, 13], where sound circular assume-guarantee rules

were proposed for safety properties. This idea is further expanded in this paper to reduce the size of assumptions, and hence to improve the scalability of assume-guarantee reasoning. Moreover, the bounded rule here presented is shown to be sound and complete and applies to liveness properties. Our approach can also be implemented using symbolic representation, and integrated with learning algorithms for automated assumption generation. Additionally, learning-based methodologies can also benefit from our approach by exploiting assumptions over local alphabets, instead of the global alphabet.

The rest of this paper is organised as follows. The simple rule $\mathbf{R_1}$ and the bounded rule $\mathbf{R}^\pi$ are presented in Section 2 and Section 3, respectively. Section 4 illustrates the case study of network congestion control, with the experimental results reported and discussed in Section 5. The conclusions of this work are summarised in Section 6.

2 Assume-Guarantee Reasoning

In this section we first introduce the notion of module in concurrent systems. Then, we present a simple assume-guarantee rule $\mathbf{R_1}$ that permits reasoning about individual modules for local specifications.

Modules. Technically we adopt the basic notion of reactive module [2] to represent concurrent systems that consist of multiple interacting agents. A module is associated with two classes of variables: *state variables* and *input variables*. The former is controlled by the module and thus defines the module's state; the latter is controlled by others that the module reacts directly with.

We assume a domain D of all variables. For a set X of variables, let D^X be the set of all valuation functions on X. For valuation $\rho\colon X \to D$ and $Y \subseteq X$, $\rho\restriction_Y\colon Y \to D$ is the restriction of ρ to Y, i.e., $(\rho\restriction_Y)(x) = \rho(x)$ for any $x \in Y$.

For valuations $\rho_1 : X_1 \to D$ and $\rho_2 : X_2 \to D$, ρ_1 and ρ_2 are *compatible*, denoted $\rho_1 \sim \rho_2$, if $\rho_1(x) = \rho_2(x)$ for any $x \in X_1 \cap X_2$. For compatible valuations ρ_1 and ρ_2, $\rho_1 \cup \rho_2$ is the extension of ρ_1 and ρ_2 to $X_1 \cup X_2$, i.e., $(\rho_1 \cup \rho_2)(x) = \rho_1(x)$ for $x \in X_1 \backslash X_2$, $(\rho_1 \cup \rho_2)(x) = \rho_2(x)$ for $x \in X_2 \backslash X_1$ and $(\rho_1 \cup \rho_2)(x) = \rho_1(x) = \rho_2(x)$ for $x \in X_1 \cap X_2$.

Definition 1 (Module). *A* module *is a tuple $M = (X, I, Q, T, \lambda, q_0)$, where*

- *X is a finite set of state variables controlled by M;*
- *I is a finite set of input variables that module M depends on with $X \cap I = \emptyset$;*
- *Q is a finite set of states;*
- *$\lambda : Q \to D^X$ labels each state $q \in Q$ with a valuation $\lambda(q) : X \to D$;*
- *$T \subseteq Q \times D^I \times Q$ is a transition relation; each transition $(q, \alpha, q') \in T$, denoted $q \xrightarrow{\alpha}_T q'$, means that the state of M evolves from q to q' under input $\alpha : I \to D$;*
- *$q_0 \in Q$ is the initial state.*

An infinite trace σ of module M is an infinite sequence $q_0\alpha_0 q_1\alpha_1 \ldots$ such that $q_i \xrightarrow{\alpha_i}_T q_{i+1}$ for any $i \geq 0$. Let $inf(\sigma)$ be the set of all the states that are visited infinitely often in σ.

D^X is referred to as the *local* alphabet of module M, where each $\rho \in D^X$ is a valuation on X. An infinite word $w = \rho_0\rho_1\ldots$ on the local alphabet D^X is *derived* by M if there exists an infinite trace $q_0\alpha_0q_1\alpha_1\ldots$ of module M such that $\rho_i = \lambda(q_i)$ for any $i \geq 0$.

D^I is referred to as the *input* alphabet of module M, where each $\alpha \in D^I$ is a valuation on I. An infinite word $\theta = \alpha_0\alpha_1\ldots$ on the input alphabet D^I is *admitted* by M if there exists an infinite trace $q_0\alpha_0q_1\alpha_1\ldots$ such that $q_i \in Q$ for any $i \geq 0$. Let $\mathcal{I}(M)$ be the set of the input words admitted by M. We say that module M is *closed* if $I = \emptyset$.

We define the composition operator for modules. We choose a notion of composition that explicitly supports asynchrony, because in distributed environments asynchrony typically arises externally from network communication or scheduling.

Definition 2 (Composition). *For modules $M_1 = (X_1, I_1, Q_1, T_1, \lambda_1, q_{0_1})$ and $M_2 = (X_2, I_2, Q_2, T_2, \lambda_2, q_{0_2})$, the composition of M_1 with M_2 is a composite module $M_1|M_2 = (X_1 \cup X_2, (I_1 \cup I_2)\backslash(X_1 \cup X_2), Q, T, \lambda, q_0)$, where*

- *$Q \subseteq Q_1 \times Q_2$ is the maximal set such that $\lambda_1(q_1) \sim \lambda_2(q_2)$ for each state $(q_1, q_2) \in Q$;*
- *$\lambda : Q \to D^{X_1 \cup X_2}$ labels each state $(q_1, q_2) \in Q$ with the valuation $\lambda_1(q_1) \cup \lambda_2(q_2)$;*
- *T is the minimal transition relation derived by the following composition rules:*

$$\textsc{asyn}_L\ \frac{q_1 \xrightarrow{\alpha_1}_{T_1} q_1' \quad q_2 \xrightarrow{\alpha_2}_{T_2} q_2'}{(q_1, q_2) \xrightarrow{\alpha}_T (q_1', q_2)} \qquad \textsc{asyn}_R\ \frac{q_1 \xrightarrow{\alpha_1}_{T_1} q_1' \quad q_2 \xrightarrow{\alpha_2}_{T_2} q_2'}{(q_1, q_2) \xrightarrow{\alpha}_T (q_1, q_2')}$$

$$\textsc{syn}\ \frac{q_1 \xrightarrow{\alpha_1}_{T_1} q_1' \quad q_2 \xrightarrow{\alpha_2}_{T_2} q_2'}{(q_1, q_2) \xrightarrow{\alpha}_T (q_1', q_2')}$$

where $\lambda(q_1) \sim \lambda(q_2)$, $\lambda(q_1') \sim \lambda(q_2)$, $\lambda(q_2') \sim \lambda(q_1)$, $\lambda(q_1') \sim \lambda(q_2')$, $\lambda(q_2) \sim \alpha_1$, $\lambda(q_1) \sim \alpha_2$, $\alpha_1 \sim \alpha_2$, and $\alpha = (\alpha_1 \cup \alpha_2)\restriction_I$.

- *$q_0 = (q_{0_1}, q_{0_2}) \in Q$.*

In rule $\textsc{asyn}_L$ (respectively, $\textsc{asyn}_R$) only M_1 (respectively, M_2) evolves; while in rule SYN both M_1 and M_2 evolve simultaneously. In the presence of several modules, the composition rules above can allow only one, some, or all modules evolve simultaneously. The notion of module and composition can be implemented by existing reactive module languages [3, 6].

For an infinite word $w = \rho_0\rho_1\ldots$ derived by $M_1|M_2$, we define the notion of stuttering projection to hide asynchronous transitions that do not affect the variables in X_1 or X_2. For $Z \subseteq X_1 \cup X_2$, a *stuttering projection* of w on Z, denoted $w|_Z$, is an infinite word $\rho_0'\rho_1'\ldots$, where there exists $0 = j_0 < j_1 < \cdots$ such that $\rho_i' = \rho_{j_i}\restriction_Z = \rho_{j_i+1}\restriction_Z = \cdots = \rho_{j_{i+1}-1}\restriction_Z$ for any $i \geq 0$. Specifically, the *restriction* of w on Z, denoted $w\restriction_Z$, is the infinite word $\rho_0'\rho_1'\ldots$, where $\rho_i' = \rho_i\restriction_Z$ for any $i \geq 0$.

Thus, a *closed* concurrent system with a finite set X of state variables can be represented as the composition of n modules $M_i = (X_i, I_i, Q_{M_i}, T_{M_i}, \lambda_{M_i}, q_{0_{M_i}})$,

where $X_i \cap X_j = \emptyset$ for any $1 \leq i \neq j \leq n$, $\bigcup_{i=1}^{n} X_i = X$ and $\bigcup_{i=1}^{n} I_i \subseteq X$. D^X is then referred to as the *global* alphabet of the system $M_1|\cdots|M_n$.

Assumptions can then be defined as extended modules with accepting states. In this paper we focus on liveness properties; therefore, we adopt the formalism of Büchi automaton for the definition of assumptions. However, the assume-guarantee rules presented later also apply to safety properties (for which assumptions are then defined as finite automata [4]). We do not discuss safety properties further.

Definition 3 (Assumption). *An assumption is a tuple $A = (X, I, Q, T, \lambda, q_0, F)$, where X, I, Q, T, λ, q_0 are as in Definition 1, and $F \subseteq Q$ is a finite set of accepting states.*

The terminology defined for modules also applies to assumptions. So, an infinite word $\rho_0\rho_1\ldots$ on alphabet D^X is *accepted* by A if there exists an infinite trace $\sigma = q_0\alpha_0q_1\alpha_1\ldots$, referred to as an *accepting trace*, such that $inf(\sigma)\cap F \neq \emptyset$ and $\rho_i = \lambda(q_i)$ for any $i \geq 0$. The *language* $\mathcal{L}(A)$ accepted by A consists of all the infinite words accepted by A. Let coA be the complement of assumption A accepting the complement language $\Omega_X \backslash \mathcal{L}(A)$, where Ω_X is the set of infinite words on alphabet D^X.

The notion of composition can be extended to assumptions. For module $M = (X_1, I_1, Q_1, T_1, \lambda_1, q_{0_1})$ and assumption $A = (X_2, I_2, Q_2, T_2, \lambda_2, q_{0_2}, F_A)$, the composition of M with A is an extended module $M|A = (X, I, Q, T, \lambda, q_0, F)$, where X, I, Q, T, λ, q_0 are as in Definition 2 and $F = \{(q_1, q_2) \in Q \mid q_2 \in F_A\}$. For extended modules $coA_i = (X_i, I_i, Q_i, T_i, \lambda_i, q_{0_i}, F_i)$ $(i = 1, 2)$, the composition of coA_1 with coA_2 is an extended module $coA_1|coA_2 = (X, I, Q, T, \lambda, q_0, F)$, where X, I, Q, T, λ, q_0 are as in Definition 2 and $F = \{(q_1, q_2) \in Q \mid q_1 \in F_1, q_2 \in F_2\}$.

The following definition formalises the notion of guarantee in the context above.

Definition 4 (Guarantee). *For k modules $M_i = (X_i, I_i, Q_i, T_i, \lambda_i, q_{0_i}), 1 \leq k \leq n$, and an assumption $A = (X_A, I_A, Q_A, T_A, \lambda_A, q_{0_A}, F_A)$ such that*

- *$X_i \cap X_j = \emptyset$ for any $1 \leq i \neq j \leq k$;*
- *$M_{i_1}, \ldots, M_{i_{k'}}$ $(1 \leq i_1, \ldots, i_{k'} \leq k)$ are all the $k' \leq k$ modules such that $X_A \cap X_{M_{i_j}} \neq \emptyset$ for $1 \leq j \leq k'$;*
- *$X_A \subseteq \bigcup_{j=1}^{k'} X_{M_{i_j}}$,*

then $M_1|\cdots|M_k$ guarantees *A, denoted $M_1|\cdots|M_k \vDash A$, if for any infinite word w derived by $M_1|\cdots|M_k$ and any stuttering projection w' of w on $\bigcup_{j=1}^{k'} X_{M_{i_j}}$ that can be derived by $M_{i_1}|\cdots|M_{i_{k'}}$, $w' \restriction_{X_A}$ is accepted by A.*

Note that, if $k' = k$, i.e., $X_A \cap X_i \neq \emptyset$ for any $1 \leq i \leq k$, $M_1|\cdots|M_k \vDash A$ simply means that for any infinite word w derived by $M_1|\cdots|M_k$, we have that $w \restriction_{X_A}$ is accepted by A.

Simple Assume-Guarantee Rule. Consider the system $M_1|\cdots|M_n$ and a global specification ψ on X that can be represented as the conjunction of local specifications φ_i on $X_i \cup I_i$ such that $\psi \Leftrightarrow \bigwedge_{i=1}^{n} \varphi_i$. The general assume-guarantee approach [11, 4, 7, 8] either generates assumptions for each module from the global specification and then checks whether these assumptions may collectively violate it (as shown by rule SYM below); or generates an assumption for some module (e.g., M_1) from the global specification and then checks whether the assumption can be guaranteed by the rest of modules (as shown by rule ASYM below).

$$\text{SYM}\ \frac{\forall 1 \le i \le n,\ M_i|A_i \models \psi \qquad \mathcal{L}(coA_1|\cdots|coA_n) = \emptyset}{M_1|\cdots|M_n \models \psi} \qquad\qquad \text{ASYM}\ \frac{M_1|A_1 \models \psi \qquad M_2|\cdots|M_n \models A_1}{M_1|\cdots|M_n \models \psi}$$

Thus, it is a common practice to generate assumptions from global specifications. However, in concurrent systems, each module typically control its state variables under inputs from *only a small* proportion of other modules. Therefore, in standard methodologies:

- each assumption A_i for module M_i may contain irrelevant valuations of state variables that module M_i does not depend on. This makes the size of assumption A_i larger than necessary.
- whenever the system is extended, each assumption A_i may have to be modified to incorporate the state variables of the additional modules. Hence, assumptions already generated for the existing modules cannot be reused for verifying the extended system.

We propose to avoid these issues by assigning each module M_i with the corresponding local specification φ_i. Inspired by rules SYM and ASYM, we present below rules $\mathbf{R_0}$ and $\mathbf{R_1}$, respectively. Recall that $\psi \Leftrightarrow \bigwedge_{i=1}^{n} \varphi_i$.

$$\mathbf{R_0}\ \frac{\forall 1 \le i \le n,\ M_i|A_i \models \varphi_i \qquad \mathcal{L}(coA_1|\cdots|coA_n) = \emptyset}{M_1|\cdots|M_n \models \bigwedge_{i=1}^{n} \varphi_i} \qquad\qquad \mathbf{R_1}\ \frac{\forall 1 \le i \le n,\ M_i|A_i \models \varphi_i \qquad M_{i_1}|\cdots|M_{i_{k_i}} \models A_i}{M_1|\cdots|M_n \models \bigwedge_{i=1}^{n} \varphi_i}$$

In rules SYM and ASYM assumptions A_i are all supposed to be generated from the global specification ψ (on X); while in rules $\mathbf{R_0}$ and $\mathbf{R_1}$ each assumption A_i is to be generated from the corresponding local specification φ_i (on $X_i \cup I_i$). In this way the size of assumption A_i can be reduced because only variables in $X_i \cup I_i$ (which is a subset of X) have to be concerned with assumption A_i.

Unsound Rule $\mathbf{R_0}$. As a side effect in rule $\mathbf{R_0}$, assumption A_i may admit more interactions with module M_i than can be admitted by the assumptions generated from the global specification ψ. This is because the variables in $X \backslash (X_i \cup I_i)$ are not constrained by the local specification φ_i. Therefore, the tentative rule $\mathbf{R_0}$ above does not preserve soundness, though its completeness is not affected by the weaker assumptions. We refer to our technical report [18] for a counterexample where rule $\mathbf{R_0}$ fails.

Sound Rule $\mathbf{R_1}$. Modules $M_{i_1}, \ldots, M_{i_{k_i}}$ $(k_i \geq 1)$ in rule $\mathbf{R_1}$ are all the k_i neighbours of module M_i that control its input variables in I_i (i.e., $I_i \subseteq \bigcup_{j=1}^{k_i} X_{i_j}$ and $I_i \cap X_{i_j} \neq \emptyset$ for any $1 \leq j \leq k_i$). Theorem 1 shows the soundness of rule $\mathbf{R_1}$ for local specifications.

Theorem 1 (Soundness). *If for any module M_i $(1 \leq i \leq n)$ there exists an assumption A_i such that $M_i|A_i \vDash \varphi_i$ and $M_{i_1}|\cdots|M_{i_{k_i}} \vDash A_i$, then $M_1|\cdots|M_n \vDash \bigwedge_{i=1}^{n} \varphi_i$.*

Proof. By contradiction. Consider an infinite word $w = \rho_0\rho_1\ldots$ on the global alphabet (i.e., each ρ_i is a valuation on X) that makes the conclusion fail on some φ_j $(1 \leq j \leq n)$. Then, since the state variables in X_j are exclusively controlled by M_j, any stuttering projection $w|_{X_j \cup I_j}$ would not be accepted by $M_j|A_j$ and hence any stuttering projection $w|_{I_j}$ would not be accepted by A_j.

However, the variables in X_{j_l} $(1 \leq l \leq k_j)$ are exclusively controlled by M_{j_l}. By the composition rules in Definition 2, there exists a stuttering projection of w on $\bigcup_{l=1}^{k_j} X_{j_l}$, denoted w', that is derived by $M_{j_1}|\cdots|M_{j_{k_j}}$. Since $I_j \subseteq \bigcup_{l=1}^{k_j} X_{j_l}$ and $M_{j_1}|\cdots|M_{j_{k_j}} \vDash A_j$, we have that $w' \restriction_{I_j}$ is accepted by A_j. This is a contradiction because $w' \restriction_{I_j}$ is also a stuttering projection of w on I_j.

Unfortunately, rule $\mathbf{R_1}$ is not complete. In fact, for each module M_i, its neighbour modules are isolated from the system when being examined against assumption A_i. This ignores the impact of the rest of modules on its neighbour modules. For example, consider a system consisting of the following four modules M_i $(1 \leq i \leq 4)$:

M_i	X_i	I_i	Transition Function
M_1	$\{x_1\}$	$\{x_2, x_3\}$	$x_1' = x_2 - x_3$
M_2	$\{x_2\}$	$\{x_4\}$	$x_2' = x_2 - x_4$
M_3	$\{x_3\}$	$\{x_4\}$	$x_3' = x_3 + x_4$
M_4	$\{x_4\}$	$\{x_2, x_3\}$	$x_4' = \begin{cases} 1 & x_2 > x_3 \text{ and } x_4 > 0 \\ -1 & x_2 < x_3 \text{ and } x_4 < 0 \\ 0 & \text{otherwise} \end{cases}$

Let x' be the next value of variable x. Then, for each module M_i, the CTL formula $AFAG\ (\bigwedge_{x \in X_i \cup I_i} (x' = x))$ specifies that the values of the variables in $X_i \cup I_i$ will always eventually remain unchanged for ever.

With an initial state $(x_1, x_2, x_3, x_4) = (u{-}v, u, v, 1)$ for any $u > v \geq 0$, it can be seen that $M_1|M_2|M_3|M_4 \vDash \bigwedge_{i=1}^{4} AFAG\ (\bigwedge_{x \in X_i \cup I_i} (x' = x))$. This is because x_2 and x_3 evolve by converging in a step of size x_4, until x_2 and x_3 meet or just cross over each other. Then, the system $M_1|M_2|M_3|M_4$ reaches a stable state where $x_4 = 0$.

However, by $M_2|M_3$ itself, x_2 and x_3 may diverge from each other. Hence, such divergent sequence of inputs (x_2, x_3) cannot lead M_1 to stabilising x_1,

and so cannot be accepted by any assumption A_1 that satisfies the premise $M_1|A_1 \vDash AFAG \bigwedge_{x \in X_1 \cup I_1} (x' = x)$.

3 Bounded Assume-Guarantee Reasoning

In this section we modify rule $\mathbf{R_1}$ to achieve completeness by exploiting the neighbourhood dependency between modules. This results in a "bounded" rule $\mathbf{R}^{\pi}$, which defines a bounded assume-guarantee reasoning approach.

Let $\mathcal{D} = \{(M_1, M_2) \mid X_2 \cap I_1 \neq \emptyset\}$ be the direct dependency relation between the modules of the system $M_1|\cdots|M_n$. $(M_1, M_2) \in \mathcal{D}$ means that module M_1 depends on the inputs from (or reacts directly with) module M_2. Then, the k-dependency relation $\mathcal{D}^k$ is defined recursively as follows: $\mathcal{D}^1 = \mathcal{D}$ and $\mathcal{D}^k = \mathcal{D}^{k-1} \cup (\mathcal{D}^{k-1} \circ \mathcal{D})$ for $k > 1$, where $\mathcal{D}^{k-1} \circ \mathcal{D}$ is the composition of $\mathcal{D}^{k-1}$ with $\mathcal{D}$.

For module M_i let $\mathcal{N}_i^k$ be the set of all the modules M except M_i such that $(M_i, M) \in \mathcal{D}^k$, and $\mathcal{C}_i^k$ be the composition of all the modules in $\mathcal{N}_i^k$. Then, rule $\mathbf{R_1}$ can be extended further to rule $\mathbf{R_k}$ as follows:

$$\mathbf{R_k} \frac{\forall 1 \leq i \leq n, \begin{array}{c} M_i|A_i \vDash \varphi_i \\ \mathcal{C}_i^k \vDash A_i \end{array}}{M_1|\cdots|M_n \vDash \bigwedge_{i=1}^{n} \varphi_i}$$

Informally, for each module M_i, rule $\mathbf{R_1}$ involves reasoning about its neighbour modules only; while rule $\mathbf{R_k}$ checks *all* the modules within the range of k hops around module M_i. Similarly, it can be proved that rule $\mathbf{R_k}$ is sound for any $k \geq 1$.

Theorem 2 (Soundness). *Given $k \geq 1$, if for any module M_i $(1 \leq i \leq n)$, there exists an assumption A_i such that $M_i|A_i \vDash \varphi_i$ and $\mathcal{C}_i^k \vDash A_i$, then $M_1|\cdots|M_n \vDash \bigwedge_{i=1}^{n} \varphi_i$.*

Proof. By contradiction. The proof is similar to that of Theorem 1.

If the modules within k hops around module M_i can together guarantee assumption A_i, then such guarantee is preserved by the modules within $k+1$ hops. This is because assumption A_i can already be guaranteed regardless of the interactions with the additional modules. Based on this observation, Theorem 3 relates rule $\mathbf{R_k}$ with rule $\mathbf{R_{k+1}}$.

Theorem 3. *Let A_i be an assumption for module M_i. Then, $\mathcal{C}_i^k \vDash A_i$ implies $\mathcal{C}_i^{k+1} \vDash A_i$.*

Proof. By the definition of $\mathcal{D}^k$, we have $\mathcal{N}_i^k \subseteq \mathcal{N}_i^{k+1}$. So, $I_i \subseteq \bigcup_{M_j \in \mathcal{N}_i^k} X_j \subseteq \bigcup_{M_j \in \mathcal{N}_i^{k+1}} X_j$. For any infinite word w derived by $\mathcal{C}_i^{k+1}$, there exists a stuttering projection of w on $\bigcup_{M_j \in \mathcal{N}_i^k} X_j$, denoted w', that can be derived by $\mathcal{C}_i^k$. Since $\mathcal{C}_i^k \vDash A_i$, $w' \restriction_{I_i}$ would be accepted by A_i for any such w'.

Since the system consists of a finite number of state variables, there exists a transitive dependency closure $\mathcal{D}^\pi$ ($\pi \geq 1$) such that $\mathcal{D}^\pi = \mathcal{D}^{\pi+1}$. Theorem 4 shows that rule $\mathbf{R}_\pi$ is complete for local specifications.

Theorem 4 (Completeness). *Suppose $\mathcal{D}^\pi$ is the transition dependency closure of the system $M_1|\cdots|M_n$. If $M_1|\cdots|M_n \vDash \bigwedge_{i=1}^{n} \varphi_i$, then for each module M_i, there exists an assumption A_i such that $M_i|A_i \vDash \varphi_i$ and $\mathcal{C}_i^\pi \vDash A_i$.*

Proof. By construction. For each module M_i, $\mathcal{C}_i^\pi$ could be extended as such assumption A_i by appointing all states in $\mathcal{C}_i^\pi$ as accepting states. This is because for any $1 \leq j \leq k$, $\bigwedge_{i=1}^{n} \varphi_i$ implies φ_j, and the variables in X_j are exclusively controlled by M_j that is independent of modules not in $M_j|\mathcal{C}_j^\pi$.

As a corollary of theorems 2, 3 and 4, rule R_π could be reformulated as rule R^π below, which is also sound and complete for local specifications.

$$\mathbf{R}^\pi \frac{\begin{array}{c} \forall 1 \leq i \leq n, \quad M_i|A_i \ \vDash \varphi_i \\ \exists 1 \leq d_i \leq \pi, \ \mathcal{C}_i^{d_i} \vDash A_i \end{array}}{M_1|\cdots|M_n \ \vDash \bigwedge_{i=1}^{n} \varphi_i}$$

Rule $\mathbf{R}^\pi$ can be applied incrementally for compositional verification of concurrent systems. For the sake of generality and reusability, we opt for the weakest assumption WA_i [4, 22] that admits as many as possible sequences of inputs to module M_i without violating the local specification φ_i. For module M_i, the weakest assumption WA_i is an assumption such that

- $\mathcal{L}(WA_i) \subseteq \mathcal{I}(M_i)$ and $M_i|WA_i \vDash \varphi_i$;
- $\mathcal{L}(A_i) \subseteq \mathcal{L}(WA_i)$ for any assumption A_i such that $\mathcal{L}(A_i) \subseteq \mathcal{I}(M_i)$ and $M_i|A_i \vDash \varphi_i$.

Then, the verification task for checking whether the system $M_1|\cdots|M_n$ satisfies the global specification ψ ($\Leftrightarrow \bigwedge_{i=1}^{n} \varphi_i$) can be decomposed into n parallel sub-tasks. For each pair of module M_i and local specification φ_i, we envisage the following procedure:

1: Generate the weakest assumption WA_i from the local specification φ_i;
2: $d_i \leftarrow 1$;
3: **while** $\mathcal{C}_i^{d_i} \nvDash WA_i$ **do**
4: **if** $\mathcal{N}_i^{d_i} \neq \mathcal{N}_i^{d_i+1}$ **then**
5: $d_i \leftarrow d_i + 1$;
6: **else**
7: **return false**;
8: **return true**;

The weakest assumption WA_i is suitable for checking an increasing number of modules as the while-loop continues (Line 3). Since the number of modules is finite, this procedure will terminate: either the assumption WA_i is guaranteed (Line 8), or all the modules that M_i reacts with have been checked (Line 7).

4 Case Study

One of our motivations for investigating assume-guarantee reasoning was to broaden the range of applications in the area of network control. We particularly wish to reason about the *overall* objectives or behaviour of the control algorithm implemented by a protocol. This section illustrates an application of rule $\mathbf{R}^{\pi}$ to verify the stability of an optimisation-based congestion control system. Both the dynamic system and the stability property exhibit compositional structures. We refer to our previous work [17] for more details about the system and the property we considered.

Multi-Path Congestion Control. For tractability, we devise a discrete version of the fluid-flow congestion control algorithm proposed by Kelly and Voice [15].

Consider a network in which a finite number of sources communicate with a finite number of destinations. Between each pair of source and destination a number of routes have been provisioned. Let $r \in s$ denote that route r is available to source s and $s(r)$ be the source that transmits along route r. Each route uses a number of links or, more generally, resources, each of which has a finite capacity constraint. Let $j \in r$ denote that resource j is used by route r.

Then, for each source s and route r available to s, the discrete trajectory in the flow rate x_r is subject to the following equation:

$$x'_r = x_r + \kappa_r x_r \left(1 - \frac{1}{\alpha_{s(r)}} \sum_{j \in r} \beta_j x_j \sum_{r' \in s(r)} x_{r'}\right)^+_{x_r} \tag{1}$$

where κ_r is a constant, x'_r is the next value of x_r and

- α_s is the utility co-efficient of source s;
- β_j is the price co-efficient of resource j;
- x_j is the aggregate flow rate at resource j, i.e., $x_j = \sum_{j \in r} x_r$;
- $(z)^+_x = \min(0, z)$ if $x \leq 0$, otherwise $(z)^+_x = z$.

Thus, each source s adjusts the flow rate x_r on route $r \in s$ based on feedback $\beta_j x_j$ from every resource $j \in r$ in the network (indicating congestion). Then, the algorithm presented in [15] is composed of these sources acting synchronously and collectively. The stability of this synchronous algorithm has been proved in [15]. Herein, we analyse the fully asynchronous variant of the algorithm under the fairness constraint that every source acts infinitely often. This asynchronous model captures uncertain delay between distributed sources.

Stability. System stability is a key property of interest for a distributed congestion control system. A system is stable if it equilibrates at certain network-wide flow configuration, i.e., where $x'_r = x_r$ for every route r. Let s_i range over all the sources. Then, the following CTL formula

$$AFAG \bigwedge_{s_i} \left(\bigwedge_{r \in s_i} x'_r = x_r \right) \tag{2}$$

represents system stability, that is, the system will always eventually be stable.

Lagrangian decomposition techniques reduce system stability onto individual modules [19]. A distributed source is stable if certain stable flow configuration is reached on all the routes using the resources consumed by the source. Let $\gamma(s_i)$ denote the set of the routes serving or sharing resource with source s_i, i.e., $\gamma(s_i) = \{r \mid j \in r \text{ for any } r' \in s_i \text{ and } j \in r'\}$. Then, local stability on source s_i is represented by the following CTL formula

$$AFAG \ (\bigwedge_{r\in\gamma(s_i)} x'_r = x_r) \tag{3}$$

Observe that the global specification (2) is equivalent to the conjunction of local specifications (3) on all the sources. Therefore, we instantiate rule $\mathbf{R}^\pi$ as rule **SS** below for **s**ystem **s**tability:

$$\mathbf{SS}\ \frac{\begin{array}{c}\forall 1 \le i \le n,\ M_i|A_i \ \vDash AFAG \bigwedge_{r\in\gamma(s_i)} x'_r = x_r \\ \exists 1 \le d_i \le \pi,\ \mathcal{C}_i^{d_i} \vDash A_i\end{array}}{M_1|\cdots|M_n \ \vDash \ AFAG \bigwedge_{s_i}(\bigwedge_{r\in s_i} x'_r = x_r)}$$

where source s_i is represented as module M_i.

Computing Assumptions. By rule **SS**, the assumption A_i for module M_i is such that $M_i|A_i$ satisfies the local specification (3). Thus, assumption A_i concerns only on the variables in $X_i \cup I_i$, and is meant to supply sequences of inputs to module M_i such that $M_i|A_i$ can eventually converge to certain configuration on $X_i \cup I_i$.

Note that under rule SYM or ASYM, assumption A_i has to concern on all the variables in X. A local stable state on $X_i \cup I_i$ would be extended to a global stable states on X to meet the global specification (2). Since module M_i controls only the variables in X_i, all the variables in $X\backslash(X_i \cup I_i)$ can converge to any possible combinations of values in domain D. Hence, for every local stable state on $X_i \cup I_i$, assumption A_i has to cover all the corresponding $|D|^{|X\backslash(X_i\cup I_i)|}$ global stable states. Such redundancy is avoided under rule **SS** by generating assumption A_i from the local specification (3).

For module $M_i = (X_i, I_i, Q_{M_i}, T_{M_i}, \lambda_{M_i}, q_{0_{M_i}})$, the assumption can be constructed as a tuple $A_i = (I_i, X_i, E_{A_i} \cup F_{A_i}, T_{A_i}, \lambda_{A_i}, q_{0_{A_i}}, F_{A_i})$ where E_{A_i}, F_{A_i}, T_{A_i} and λ_{A_i} are the minimal sets of non-accepting states, accepting states, transitions and the labelling function derived through the following algorithm, respectively.

1. For each valuation α on I_i, there exists one and only one state $p \in E_{A_i}$ such that $\lambda_{A_i}(p) = \alpha$.
2. For any $q \xrightarrow{\alpha}_{M_i} q'$ and the state $p \in E_{A_i}$ such that $\lambda_{A_i}(p) = \alpha$, $p \xrightarrow{\lambda_{M_i}(q)}_{A_i} p'$ for all $p' \in E_{A_i}$.
3. For any $q \xrightarrow{\alpha}_{M_i} q$, there exists one and only one state $p_q \in F_{A_i}\backslash E_{A_i}$ such that $\lambda_{A_i}(p_q) = \alpha$ and
   - $p_q \xrightarrow{\lambda_{M_i}(q)}_{A_i} p_q$;
   - $p \xrightarrow{\lambda_{M_i}(q)}_{A_i} p_q$, where $p \in E_{A_i}$ is the state such that $\lambda_{A_i}(p) = \alpha$;

4. $q_{A_{i_0}}$ is the initial state, and $\lambda(q_{A_{i_0}})$ is the given initial configuration on I_i.

Intuitively, step 1 logs all possible inputs to module M_i as the non-accepting states of assumption A_i; while step 2 traces the state changes of module M_i as the transitions of assumption A_i. Step 3 defines the accepting states of assumption A_i to characterise all configuration on $X_i \cup I_i$ where $M_i|A_i$ can possibly settle. Each self-loop transition $q \xrightarrow{\alpha}_{M_i} q$ contributes to an accepting state p_q with $\lambda_{A_i}(p_q) = \alpha$. Apparently, module M_i at state q would remain at this state under constantly repeated inputs α, which is exactly what the local specification (3) expects.

Thus, we compute an assumption A_i for module M_i based on the module itself, regardless of the underlying topology. Theorem 5 shows that the assumption is an appropriate one for our purpose.

Theorem 5. *Assumption A_i generated by the above algorithm for module M_i is the weakest assumption with respect to the local specification* (3)*.*

Proof. By definition, it can be seen that any accepting trace of $M_i|A_i$ will fall into an infinite loop at some state (q, p_q), where $q \in Q_{M_i}$ admits a self-loop transition under input $\lambda_{A_i}(p_q)$. Correspondingly, the infinite word accepted through such an accepting trace will terminate with an infinite loop of the valuation on $\lambda_{M_i}(q) \cup \lambda_{A_i}(p_q)$. Therefore, $M_i|A_i$ satisfies the local specification (3).

We then prove by contradiction that assumption A_i is the weakest assumption with respect to the local specification (3). Suppose there exists an assumption A_i' such that $\mathcal{L}(A_i') \subseteq \mathcal{I}(M_i)$ and $M_i|A_i'$ satisfies the local specification (3), but there exists an infinite word $\theta = \alpha_0\alpha_1 \cdots \in \mathcal{L}(A_i')$ that is not accepted by A_i. Then, by this hypothesis and the definition of step 3, θ cannot be derived by A_i.

Assume $\alpha_0 \ldots \alpha_k$ $(k \geq 0)$ is the longest prefix that can be derived from A_i. This means that, for any valuation ρ on X_i, no transition $p \xrightarrow{\rho}_{A_i} p'$ exists such that $\lambda_{A_i}(p) = \alpha_k$ and $\lambda_{A_i}(p') = \alpha_{k+1}$. Hence, by the definition of step 2, no transition $q \xrightarrow{\alpha_k}_{M_i} q'$ exists such that for any states $q, q' \in Q_{M_i}$. This conflicts with the hypothesis, which implies $\theta \in \mathcal{I}(M_i)$.

The time complexity of this algorithm is linear to the size of module M_i. The worst run-time is $O(2|T_{M_i}|)$. The size of the resulting assumption A_i is also linear to the size of module M_i. In the worst case, assumption A_i contains $|D|^{|I_i|} + |T_{M_i}|$ number of states and $|T_{M_i}||D|^{|I_i|} + 2|T_{M_i}|$ number of transitions.

By omitting step 4, this algorithm can be revised to generate a *super* assumption with the universal set of all possible initial states, each labelled with a valuation on I_i. The language accepted by the super assumption is then the disjoint union of the languages accepted by the assumptions under each possible initial valuation on I_i.

5 Experiments

This section illustrates how reduced assumptions can help improve the efficiency and scalability of assume-guarantee reasoning. Specifically, we show how one set

of verification checks under rule **SS** can prove stability regardless of the number of sources and their initial flow configurations, and for any topology of bounded degree.

Consider a simple topology where each source is provisioned with two routes and each resource is shared by two sources. Thus, each source module has two state variables and two input variables. Let $M_{u,v}$ be a source with an initial configuration $(u, v) \in D^2$ and the transitions defined by Equation (1). Then, no matter how many sources a network may consist of, each source is of the general form $M_{u,v}$, where $u, v \in D$.

Let $A_{u,v}$ be the super assumption generated by the above algorithm for module $M_{u,v}$. We start with checking whether the composition of any two possible neighbour modules can guarantee these assumptions. This amounts to check whether

$$M_{u_1,v_1'}|M_{u_1',v_1} \vDash A_{u_0,v_0} \tag{4}$$

for any initial configuration $(u_0, v_0, u_1, v_1, u_1', v_1') \in D^6$. For the domain $D = [1, 6]$ this means that $6^6 (= 46656)$ instances of Equation (4) need to be verified. These checks are done through, as usual, by establishing whether any infinite word derived by $M_{u_1,v_1'}|M_{u_1',v_1}$ can be accepted by coA_{u_0,v_0}, the complement of assumption A_{u_0,v_0}.

We use the GOAL tool [25] to compute and simplify each complement coA_{u_0,v_0}. Each assumption A_{u_0,v_0} and its complement coA_{u_0,v_0} are encoded as Büchi automata in GOAL. Table 1 reports the size of each automaton in terms of the number of states (in Columns *#states*) and the number of transitions (in Columns *#transitions*), and the time usage in seconds for complementing each assumption A_{u_0,v_0} (in Column *time*). Note that M_{v_0,u_0} is equivalent to M_{u_0,v_0} under permutation. For sake of comparison, Table 1 also reports the size of each assumption $A^{\psi}_{u_0,v_0}$, generated from the global specification (2), and the time usage (in seconds) for complementing it. The symbol '-' means that the tool did not return any result within 10 hours. All experiments were ran on a Linux server with two Intel 2.8GHz Quad Core Xeon processors and 16G memory. Observe that GOAL is not a tool optimised for speed; faster results may possibly be achievable.

It can be seen that assumptions for each module M_{u_0,v_0} are greatly reduced under rule **SS**. In average each assumption A_{u_0,v_0} is reduced by a factor of 36 times in the number of states and a factor of 569 in the number of transitions compared with the corresponding assumption $A^{\psi}_{u_0,v_0}$. This is because the combinatorial explosion with the redundant variables in $X \backslash (X_i \cup I_i)$ for each module M_i is avoided without loss of expressiveness of the assumptions. The advantage of using reduced assumptions is particularly apparent when computing their complements. The tool took no more than 2.5 minutes to complement each assumption A_{u_0,v_0}, but only 10 out of 21 complementation instances $coA^{\psi}_{u_0,v_0}$ can be computed by the tool. Considering that it is very time-consuming to simplify a Büchi automaton, our approach is more efficient than that of applying the general assume-guarantee rules with simplified assumptions $A^{\psi}_{u_0,v_0}$.

Table 1. Experimental Results for Computing Assumptions

u_0	v_0	$A^{\psi}_{u_0,v_0}$			A_{u_0,v_0}			coA_{u_0,v_0}	
		#states	#transitions	time(s)	#states	#transitions	time(s)	#states	#transitions
1	1	1332	49248	1511.0	37	108	3.3	73	2628
1	2	1332	49248	1475.1	37	108	1.9	73	2628
1	3	1332	49248	1415.8	37	108	1.7	73	2628
1	4	2016	97200	3292.9	56	180	3.5	110	3960
1	5	2268	144288	4247.5	63	228	4.9	123	4428
1	6	2304	190944	5693.8	64	264	5.0	124	4464
2	2	1332	49248	1477.2	37	108	1.6	73	2628
2	3	4752	195840	14207.2	132	400	19.3	114	4104
2	4	5760	291024	21088.4	160	524	31.5	168	6048
2	5	5796	337680	25180.2	161	560	33.1	169	6084
2	6	6084	431424	-	169	644	34.6	183	6588
3	3	8532	389736	-	237	746	77.3	174	6264
3	4	9648	531720	-	268	910	103.5	233	8388
3	5	9684	578376	-	269	946	106.4	234	8424
3	6	9756	671688	-	271	1018	105.9	236	8496
4	4	8568	436392	-	238	782	74.7	175	6300
4	5	9684	578376	-	269	946	108.1	234	8424
4	6	9684	578376	-	269	946	104.1	234	8424
5	5	10836	767016	-	301	1146	145.1	294	10584
5	6	10836	767016	-	301	1146	138.0	294	10584
6	6	10836	767016	-	301	1146	138.1	294	10584

Equation (4) was verified in our experiments for all the values of parameters $u_0, v_0, u_1, v_1, u_1', v_1'$ in domain D. As a consequence, any assumption A_{u_0,v_0} can be guaranteed by the composition of any two possible modules. Thus, our experiments show the stability of such system for *any* number of sources and *any* initial flow configuration under the given topology.

Furthermore, the experiments reported can be extended for any topology with bounded degree (i.e., each source is sharing resources with a bounded number of other sources). Suppose each source has at most m routes, the general form of each module is $M_{\boldsymbol{u}}$, where vector $\boldsymbol{u}$ ranges over $\bigcup_{k=1}^{m} D^k$. This is particularly appealing to us as previous results in the literature on verification of congestion control models (e.g., [26, 17]) apply only to fixed network topologies.

6 Conclusions

The paper presents a methodology for assume-guarantee reasoning for global specifications consisting of conjunctions of local specifications. The rule $\mathbf{R}^{\pi}$ presented is both sound and complete for local specifications, yet can be applied to draw conclusions on global specifications. Thus, a verification task on a system can be decomposed onto individual modules and local specifications. The methodology is based on an incremental approach to exploit the neighbourhood

dependency between modules. Each increment explores the modules' interactions one step further into the neighbourhood.

We applied the rule to verify the stability of a distributed congestion control system with any number of modules, any initial state, and any topology of bound degree. We proved system stability by considering only local stability of each individual source when interacting with its neighbours. In this way, the technique presented could greatly extend the range of network problems that model checking could be applied to.

References

1. Abadi, M., Lamport, L.: Conjoining specifications. ACM Transactions on Programming Languages and Systems 17(3), 507–535 (1995)
2. Alur, R., Henzinger, T.A.: Reactive modules. In: Proc. 11th Annual IEEE Symposium on Logic in Computer Science Logic in Computer Science (LICS'96). pp. 207–218. New Brunswick, USA (27–30 July 1996)
3. Alur, R., Henzinger, T.A., Mang, F., Qadeer, S., Rajamani, S.K., Tasiran, S.: MOCHA: Modularity in model checking. In: Proc. 10th International Conference on Computer-aided Verification (CAV'98). pp. 521–525. Vancouver, Canada (28 June–2 July 1998)
4. Barringer, H., Giannakopoulou, D., Păsăreanu, C.S.: Proof rules for automated compositional verification through learning. In: Proc. 2003 Workshop on Specification and Verification of Component-Based Systems (SAVCBS'03). pp. 14–21. Helsinki, Finland (1–2 September 2003)
5. Berezin, S., Campos, S.V.A., Clarke, E.M.: Compositional reasoning in model checking. In: Revisted Lectures from Proc. International Symposium on Compositionality. pp. 81–102. Bad Malente, Germany (8–12 September 1997)
6. Cimatti, A., Clarke, E.M., Giunchiglia, E., Giunchiglia, F., Pistore, M., Roveri, M., Sebastiani, R., Tacchella., A.: NuSMV 2: An opensource tool for symbolic model checking. In: Proc. 14th International Conference on Computer-Aided Verification (CAV'02). Copenhagen, Denmark (July 27–31 2002)
7. Cobleigh, J., Giannakopoulou, D., Păsăreanu, C.: Learning assumptions for compositional verification. In: Proc. 9th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'03). pp. 331–346. Warsaw, Poland (7–11 April 2003)
8. Farzan, A., Chen, Y.F., Clarke, E.M., Tsay, Y.K., Wang, B.Y.: Extending automated compositional verification to the full class of omega-regular languages. In: Proc. 14th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'08). pp. 2–17. Budapest, Hungary (29 March–6 April 2008)
9. Francez, N., Pnueli, A.: A proof method for cyclic programs. Acta Informatica 9(2), 133–157 (1978)
10. Gheorghiu Bobaru, M., Păsăreanu, C.S., Giannakopoulou, D.: Automated assume-guarantee reasoning by abstraction refinement. In: Proc. 20th International Conference on Computer Aided Verification (CAV'08). pp. 135–148. Princeton, USA (7–14 July 2008)
11. Giannakopoulou, D., Păsăreanu, C.S., Barringer, H.: Assumption generation for software component verification. In: Proc. 17th IEEE International Conference

on Automated Software Engineering (ASE'02). pp. 3–12. Edinburgh, UK (23–27 September 2002)
12. Grumberg, O., Long, D.E.: Model checking and modular verification. ACM Transactions on Programming Languages and Systems 16(3), 843–871 (1994)
13. Henzinger, T.A., Qadeer, S., Rajamani, S.K.: You assume, we guarantee: Methodology and case studies. In: Proc. 10th International Conference on Computer Aided Verification (CAV'98). pp. 440–451. Vancouver, Canada (28 June–02 July 1998)
14. Jones, C.B.: Tentative steps toward a development method for interfering programs. ACM Transactions on Programming Languages and Systems 5(4), 596–619 (1983)
15. Kelly, F., Voice, T.: Stability of end-to-end algorithms for joint routing and rate control. ACM SIGCOMM Computer Communication Review 35(2), 5–12 (2005)
16. Kupferman, O., Vardi, M.Y.: An automata-theoretic approach to modular model checking. ACM Transactions on Programming Languages and Systems 22(1), 87–128 (2000)
17. Lomuscio, A., Strulo, B., Walker, N., Wu, P.: Model checking optimisation-based congestion control models. In: Proc. 2009 Workshop on Concurrency, Specification, and Programming (CS&P 2009). pp. 386–397. Kraków-Przegorzały, Poland (28–30 September 2009)
18. Lomuscio, A., Strulo, B., Walker, N., Wu, P.: Assume-guarantee verification for distributed systems with local specifications. Tech. Rep. RN/10/01, Department of Computer Science, University College London (February 2010)
19. Low, S.H., Lapsley, D.E.: Optimization flow control, I: basic algorithm and convergence. IEEE/ACM Transactions on Networking 7(6), 861–874 (1999)
20. Maier, P.: Compositional circular assume-guarantee rules cannot be sound and complete. In: Proc. 6th International Conference on Foundations of Software Science and Computational Structures (FoSSaCS'03). pp. 343–357. Warsaw, Poland (7–11 April 2003)
21. Misra, J., Chandy, K.M.: Proof of networks of processes. IEEE Transactions on Software Engineering SE-7(4), 417–426 (1981)
22. Nam, W., Alur, R.: Learning-based symbolic assume-guarantee reasoning with automatic decomposition. In: Proc. 4th International Symposium on Automated Technology for Verification and Analysis (ATVA'06). pp. 170–185. Beijing, China (23–26 October 2006)
23. Nam, W., Madhusudan, P., Alur, R.: Automatic symbolic compositional verification by learning assumptions. Formal Methods in System Design 32(3), 207–234 (2008)
24. Păsăreanu, C.S., Giannakopoulou, D., Bobaru, M.G., Cobleigh, J.M., Barringer, H.: Learning to divide and conquer: applying the L* algorithm to automate assume-guarantee reasoning. Formal Methods in System Design 32(3), 175–205 (2008)
25. Tsay, Y.K., Chen, Y.F., Tsai, M.H., Wu, K.N., Chan, W.C.: GOAL: A graphical tool for manipulating büchi automata and temporal formulae. In: Proc. 13th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'07). pp. 466–471. Braga, Portugal (24 March–1 April 2007)
26. Yuen, C., Tjioe, W.: Modeling and verifying a price model for congestion control in computer networks using promela/spin. In: Proc. 8th International SPIN Workshop on Model Checking of Software (SPIN'01). pp. 272–287. Toronto, Canada (19–20 May 2001)