International Journal of Foundations of Computer Science

Assume-guarantee reasoning with local specifications

Alessio Lomuscio

Department of Computing, Imperial College London, 180 Queen's Gate
London SW7 2AZ, UK
a.lomuscio@imperial.ac.uk

Ben Strulo

BT Innovate, Adastral Park, Martlesham Heath
Ipswich IP5 3RE, UK
ben.strulo@bt.com

Nigel Walker

BT Innovate, Adastral Park, Martlesham Heath
Ipswich IP5 3RE, UK
nigel.g.walker@bt.com

Peng Wu*

State Key Laboratory of Computer Science, Institute of Software
Chinese Academy of Sciences, P.O.Box 8718
Beijing 100190, China
wp@ios.ac.cn

Received
Revised
Accepted
Communicated by

We investigate assume-guarantee reasoning for global specifications consisting of conjunctions of local specifications. We present a sound and complete assume-guarantee methodology that enables us to establish properties of a composite system by checking local specifications of its individual modules. 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 system for network stability and show its efficiency to reason about any number of agents, any initial flow configuration, and any topology of bounded degree.

Keywords: assume-guarantee reasoning; compositional model checking; distributed system

*Corresponding author

1. Introduction

Assume-guarantee reasoning [1, 2, 3] is one of the key techniques to alleviate the state explosion in model checking. In this paradigm a system composed of a number of reactive modules is analysed by considering each module 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 [4, 5, 6, 7]. However, many studies have demonstrated that assumptions may become too large to be treated effectively [8, 9, 10, 11]. 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 model checking, particularly in the context of network control protocols.

Our starting point is the observation that a module in a system typically reacts *directly* with relatively few other modules. However, under the 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. This growth in the local assumptions causes inefficiencies in the general assume-guarantee techniques.

In this paper we show that for a system property that can be formulated as the conjunction of local specifications on individual modules, these scalability issues can be resolved by generating assumptions with respect to local specifications. Our main contribution consists in a number of assume-guarantee rules reasoning with local specifications to derive 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. Through a counterexample, we show that this simple rule is not complete, as it exploits only the direct dependency between modules.

We then extend rule $\mathbf{R_1}$ to achieve completeness. This leads us to a bounded assume-guarantee rule $\mathbf{R}^\pi$ that we prove to be sound and complete. $\mathbf{R}^\pi$ encodes interactions between modules only up to π hops away from each other. We use this rule to propose a bounded assume-guarantee reasoning approach, in which the dependency between modules is exploited incrementally.

In doing so we are inspired by the topological properties of networked systems in which the components, or hosts, interact only through their immediate neighbours. We evaluate the approach through a case study of an optimisation based congestion control system as proposed in [12]. The optimisation approach allows a distributed solution for network congestion control. A congestion control system is stable if 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 the 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 has so far focused on verifying network protocols *under given topologies only*. By contrast, the assume-guarantee framework developed here supports verification of network-wide objectives *irrespective of the underlying network topologies*.

In the case study we analyse the direct neighbourhood dependency is enough for establishing the stability of the system. This shows the potential of our approach when used to reason about global properties of a distributed network within a relatively near neighbourhood.

Related Work. The history of compositional verification of concurrent systems dates back to the late 70s and 80s [13, 14, 1]. Since then, considerable effort has been devoted to studying the soundness of circular assume-guarantee reasoning. [15] showed that compositional circular assume-guarantee rules cannot be both sound and complete. [9] presented an automata-theoretic approach to model checking assume-guarantee assertions.

More recently, [4, 5, 6, 10, 11] developed sound and complete non-circular assume-guarantee reasoning approaches for safety properties, with support for automated learning of assumptions. [16, 17] presented a symbolic implementation of learning-based assume-guarantee reasoning. [10, 11] proposed an alphabet refinement technique to reduce the size of assumptions. [7] extended the assume-guarantee reasoning approaches to liveness properties, based on the observation that ω-regular languages preserve the essential closure properties of regular languages. This was further developed in [18] where a general formalisation framework is presented to use learning in the context of assume-guarantee reasoning.

The starting point for this paper is work on reasoning about local specifications, including studies on compositional verification [19, 8], where only sound circular assume-guarantee rules were proposed for safety properties. We first show that local dependencies can be exploited to generate smaller, hence computationally more attractive, assumptions. The bounded methodology here presented is shown to be sound and complete with respect to liveness properties. Our approach is amenable to implementation using symbolic representation, and integration 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.

Some of the ideas here developed were put forward in previous, preliminary work [20]. However, the material here presented gives more emphasis to the proofs of the technical parts and shows the applicability of the methodology in greater detail.

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

2. Assume-guarantee reasoning

In this section we first introduce the notions of module, specification and assumption in concurrent systems. Then, we present two general assume-guarantee rules SYM and ASYM that have been applied to reason about individual modules over global specifications.

2.1. *Modules*

We adopt the notion of reactive module [21] 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 other modules that the module reacts directly with.

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

Two valuations $\rho_1 : X_1 \to D$ and $\rho_2 : X_2 \to D$ are *compatible*, denoted $\rho_1 \sim \rho_2$, if $\rho_1(x) = \rho_2(x)$ for any $x \in X_1 \cap X_2$. Given two compatible valuations $\rho_1 : X_1 \to D$ and $\rho_2 : X_2 \to D$, $\rho_1 \cup \rho_2 : X_1 \cup X_2 \to D$ 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 *directly depends on;*
- $X \cap I = \emptyset$*;*
- 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* $q \xrightarrow{\alpha}_T q'$*, models the evolution of* M *from state* q *to state* q' *under input* $\alpha : I \to D$*;*
- $q_0 \in Q$ *is the initial state.*

Specifically, module M *is said to be* closed *if* $I = \emptyset$*.*

An infinite trace of module M is an infinite sequence $\sigma = q_0\alpha_0 q_1\alpha_1 \ldots$ 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 $q_0\alpha_0 q_1\alpha_1 \ldots$ 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 $q_0\alpha_0 q_1\alpha_1 \ldots$ 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.

For simplicity of presentation, we consider only deadlock-free modules. A state is a *deadlock* state if there do not exist any $\alpha \in D^I$ and $q' \in Q$ such that $q \xrightarrow{\alpha}_T q'$. A module is *deadlock-free* if it contains no deadlock state. This hypothesis does not restrict the applicability of our approach because a deadlock state could be regarded here as a steady state that remains constant under any input.

We now define the composition operator for compatible modules. Two modules $M_1 = (X_1, I_1, Q_1, T_1, \lambda_1, q_{0_1})$ and $M_2 = (X_2, I_2, Q_2, T_2, \lambda_2, q_{0_2})$ are *compatible* if $\lambda_1(q_{0_1}) \sim \lambda_2(q_{0_2})$. We choose a notion of composition that explicitly supports asynchrony. This is because in distributed environments factors external to the modules, such as network latency or communication scheduling, make asynchronous modelling essential.

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

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

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

  where $\alpha = (\alpha_1 \cup \alpha_2) \restriction_{(I_1 \cup I_2)\backslash(X_1 \cup X_2)}$*,* $\lambda(q_1) \sim \lambda(q_2)$*,* $\lambda(q_1') \sim \lambda(q_2)$*,* $\lambda(q_2') \sim \lambda(q_1)$*,* $\lambda(q_1') \sim \lambda(q_2')$*,* $\lambda(q_2) \sim \alpha_1$*,* $\lambda(q_1) \sim \alpha_2$*, and* $\alpha_1 \sim \alpha_2$*.*
- $(q_{0_1}, q_{0_2}) \in Q$ *is the initial state (note that* $\lambda_1(q_{0_1}) \sim \lambda_2(q_{0_2})$ *because* M_1 *and* M_2 *are compatible).*

Note that rule ASYN$_L$ (respectively, ASYN$_R$) models transitions in which only M_1 (respectively, M_2) evolves; while by rule SYN both M_1 and M_2 evolve simultaneously. The composition rules above are applicable for the concurrent systems considered throughout this paper. In the presence of a number of modules these composition rules permit one, some, or all the modules to evolve simultaneously. The notions of module and composition can be implemented by existing modular languages, such as the input languages of MOCHA [22] and NuSMV [23]. However, in these modelling languages asynchrony is implemented as a non-deterministic choice of the modules themselves.

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. A *stuttering projection* of w on a subset Y of $X_1 \cup X_2$, denoted $w|_Y$, is an infinite word $\rho'_0\rho'_1\ldots$, where there exists $0 = j_0 < j_1 < \cdots$ such that $\rho'_i = \rho_{j_i} \restriction_Y = \rho_{j_i+1} \restriction_Y = \cdots = \rho_{j_{i+1}-1} \restriction_Y$ for any $i \geq 0$. As a special case of stuttering projection, the *restriction* of w on Y, denoted $w \restriction_Y$, is the infinite word $\rho'_0\rho'_1\ldots$, where $\rho'_i = \rho_i \restriction_Y$ for any $i \geq 0$.

Thus, 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 $1 \leq i \neq j \leq n$, and $\bigcup_{i=1}^{n} I_i \subseteq \bigcup_{i=1}^{n} X_i$, constitutes a *closed* concurrent system $M_1|\cdots|M_n$ with a finite set of state variables $X = \bigcup_{i=1}^{n} X_i$. D^X is then referred to as the *global* alphabet of the system. Since $X_i \cap X_j = \emptyset$ for any $1 \leq i \neq j \leq n$, each $x \in X$ is *exclusively controlled* by module M_i such that $x \in X_i$.

2.2. *Specifications*

We now recall the syntax and the semantics of the Linear-time Temporal Logic (LTL) with Presburger constraints [24], that we will use as the specification language. A constraint system is a pair $\mathbb{C} = \langle D, (\mathcal{R}_i)_i \rangle$ where $(\mathcal{R}_i)_i$ is a countable family of relations on domain D. An atomic $\mathbb{C}$-constraint is a term of the form $\mathtt{R}(x_1, \ldots, x_k)$, where $x_1, \ldots, x_k \in X$, $\mathtt{R}$ is interpreted as a relation on domain D and k is the arity of $\mathtt{R}$. A valuation $\rho : X \to D$ satisfies the $\mathbb{C}$-constraint $\mathtt{R}(x_1, \ldots, x_k)$ if $(\rho(x_1), \ldots, \rho(x_k)) \in \mathcal{R}$, where $\mathcal{R}$ is the relation in domain D associated with the symbol $\mathtt{R}$. For instance, $\langle D, = \rangle$ defines the equality constraints on domain D.

The logic CLTL($\mathbb{C}$) is then defined as an extension of LTL where propositional variables are refined by atomic $\mathbb{C}$-constraints over terms. A term, denoted $\mathtt{X}^l x$, represents the variable x prefixed by a number $l \geq 0$ of operators $\mathtt{X}$ for "next" (see below). This can be interpreted as specifying the value of x at the l-th next state. Specifically $\mathtt{X}^1 x$ is abbreviated as x', representing the value of x at the following state. The syntax of CLTL($\mathbb{C}$) can be defined in BNF form as follows:

$$\varphi := \quad \mathtt{R}(\mathtt{X}^{j_1} x_1, \ldots, \mathtt{X}^{j_k} x_k) \mid \neg\varphi \mid \varphi_1 \wedge \varphi_2 \mid \mathtt{X}\varphi \mid \varphi_1 \mathtt{U} \varphi_2$$

The symbols $\mathtt{X}$ and $\mathtt{U}$ are the classical "next" and "until" LTL operators, respectively [25, 26, 27]. The Boolean values $\mathtt{tt}$ and $\mathtt{ff}$ are defined as standard. We use the usual notations $\mathtt{F}\varphi$ and $\mathtt{G}\varphi$ as the abbreviations for $\mathtt{tt}\ \mathtt{U}\ \varphi$ and $\neg\mathtt{F}\neg\varphi$. Observe that the symbol $\mathtt{X}$ is here overloaded as in [24]. The semantics of CLTL($\mathbb{C}$) is defined with respect to a *closed* module $M = (X, \emptyset, Q, T, \lambda, q_0)$, i.e., a module with no input variables. For a formula φ and an infinite word $w = \rho_0\rho_1\cdots$, let $(w, i) \vDash \varphi$ represent that the formula φ holds on the suffix of the word w from the $(i+1)$-th position. The satisfaction relation $\vDash$ is defined inductively as follows:

- $(w, i) \vDash \mathtt{R}(\mathtt{X}^{j_1} x_1, \ldots, \mathtt{X}^{j_k} x_k)$ if $(\rho_{i+j_1}(x_1), \ldots, \rho_{i+j_k}(x_k)) \in \mathcal{R}$;
- $(w, i) \vDash \neg\varphi$ if $(w, i) \nvDash \varphi$;
- $(w, i) \vDash \varphi_1 \wedge \varphi_2$ if $(w, i) \vDash \varphi_1$ and $(w, i) \vDash \varphi_2$;

- $(w, i) \vDash \mathtt{X}\varphi$ if $(w, i+1) \vDash \varphi$;
- $(w, i) \vDash \varphi_1 \mathtt{U} \varphi_2$ if there exists $j \geq i$ such that $(w, j) \vDash \varphi_2$ and for every $i \leq k \leq j$, we have that $(w, k) \vDash \varphi_1$.

A module $M = (X, I, Q, T, \lambda, q_0)$ satisfies a formula φ, denoted $M \vDash \varphi$, if $(w, 0) \vDash \varphi$ for any word w derived by M. In the following we only consider specifications expressed in the syntax above.

2.3. *Assumptions*

Assumptions characterise the abstract environments that individual modules could possibly interact with to make the given specifications hold. For verification purposes assumptions can be defined as modules extended by accepting states. We here focus on liveness properties; therefore, we adopt the formalism of Büchi automata for the definition of assumptions. However, it can be shown that the assume-guarantee rules presented in the following are also valid in the context of safety properties, for which assumptions are defined as finite automata [5]. We do not pursue this here.

Definition 3 (Extended Module) *An extended module 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 above for modules also applies to extended modules. An infinite word $\rho_0 \rho_1 \ldots$ is *accepted* by an extended module A if there exists an infinite trace $\sigma = q_0 \alpha_0 q_1 \alpha_1 \ldots$ of the module A, 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 the module A consists of all the infinite words accepted by the module A. Let coA be the complement of the module A accepting the complement language $\Omega_X \backslash \mathcal{L}(A)$, where Ω_X is the set of infinite words on alphabet D^X. We here rely on existing techniques [28] to compute complements of Büchi automata.

For a module $M = (X_1, I_1, Q_1, T_1, \lambda_1, q_{0_1})$ and an extended module $A = (X_2, I_2, Q_2, T_2, \lambda_2, q_{0_2}, F_A)$, the composition of M with A is an extended module $M|A = (X, I, Q, T, \lambda, q_0, F)$, where $F = \{(q_1, q_2) \in Q \mid q_2 \in F_A\}$ and the rest of the components are as in Definition 2. Moreover, let $M|A \vDash \varphi$ denote that the extended module $M|A$ satisfies a formula φ, where only the accepting traces of $M|A$ are checked with respect to φ. It follows that assumptions can be formally represented as the extended modules that characterise the acceptable executions in question.

Similarly, for extended modules $coA_i = (X_i, I_i, Q_i, T_i, \lambda_i, q_{0_i}, F_i)$ $(i \in \{1, 2\})$, 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 $F = \{(q_1, q_2) \in Q \mid q_1 \in F_1, q_2 \in F_2\}$ and the rest of the components are as in Definition 2. Since our work is motivated by local assumptions, the extended modules coA_1 and coA_2 might not be associated with the same set of variables, i.e., in general $X_1 \neq X_2$. Given this we cannot compute the intersection of $\mathcal{L}(coA_1)$ and $\mathcal{L}(coA_2)$ as in the literature of modular languages.

The following definition formalises the notion of *guarantee* in the context above by linking assumptions to the system's behaviour.

Definition 4 (Guarantee) *For $k \geq 1$ modules $M_i = (X_i, I_i, Q_i, T_i, \lambda_i, q_{0_i})$, 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* $X_A \subseteq \bigcup_{j=1}^{k'} X_{M_{i_j}}$ *and* $X_A \cap X_{M_{i_j}} \neq \emptyset$ *for* $1 \leq j \leq k'$;

then $M_1 | \cdots | M_k$ guarantees *the assumption 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 the assumption A.*

Specifically, if $k' = k$, i.e., $X_A \cap X_i \neq \emptyset$ for any $1 \leq i \leq k$, then $M_1 | \cdots | M_k \vDash A$ 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 the assumption A.

2.4. *General assume-guarantee reasoning*

For a system $M_1 | \cdots | M_n$ and a global specification φ (i.e., a specification defined on the whole state variables $\bigcup_{i=1}^{n} X_i$), assume-guarantee approaches [4, 5, 6, 7] establish conditions on the individual modules that lead to the satisfaction of φ on the overall system. For example, the symmetric rule SYM shown in Fig. 1 uses an assumption A_i for each module M_i such that $M_i | A_i$ satisfies φ, and a further check as to whether these assumptions may cause mutual conflict between each other. By contrast, the asymmetric rule ASYM shown in Fig. 1 uses only assumption A_1 for module M_1 such that $M_1 | A_1$ satisfies φ, and a further check as to whether this assumption A_1 is satisfied by all other modules.

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

Fig. 1. General Assume-Guarantee Rules

A rule is sound if the conclusions (represented under its rule line) drawn from the hypotheses (represented above its rule line) are valid. Conversely, a rule is complete, if whenever the conclusions hold, the hypotheses also hold [5, 15]. The rules SYM and ASYM are sound and complete; so the system satisfies the global specification if and only if there exist certain specific assumptions constrained by the global specification. In the literature these assumptions in question are identified from the

perspective of the global system, i.e., these assumptions are generated with respect to global specifications.

3. Local assume-guarantee reasoning

3.1. *The rules* $\mathbf{R_0}$ *and* $\mathbf{R_1}$

We here observe, however, that it is often the case that each module of a concurrent system controls its state variables under inputs from *only a small proportion* of the other modules. Therefore, in general assume-guarantee methodologies:

- Each assumption A_i for a module M_i may contain valuations of state variables that module M_i does not actually depend on. This may make the size of assumption A_i larger than necessary.
- If the system is extended with the addition of other modules, each assumption A_i may have to be regenerated 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.

In the following we aim to exploit these considerations by identifying assumptions from the perspective of individual modules.

We focus on global specifications φ that can be formulated as the conjunction of local specifications φ_i (i.e., specifications defined on $X_i \cup I_i$) such that $\varphi \Leftrightarrow \bigwedge_{i=1}^{n} \varphi_i$. Under this condition the rule SYM above can be modified as:

$$\mathbf{R_0}\ \frac{\begin{array}{c}\forall i : 1 \leq i \leq n,\ M_i | A_i \models \varphi_i \\ \mathcal{L}(coA_1 | \cdots | coA_n) = \emptyset\end{array}}{M_1 | \cdots | M_n \models \bigwedge_{i=1}^{n} \varphi_i}$$

Note that rule $\mathbf{R_0}$ is not simply an instance of rule SYM. We investigate below the soundness and completeness of rule $\mathbf{R_0}$.

Crucially, while in rule SYM assumptions A_i are all checked with respect to the global specification φ, in rule $\mathbf{R_0}$ each assumption A_i is tested with respect to the corresponding local specification φ_i. In this way the size of each assumption A_i may be reduced because only variables in $X_i \cup I_i \subseteq X$ are involved in assumption A_i.

However, as a side effect of rule $\mathbf{R_0}$, such an assumption A_i may admit more interactions with module M_i than can be admitted by the assumptions checked with respect to the global specification φ. This is because the variables in $X \setminus (X_i \cup I_i)$ are not constrained by the local specification φ_i. For example, consider a system consisting of the four modules M_i $(1 \leq i \leq n)$ shown in Fig. 2.

For each module M_i, the CLTL($\langle D, = \rangle$) formula

$$\mathtt{FG}\ (\bigwedge_{x \in X_i \cup I_i} (x' = x)) \tag{1}$$

specifies that the values of the variables in $X_i \cup I_i$ will always eventually remain unchanged for ever, i.e., they will stabilise. Observe that the formula $x' = x$ encodes

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	$\{x_4\}$	$\{x_2, x_3\}$	$x_4' = 1$

Fig. 2. A counterexample for the soundness of rule $\mathbf{R_0}$

an equality constraint in $\langle D, = \rangle$ applied on terms $x(i.e., \mathtt{X}^0 x)$ and $x'(i.e., \mathtt{X}^1 x)$ [24].

Consider 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

$$M_1|M_2|M_3|M_4 \not\models \bigwedge_{i=1}^{4} \mathtt{FG}\, (\bigwedge_{x \in X_i \cup I_i} (x' = x))$$

because M_2 and M_3 evolve by diverging from each other. However, the modules M_1, M_2 and M_3 themselves could stabilise under certain inputs. So, there exist assumptions A_i such that $M_i|A_i$ satisfies the local specification (1) for $i \in \{1, 2, 3\}$.

Obviously, module M_4 is already in a stable state no matter what inputs may be. So, an assumption A_4 that accepts any word on D^{I_4} satisfies the premise

$$M_4|A_4 \vDash \mathtt{FG}\, (\bigwedge_{x \in X_4 \cup I_4} (x' = x)).$$

Then, we have that $\mathcal{L}(coA_4) = \emptyset$. Hence, for any assumptions A_i ($i \in \{1, 2, 3\}$) such that $M_i|A_i$ satisfies the local specification (1), we have that $\mathcal{L}(coA_1|coA_2|coA_3|coA_4) = \emptyset$.

We conclude that the tentative rule $\mathbf{R_0}$ above does not preserve soundness, although its completeness is not affected by the weaker assumptions, as we show below.

Theorem 5 (Completeness of $\mathbf{R_0}$) *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*

$$\mathcal{L}(coA_1|\ldots|coA_n) = \emptyset.$$

Proof. For each module M_i assume that WA_i is the weakest assumption with respect to φ_i [5, 16], i.e.,

- $\mathcal{L}(WA_i) \subseteq \mathcal{I}(M_i)$ 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 $M_i|A_i \vDash \varphi_i$.

Since $M_1|\cdots|M_n \vDash \bigwedge_{i=1}^{n} \varphi_i$ and $\bigwedge_{i=1}^{n} \varphi_i$ implies φ_j for any $1 \leq j \leq n$, such weakest assumption does exist for each module M_i.

We show the result by contradiction. Suppose there exists an infinite word w accepted by $coWA_1|\cdots|coWA_n$. Hence, there exists a stuttering projection of w

on each $X_j \cup I_j$ $(1 \leq j \leq n)$, denoted w_j, that is accepted by $coWA_j$. So, for any $1 \leq j \leq n$, there exists an infinite word w'_j accepted by $M_j|coWA_j$ such that w'_j does not satisfy φ_j. Thus, there exists an infinite word w' derived by $M_1|\cdots|M_n$ such that w'_j is a stuttering projection of w' on $X_j \cup I_j$ for any $1 \leq j \leq n$. Therefore, w' does not satisfy $\bigwedge_{i=1}^{n} \varphi_i$. This conflicts with the premise $M_1|\cdots|M_n \models \bigwedge_{i=1}^{n} \varphi_i$. □

In the following we intend to regain soundness of local assume-guarantee rules by exploiting the neighbourhood dependency between individual modules. For modules $M_1 = (X_1, I_1, Q_1, T_1, \lambda_1, q_{0_1})$ and $M_2 = (X_2, I_2, Q_2, T_2, \lambda_2, q_{0_2})$, M_1 *directly depends on* M_2 if $X_2 \cap I_1 \neq \emptyset$, i.e., module M_1 depends on the inputs from, reacts directly with, or, in other words, is one hop away from module M_2. In this case, module M_2 is referred to as a *direct neighbour* of module M_1.

Let $\mathcal{D} = \{(M_i, M_j) \mid 1 \leq i, j \leq n, I_i \cap X_j \neq \emptyset\}$ be the direct dependency relation between the modules of the system $M_1|\cdots|M_n$. For module M_i let $\mathcal{N}_i$ be the set of all the modules M_j such that $(M_i, M_j) \in \mathcal{D}$, and $\mathcal{C}_i$ be the composition of all the modules in $\mathcal{N}_i$. Note that $\mathcal{D}$ is irreflexive because $I_i \cap X_i = \emptyset$ for each module $M_i = (X_i, I_i, Q_i, T_i, \lambda_i, q_{0_i})$. Then, we introduce rule $\mathbf{R_1}$, which is inspired by rule ASYM.

$$\mathbf{R_1}\ \frac{\forall i : 1 \leq i \leq n, \quad \begin{array}{c} M_i|A_i \models \varphi_i \\ \mathcal{C}_i \models A_i \end{array}}{M_1|\cdots|M_n \models \bigwedge_{i=1}^{n} \varphi_i}$$

Theorem 6 shows the soundness of rule $\mathbf{R_1}$ with respect to local specifications.

Theorem 6 (Soundness of $\mathbf{R_1}$) *Assume that for any module M_i $(1 \leq i \leq n)$ there exists an assumption A_i such that $M_i|A_i \models \varphi_i$ and $\mathcal{C}_i \models A_i$. Then we have that $M_1|\cdots|M_n \models \bigwedge_{i=1}^{n} \varphi_i$.*

Proof. By contradiction. Consider an infinite word $w = \rho_0\rho_1\ldots$ on the global alphabet D^X (i.e., each ρ_i is a valuation on X) that makes the conclusion fail on some φ_j $(1 \leq j \leq n)$. Since, by the definition of the system $M_1|\cdots|M_n$, the state variables in X_j are exclusively controlled by M_j, any stuttering projection $w|_{X_j \cup I_j}$ is not accepted by $M_j|A_j$ and so any stuttering projection $w|_{I_j}$ is not accepted by A_j.

However, for each $M_{j_l} \in \mathcal{N}_j$, the variables in X_{j_l} are exclusively controlled by M_{j_l}. By the composition rules in Definition 2, there exists a stuttering projection of w on $\bigcup_{M_{j_l} \in \mathcal{N}_j} X_{j_l}$, denoted w', that is derived by $\mathcal{C}_j$. Recall that $\mathcal{C}_j$ is the composition of all the modules in $\mathcal{N}_j$, that is, $\mathcal{C}_j$ is composed of all the modules that module M_i directly depends on. So we have that $I_j \subseteq \bigcup_{M_{j_l} \in C_j} X_{j_l}$. Then, by the premise $\mathcal{C}_j \models A_j$, 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. □

While rule $\mathbf{R_1}$ is sound, it is not complete. This is because $\mathcal{C}_i$ may constitute an over-approximation of module M_i's environment without being constrained by the other modules in the system. For example, consider a system consisting of the following four modules M_i $(1 \leq i \leq 4)$ shown in Fig. 3.

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	$\{x_4\}$	$\{x_2, x_3\}$	$x_4' = \begin{cases} 1 & x_2 > x_3 \text{ and } x_4 > 0 \\ -1 & x_2 < x_3 \text{ and } x_4 < 0 \\ 0 & \text{otherwise} \end{cases}$

Fig. 3. A counterexample for the completeness of rule $\mathbf{R_1}$

Consider 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

$$M_1|M_2|M_3|M_4 \models \bigwedge_{i=1}^{4} \mathtt{FG} \left(\bigwedge_{x \in X_i \cup I_i} (x' = x) \right)$$

because M_2 and M_3 evolve by converging in 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 stabilise x_1 (in M_1), and so cannot be accepted by any assumption A_1 that satisfies the premise

$$M_1|A_1 \models \mathtt{FG} \bigwedge_{x \in X_1 \cup I_1} (x' = x).$$

3.2. *Bounded assume-guarantee reasoning*

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

For the modules of the system $M_1|\cdots|M_n$ let $\mathcal{D}^k$ denote the irreflexive k-dependency relation defined recursively as follows:

$$\mathcal{D}^k = \begin{cases} \mathcal{D} & k = 1 \\ \mathcal{D}^{k-1} \cup \{(M_i, M_j) \mid 1 \leq i \neq j \leq n \text{ and } (M_i, M_j) \in \mathcal{D}^{k-1} \circ \mathcal{D}\} & k > 1 \end{cases}$$

where $\circ$ is the composition operator of binary relations. So, $(M_i, M_j) \in \mathcal{D}^k$ encodes the fact that module M_j is within the range of k hops away from module M_i. Recall that $\mathcal{D}$ itself is irreflexive by definition.

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

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

Observe that while rule $\mathbf{R_1}$ considers only the direct neighbours of M_i, rule $\mathbf{R_k}$ checks *all* the modules within the range of k hops away from module M_i. It can be proved that rule $\mathbf{R_k}$ is sound for any $k \geq 1$.

Theorem 7 (Soundness of $\mathbf{R_k}$) *Assume that for any module M_i $(1 \leq i \leq n)$, there exists an assumption A_i such that $M_i | A_i \models \varphi_i$ and $\mathcal{C}_i^k \models A_i$, $k \geq 1$. Then we have that $M_1 | \cdots | M_n \models \bigwedge_{i=1}^{n} \varphi_i$.*

Proof. By contradiction. The proof is similar to that of Theorem 6. □

Observe that if the modules within k hops away from module M_i can guarantee an assumption A_i, then such guarantee is preserved by the modules within $k+1$ hops. This is because the additional modules do not influence the behaviours of those modules within k hops. Based on this observation, Theorem 8 relates rule $\mathbf{R_k}$ with rule $\mathbf{R_{k+1}}$.

Theorem 8. *Let A_i be an assumption for module M_i. Then, if $\mathcal{C}_i^k \models A_i$, then we have that $\mathcal{C}_i^{k+1} \models A_i$.*

Proof. By the definition of $\mathcal{D}^k$, we have that $\mathcal{N}_i^k \subseteq \mathcal{N}_i^{k+1}$. 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 $\mathcal{C}_i^k \models A_i$, $w' \restriction_{I_i}$ would be accepted by A_i for any such w'. □

Since the system $M_1 | \cdots | M_n$ consists of a finite number n of state variables, there exists an irreflexive transitive dependency closure $\mathcal{D}^\pi$ $(\pi \geq 1)$ such that $\mathcal{D}^\pi = \mathcal{D}^{\pi+1}$. Theorem 9 shows that rule $\mathbf{R}_\pi$ is complete with respect to local specifications. Observe that rule $\mathbf{R}_\pi$ is the instantiation of rule $\mathbf{R_k}$ with $k = \pi$.

Theorem 9 (Completeness of $\mathbf{R_\pi}$) *Suppose $\mathcal{D}^\pi$ is the irreflexive transitive dependency closure of the system $M_1 | \cdots | M_n$. If $M_1 | \cdots | M_n \models \bigwedge_{i=1}^{n} \varphi_i$, then for each module M_i there exists an assumption A_i such that $M_i | A_i \models \varphi_i$ and $\mathcal{C}_i^\pi \models A_i$.*

Proof. By construction. Suppose $M_j = (X_j, I_j, Q_j, T_j, \lambda_j, q_{0_j})$ and $\mathcal{C}_j^\pi = (X_j^\pi, I_j^\pi, Q_j^\pi, T_j^\pi, \lambda_j^\pi, q_{0_j}^\pi)$ for any $1 \leq j \leq n$. For each module M_j, we can build an as-

sumption $A_j = (I_j, I_j^\pi, Q_j^\pi, T_j^\pi, \lambda_{A_j}, q_{0_j}^\pi, Q_j^\pi)$, where all the states in Q_j^π is accepting states and λ_{A_j} is the restriction of λ_j^π on I_j, i.e., $\lambda_{A_j} = \lambda_j^\pi \restriction_{I_j}$.

By the definition of $\mathcal{D}^\pi$, we have that $I_j \subseteq X_j^\pi$ and hence $\mathcal{C}_j^\pi \vDash A_j$ for any $1 \leq j \leq n$. Note also that $\bigwedge_{i=1}^{n} \varphi_i$ implies φ_j for any $1 \leq j \leq n$. Since the variables in any X_j are exclusively controlled by module M_j irrespective of the modules not in $M_j|\mathcal{C}_j^\pi$, we have that $I_j^\pi \subseteq X_j$ and hence $M_j|\mathcal{C}_j^\pi \vDash \varphi_j$ for any $1 \leq j \leq n$. □

As a corollary of Theorem 7, 8 and 9, rule $\mathbf{R}_\pi$ can be reformulated as rule $\mathbf{R}^\pi$ (shown below), which can similarly be shown to be sound and complete with respect to local specifications.

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

With rule $\mathbf{R}_\pi$ the scalability issues discussed in Section 3 can be avoided. Indeed, we note that:

- All assumptions A_i in rule $\mathbf{R}_\pi$ are checked with respect to local specifications φ_i that contain only the variables in $X_i \cup I_i$. Thus, all assumptions A_i concern only the variables (in X_i) that modules M_i control and the ones (in I_i) that modules M_i directly depend on, rather than the whole system variables (in X).
- Whenever the system is extended with additional modules, the assumptions A_i may still be reused for the verification of the extended system, without taking the additional variables into account.

Compared to rule ASYM, rule $\mathbf{R}_\pi$ requires only local assumptions that are defined with respect to local specifications. As shown later in a case study, the overall verification task may benefit from this reduction. Moreover, these local assumptions are not valid for the general assume-guarantee rules such as rule ASYM. For instance, in the aforementioned counterexamples, any local assumption A_1 satisfying the premise

$$M_1|A_1 \vDash \mathtt{FG}(\bigwedge_{x \in X_1 \cup I_1} x' = x)$$

is too weak to satisfy the premise

$$M_1|A_1 \vDash \bigwedge_{i=1}^{4} \mathtt{FG}(\bigwedge_{x \in X_i \cup I_i} x' = x).$$

because the variables in $\bigcup_{i=2}^{4} X_i \cup I_i$ are not constrained properly in the assumption A_1.

To apply rule $\mathbf{R}^\pi$ we are required to generate the assumptions A_i in the antecedent of the rule. Given considerations of generality and reusability, we use the

weakest assumptions WA_i that accept the maximal set of input sequences to individual modules M_i without violating their local specifications φ_i. Thus, rule $\mathbf{R}^{\pi}$ can be applied for the compositional verification of concurrent systems in an incremental manner.

As shown in Algorithm 10, the verification task for checking whether the system $M_1|\cdots|M_n$ satisfies the global specification $\bigwedge_{i=1}^{n} \varphi_n$ can be decomposed into n BAG routines operating on a pair of module M_i and local specification φ_i. The BAG routines can run independently, each exploring just one module's dependency neighbourhood, and therefore amenable to parallelisation. If BAG$(M_i, \varphi_i, \mathcal{D})$ returns **false** for some i, then the module M_i can never meet the local specification φ_i. Whenever this happens, the algorithm returns **false** directly as this entails that the system cannot meet the global specification φ (Line 4). Otherwise, the algorithm exits with a positive answer (Line 7).

Algorithm 10. *Bounded Assume-Guarantee Reasoning*
Inputs: *System* $M_1|\cdots|M_n$ *and global specification* $\varphi_1 \wedge \cdots \wedge \varphi_n$

```
1:  D = {(M_i, M_j) | 1≤i≠j≤n, I_i ∩ X_j ≠ ∅};
2:  for each pair (M_i, φ_i) do
3:      if not BAG(M_i, φ_i, D) then
4:          return false;
5:      end if
6:  end for
7:  return true;

8:  function BAG(M_i, φ_i, D)
9:      Generate WA_i with respect to φ_i;
10:     d_i ← 1;
11:     D^{d_i} ← D;
12:     N_i^{d_i} ← {M_j | (M_i, M_j) ∈ D^{d_i}};
13:     while C_i^{d_i} ⊭ WA_i do
14:         D^{d_i+1} ← D^{d_i} ∪ {(M_i, M_j) | 1≤i≠j≤n and (M_i, M_j) ∈ D^{d_i} ∘ D};
15:         N_i^{d_i+1} ← {M_j | (M_i, M_j) ∈ D^{d_i+1}};
16:         if N_i^{d_i} ≠ N_i^{d_i+1} then
17:             d_i ← d_i + 1;
18:         else
19:             return false;
20:         end if
21:     end while
22:     return true;
23: end function
```

The function BAG implements bounded assume-guarantee reasoning with local

specifications. In each routine BAG$(M_i, \varphi_i, \mathcal{D})$ called by Algorithm 10, the weakest assumption WA_i is used for checking an increasing number of modules in the while-loop (Line 13). Since the number of modules is finite, this routine will terminate: either the assumption WA_i is guaranteed (Line 22), or all the modules that M_i reacts with have been checked (Line 19). Recall that $\mathcal{C}_i^{d_i}$ is the composition of the modules in $\mathcal{N}_i^{d_i}$.

We will apply the methodology above to a network control problem in Section 4.

4. Verifying stability of network protocols

One of our motivations for investigating assume-guarantee reasoning is to broaden the range of applications in the area of network control. We particularly expect 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 previous work [29] for more details about the system and the property we consider.

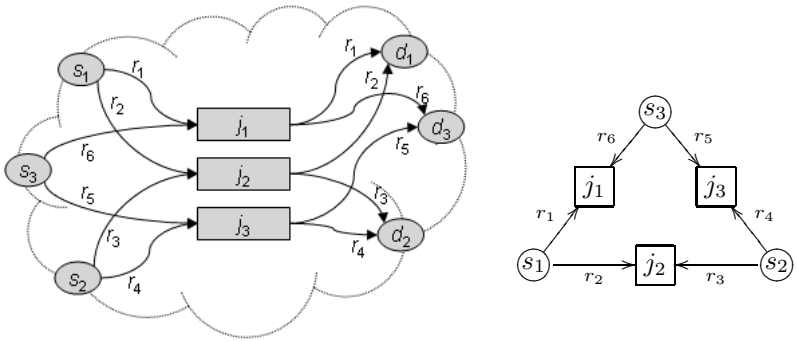
4.1. *Multi-path congestion control*

This subsection briefly presents an optimisation formulation of a congestion control problem, and follows the lines presented in [29]. We imagine 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 previously provisioned, and a source can split its traffic over these routes. Each route uses a number of links or, more generally, resources, each of which has a finite capacity constraint. We formalise this as follows.

Assume a network with a finite set S of sources and a finite set J of resources. Let R be a set of routes, each identifying a non-empty subset of resources. Each route connects only one source with its pre-defined destination. Let $r \in s$ denote that source s can transmit along route r and $s(r)$ be the unique source s such that $r \in s$. For example, in the network shown in Fig. 4(a), $S = \{s_1, s_2, s_3\}$, $J = \{j_1, j_2, j_3\}$, $R = \{r_1, \cdots, r_6\}$, and each source s_i $(1 \leq i \leq 3)$ transmits data to its destination d_i along two routes r_{2i-1} and r_{2i}. Fig. 4(b) presents the resource topology of the network, in which each resource is shared by two routes (i.e., $j_1 \in r_1, j_1 \in r_6$ and $j_i \in r_{2(i-1)}, j_i \in r_{2i-1}$ for $i = 2, 3$).

Let x_r be the flow rate on route r and C_j be the capacity of resource j. It is convenient to introduce vector notations for the flows and capacity constraints. Let $\vec{x} = (x_r, r \in R)$ and $C = (C_j, j \in J)$. Define the resource matrix $A = (A_{jr}, j \in J, r \in R)$ such that $A_{jr} = 1$ if $j \in r$ and $A_{jr} = 0$ otherwise.

The multi-path congestion control problem is to find an assignment of flows $\vec{x}$ which maximises the overall utility of the network such that no resource is congested. We assume that the utility U_s experienced by each source s depends on the total



(a) Network Topology

(b) Resource Topology

Fig. 4. A Communication Network

flow x_s sent over all the routes available to it, i.e., $x_s = \sum_{r \in s} x_r$, and that the overall utility of the network can be expressed as a sum of utilities of all the sources. These assumptions are standard in the networking literature, and allow the multi-path congestion control problem to be specified as the following optimisation problem:

$$\max \sum_{s \in S} U_s(x_s) \text{ subject to } A\vec{x} \leq C, \vec{x} \geq 0 \tag{2}$$

The utility functions U_s are strictly increasing and concave in their argument. The inequalities apply component-wise on the vectors $\vec{x}$ and C.

Congestion control protocols typically feature feedback signals that oblige the sources to reduce their flow as the load on the resources approaches capacity limits. These signals carry the interpretation of a price associated with congestion, and can be identified with the dual variables (Lagrange multipliers) in the optimisation formulation (2).

As a specific example, and for tractability, we devise a discrete version of the fluid-flow congestion control algorithm proposed in [12]. We allocate a utility function $U_s(x_s) = \alpha_s ln(x_s)$ for each source s. This is a common choice in the networking literature as it leads to a widely accepted notion of fairness in the equilibrium allocation of flows. For the resources we allocate price functions that increase linearly with load (with coefficient β_j for resource j). Substituting these choices into Equation (1) in [12] and taking discrete time steps gives, for each source s and route r available to s, a flow rate x_r subject to the following equation:

$$x'_r = \left[x_r + \kappa_r x_r \left(1 - \frac{x_{s(r)}}{\alpha_{s(r)}} \sum_{j \in r} \beta_j x_j \right) \right]_D \tag{3}$$

where the κ_r is a constant that determines how rapidly the path flows adjust to prices and x_j is the aggregate flow rate at resource j (i.e., $x_j = \sum_{j \in r} x_r$). In this

discrete version we have used the notation that, if $\bot_D$ and $\top_D$ are the minimal and the maximal value of domain D, respectively, then

$$[x]_D = \begin{cases} \bot_D & x \leq \bot_D \\ \top_D & x \geq \top_D \\ x & \text{otherwise} \end{cases}$$

The full motivation and derivation of the continuous version in [12] draws heavily on the interpretation of optimisation theory in a congestion control setting. Here we only mention that the derivative $\frac{\alpha_{s(r)}}{x_{s(r)}}$ of the utility function is the price source $s(r)$ is willing to pay to send flow $x_{s(r)}$. If this is equal to the sum $\sum_{j \in r} \beta_j x_j$ of the costs of the resources for route r, then the term in parentheses in Equation (3) becomes zero and the flow on route r is in equilibrium.

Each source s_i is modelled as an individual module M. The set of state variables of source s is defined as

$$X_s = \{x_r \mid r \in s\}.$$

The set of input variables for source s is

$$I_s = \{x_r \mid r \notin s \text{ and there exists } j \in J, r' \in s \text{ such that } j \in r \text{ and } j \in r'\},$$

i.e., I_s contains the flow variables x_r associated with all the routes sharing resources with source s. Equation (3) defines the transition relation for the source modules. 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). The algorithm presented in [12] assumes that these sources act synchronously. Under this assumption, the authors proved the stability of the algorithm. 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. In the following we consider all possible initial states of each module. This will enable us to evaluate the behaviour of the congestion control algorithm under any possible initial network configuration.

4.2. *Stability*

System stability is a key property of interest when reasoning about distributed congestion control systems. A system is said to be *stable* if it equilibrates at certain network-wide flow configuration, i.e., $x'_r = x_r$ for every route r. Let s_i range over all the sources. Then, the CLTL($\langle D, = \rangle$) formula

$$\mathtt{FG} \bigwedge_{s_i} \Big(\bigwedge_{r \in s_i} x'_r = x_r \Big) \tag{4}$$

represents system stability, i.e., the fact that all the flows will eventually reach a permanent value.

Lagrangian decomposition techniques reduce system stability onto individual modules [30]. A distributed source is *stable* if certain permanent flow configuration is reached on all the routes using the resources consumed by the source. The local stability for source s_i is represented by the following CLTL($\langle D, =\rangle$)formula

$$\texttt{FG}\,(\bigwedge_{x_r \in X_{s_i} \cup I_{s_i}} x'_r = x_r) \tag{5}$$

Then, to study the stability of the system, we instantiate rule $\mathbf{R}^{\pi}$ as rule **SS** (for system stability) below:

$$\textbf{SS}\ \frac{\begin{array}{c}\forall i : 1 \leq i \leq n, \quad M_i | A_i \ \vDash \texttt{FG} \bigwedge_{x_r \in X_{s_i} \cup I_{s_i}} x'_r = x_r \\ \exists d_i : 1 \leq d_i \leq \pi,\ \mathcal{C}_i^{d_i} \vDash A_i\end{array}}{M_1 | \cdots | M_n \ \vDash \bigwedge_{s_i} \texttt{FG} \bigwedge_{x_r \in X_{s_i} \cup I_{s_i}} x'_r = x_r}$$

where source s_i is represented as module M_i.

Observe that, given the commutativity of the Boolean connective '$\wedge$' with the universal temporal modalities, the conjunction of the local specifications (5) for all the sources is equivalent to the global specification (4) as we have $\bigcup_{s_i}(X_{s_i} \cup I_{s_i}) = \bigcup_{s_i}\{x_r \mid r \in s_i\}$. Therefore, the global specification (4) can be examined by reasoning about each individual sources under rule **SS**.

Given the above, our approach can take advantage of the inherent compositional structure of the congestion control system and its stability property. This can result in more efficient assumptions for local assume-guarantee reasoning. There are correspondences between our approach and [31] where a method to propagate global specifications into individual modules for compositional reasoning was put forward. Indeed, [31] supports the choice of auxiliary assertions over process interfaces. By doing so one can decompose the task of verifying the original specification $\texttt{G}(\bigwedge_{i=1}^{n} \varphi_i)$ into subtasks of checking whether $\bigwedge_{j=1, j\neq i}^{n} \varphi_j$ could constrain φ_i under those assertions along any computation of each module M_i. But the rule proposed in [31] is tuned specifically for synchronous systems, thus cannot be directly applied to this case study. Besides, the specification concerned in each subtask as many variables as contained in the original specification. Hence, the rule proposed in [31] would still suffer from the scalability issues observed in this paper.

4.3. *Stability assumptions*

By rule **SS**, the assumption A_i for module M_i is such that $M_i|A_i$ satisfies the local specification (5). Thus, assumption A_i concerns only 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 stabilise at a certain configuration on $X_i \cup I_i$.

Conversely, under rules SYM and ASYM, the assumption A_i has to include all the variables in X. To meet the global specification (4) a local stable state on $X_i \cup I_i$

would have to be extended to a stable global state on X. 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 $X_i \cup I_i$, the assumption A_i has to cover all the corresponding $|D|^{|X \backslash (X_i \cup I_i)|}$ stable global states. This redundancy is avoided under rule **SS** by generating assumption A_i with respect to the local specification (5).

In the following we construct assumptions analytically.

Definition 11 (Stability Assumption) *For a module*

$$M_i = (X_i, I_i, Q_{M_i}, T_{M_i}, \lambda_{M_i}, q_{0_{M_i}})$$

the assumption A_i can be constructed as the 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}, λ_{A_i} and $q_{0_{A_i}}$ are, respectively, the minimal sets of non-accepting states, accepting states, transitions, the labelling function, and the initial state defined as follows.

(1) $E_{A_i} = \{p^\alpha \mid \alpha \in D^{I_i}\}$, *i.e., for each valuation α on I_i, there exists one and only one state $p^\alpha \in E_{A_i}$; for each $p^\alpha \in E_{A_i}$, $\lambda_{A_i}(p^\alpha) = \alpha$;*

(2) $F_{A_i} = \{p^\alpha_q \mid q \xrightarrow{\alpha}_{M_i} q\}$, *i.e., for each transition $q \xrightarrow{\alpha}_{M_i} q$ of module M_i, there exists one and only one state $p^\alpha_q \in F_{A_i}$; for each $p^\alpha_q \in F_{A_i}$, $\lambda_{A_i}(p^\alpha_q) = \alpha$;*

(3) $T_{A_i} = T_{A_{i_0}} \cup T_{A_{i_1}} \cup T_{A_{i_2}}$, *where*

- *(a)* $T_{A_{i_0}} = \{p^\alpha \xrightarrow{\lambda_{M_i}(q)}_{A_i} p^{\alpha'} \mid p^\alpha, p^{\alpha'} \in E_{A_i}, q \xrightarrow{\alpha}_{M_i} q'\}$;
- *(b)* $T_{A_{i_1}} = \{p^\alpha_q \xrightarrow{\lambda_{M_i}(q)}_{A_i} p^\alpha_q \mid p^\alpha_q \in F_{A_i}, q \xrightarrow{\alpha}_{M_i} q\}$;
- *(c)* $T_{A_{i_2}} = \{p^\alpha \xrightarrow{\lambda_{M_i}(q)}_{A_i} p^\alpha_q \mid p^\alpha \in E_{A_i}, p^\alpha_q \in F_{A_i}, q \xrightarrow{\alpha}_{M_i} q\}$;

(4) $q_{0A_i} = p^{\alpha_0} \in E_{A_i}$ *is the initial state where α_0 is the given initial configuration on I_i.*

Intuitively, E_{A_i} records all possible inputs to module M_i as the non-accepting states of assumption A_i; $T_{A_{i_0}}$ traces the state changes of module M_i as the transitions of assumption A_i; F_{A_i} encodes the accepting states of assumption A_i to characterise all the configurations on $X_i \cup I_i$ where $M_i|A_i$ can possibly stabilise. Each self-loop transition $q \xrightarrow{\alpha}_{M_i} q$ contributes to an accepting state $p^\alpha_q \in F_{A_i}$ with two additional transitions leading to it(in $T_{A_{i_1}}$ and $T_{A_{i_2}}$, respectively), as shown in Fig. 5. Module M_i at state q will remain at this state under constant input α, which is exactly what the local specification (5) expects.

Thus, by means of the above, we can construct an assumption A_i for module M_i based on the module itself *regardless of the underlying topology*. Theorem 12 shows that the assumption A_i is appropriate for rule **SS**.

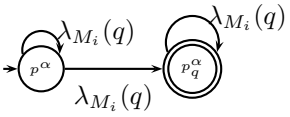


Fig. 5. Büchi Accepting State in Assumptions

Theorem 12. *Given a module M_i, the assumption A_i defined in Definition 11 is the weakest assumption with respect to the local specification* (5)*.*

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^α_q), where $q \in Q_{M_i}$ admits a self-loop transition under input α. Correspondingly, the infinite word accepted through such an accepting trace will terminate with an infinite loop of the valuation on $\lambda_{M_i}(q) \cup \lambda_{A_i}(p^\alpha_q)$. Therefore, $M_i|A_i$ satisfies the local specification (5).

We then prove by contradiction that assumption A_i is the weakest assumption with respect to the local specification (5). 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 (5), but there exists an infinite word $\theta = \alpha_0\alpha_1 \ldots \in \mathcal{L}(A'_i)$ that is not accepted by A_i. Then, by this hypothesis and the condition 2) in Definition 11, 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, there does not exist a transition $p^{\alpha_k} \xrightarrow{\rho}_{A_i} p^{\alpha_{k+1}}$ in A_i. Hence, by the condition 3a) in Definition 11, no transition $q \xrightarrow{\alpha_k}_{M_i} q'$ exists for any states $q, q' \in Q_{M_i}$. This conflicts with the hypothesis, which implies $\theta \in \mathcal{I}(M_i)$. □

By removing the condition 4) for A_i in Definition 11, we can 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.

Remark 13. *It can easily be seen that the time complexity of this construction method 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.*

5. Evaluation

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 the stability of a network of bounded degree irrespective of the number of sources and their initial flow configurations.

For the purposes of experiments, we consider the simple topology shown in Fig. 6, 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 module with an initial configuration $(u, v) \in D^2$. The transitions of $M_{u,v}$ are defined by Equation (3) with $\alpha_s = 36\beta_j, \kappa_r = 0.2$ for each source s, resource j, and route r. Then, no matter how many sources a network may consist of, each source is of the general form $M_{u,v}$.

Let $A_{u,v}$ be the super assumption generated by Definition 11 for module $M_{u,v}$. We start by checking whether the composition of any two possible direct neighbour modules can guarantee these assumptions. This amounts to check whether

$$M_{u_1,v_1'} | M_{u_1',v_1} \models A_{u_0,v_0} \tag{6}$$

for any initial configuration $(u_0, v_0, u_1, v_1, u_1', v_1') \in D^6$.

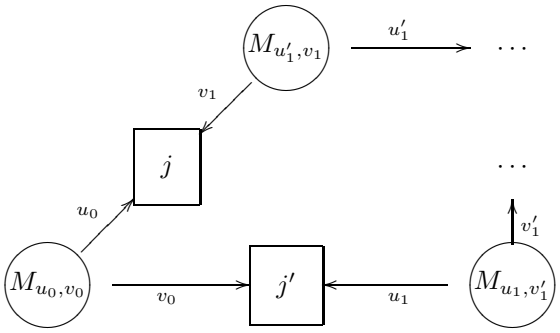


Fig. 6. Resource Topology

For the domain $D = [1, 6]$ there are $6^6 (= 46656)$ instances of Equation (6) need to be verified. These checks are done through 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 tool GOAL [28] 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 *#st.*) and the number of transitions (in Columns *#trans.*), and the time used in seconds for complementing each assumption A_{u_0,v_0} (in Columns *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 $A^{\varphi}_{u_0,v_0}$, generated with respect to the global specification (4), and the time used in seconds for complementing it. The symbol '-' means that the tool did not return a result within 10 hours. All experiments were ran on a Linux 2.6.18 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 are certainly achievable.

It can be seen that the size of assumptions for each module M_{u_0,v_0} is greatly reduced under rule **SS**. On average each assumption A_{u_0,v_0} is reduced by a factor

Table 1. Experimental Results for Computing Assumptions

u_0	v_0	$A^{\varphi}_{u_0,v_0}$			A_{u_0,v_0}			coA_{u_0,v_0}	
		#st.	#trans.	time	#st.	#trans.	time	#st.	#trans.
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

of 36 times in the number of states and a factor of 569 in the number of transitions compared with the corresponding assumption $A^{\varphi}_{u_0,v_0}$. 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 in 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 $coA^{\varphi}_{u_0,v_0}$ could be computed by the tool. Considering that simplifying a Büchi automaton is very time-consuming, we conclude that our approach is significantly more efficient than that of applying the general assume-guarantee rules with simplified version of assumptions $A^{\varphi}_{u_0,v_0}$.

Equation (6) was verified in our experiments for all the valuations of the initial configuration 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 of

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 $M_{\vec{u}}$, where vector $\vec{u}$ ranges over $\bigcup_{k=1}^{m} D^k$. Then, reasoning with any topology of bounded degree η will amount to check whether

$$M_{\vec{u}_1} | \cdots | M_{\vec{u}_l} \ \vDash \ A_{\vec{u}_0} \tag{7}$$

for any $1 \leq l \leq \eta$ and $\vec{u}_i \in \bigcup_{k=1}^{m} D^k (i = 1, \ldots, l)$. This is particularly appealing to us as previous results in the literature on verification of congestion control models (e.g., [32, 29]) apply only to fixed network topologies.

6. Conclusions and Future Work

This paper presents a methodology for assume-guarantee reasoning for global specifications that consist of conjunctions of local specifications. The methodology described is both sound and complete for local specifications, yet it can be applied to draw conclusions on global specifications. Thus, a verification task on a concurrent system can be decomposed onto individual modules and their 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. Our case study demonstrated that there are scenarios of interest where only a limited number of neighbours need to be considered. In general, however, we cannot provide a bound as to how deeply the neighbourhood dependency needs to be considered. This is in line with our intuition as there will be corner cases where the whole neighbourhood needs to be considered.

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 bounded 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 can be applied to.

In terms of future work, we note that automated learning algorithms have been proposed to generate assumptions automatically [4, 5, 6, 16, 10, 11, 17, 7, 18]. We intend to extend these to adapt Algorithm 10 to support generating local assumptions semi-automatically for each individual module. For instance, candidate assumptions A_i could in principle be generated through learning with respect to φ_i at Line 8. The incremental guarantee checking can still be applied, but the exit at Line 14 needs to be replaced by a case analysis on a counterexample w to $\mathcal{C}_i^{d_i} \vDash A_i$:

- If w presents a context in which φ_i can hold but A_i is not weak enough to incorporate it, then A_i is to be expanded with w for another round of checking;
- If w suggests a context in which φ_i could possibly be violated, then model M_i fails to meet the local specification φ_i indeed.

The difficulty to be faced in this direction is to overcome the present limitations in the automatic generation of assumptions by learning algorithms.

Acknowledgments

The authors are grateful to the anonymous referees for their valuable comments on an earlier version of this paper. This research was funded by BT Innovate through a grant to University College London which supported the corresponding author between 2007 and 2010. The corresponding author was then supported by the National Natural Science Foundation of China (NSFC) under Grant No. 61100069, ANR-NSFC under Grant No. 61161130530, and the National Science and Technology Major Project of China under Grant No. 2012ZX01039-004.

References

[1] J. Misra, K. M. Chandy, Proof of networks of processes, IEEE Transactions on Software Engineering SE-7 (4) (1981) 417–426.

[2] O. Grumberg, D. E. Long, Model checking and modular verification, ACM Transactions on Programming Languages and Systems 16 (3) (1994) 843–871.

[3] S. Berezin, S. V. A. Campos, E. M. Clarke, Compositional reasoning in model checking, in: Revised Lectures from the International Symposium on Compositionality: The Significant Difference (COMPOS'97), Lecture Notes in Computer Science 1536, Springer-Verlag, 1998, pp. 81–102.

[4] D. Giannakopoulou, C. S. Păsăreanu, H. Barringer, Assumption generation for software component verification, in: Proc. 17th IEEE International Conference on Automated Software Engineering (ASE'02), IEEE Computer Society, 2002, pp. 3–12.

[5] H. Barringer, D. Giannakopoulou, C. S. Păsăreanu, Proof rules for automated compositional verification through learning, in: Proc. 2003 Workshop on Specification and Verification of Component-Based Systems (SAVCBS'03), Helsinki, Finland, 2003, pp. 14–21.

[6] J. Cobleigh, D. Giannakopoulou, C. Păsăreanu, Learning assumptions for compositional verification, in: Proc. 9th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'03), Lecture Notes in Computer Science 2619, Springer-Verlag, 2003, pp. 331–346.

[7] A. Farzan, Y.-F. Chen, E. M. Clarke, Y.-K. Tsay, B.-Y. Wang, 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), Lecture Notes in Computer Science 4963, Springer-Verlag, 2008, pp. 2–17.

[8] T. A. Henzinger, S. Qadeer, S. K. Rajamani, You assume, we guarantee: Methodology and case studies, in: Proc. 10th International Conference on Computer Aided Verification (CAV'98), Lecture Notes in Computer Science 1427, Springer-Verlag, 1998, pp. 440–451.

[9] O. Kupferman, M. Y. Vardi, An automata-theoretic approach to modular model checking, ACM Transactions on Programming Languages and Systems 22 (1) (2000) 87–128.

[10] M. Gheorghiu Bobaru, C. S. Păsăreanu, D. Giannakopoulou, Automated assume-guarantee reasoning by abstraction refinement, in: Proc. 20th International Confer-

ence on Computer Aided Verification (CAV'08), Lecture Notes in Computer Science 5123, Springer-Verlag, 2008, pp. 135–148.

[11] C. S. Păsăreanu, D. Giannakopoulou, M. G. Bobaru, J. M. Cobleigh, H. Barringer, Learning to divide and conquer: applying the L* algorithm to automate assume-guarantee reasoning, Formal Methods in System Design 32 (3) (2008) 175–205.

[12] F. Kelly, T. Voice, Stability of end-to-end algorithms for joint routing and rate control, ACM SIGCOMM Computer Communication Review 35 (2) (2005) 5–12.

[13] N. Francez, A. Pnueli, A proof method for cyclic programs, Acta Informatica 9 (2) (1978) 133–157.

[14] C. B. Jones, Tentative steps toward a development method for interfering programs, ACM Transactions on Programming Languages and Systems 5 (4) (1983) 596–619.

[15] P. Maier, 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), Lecture Notes in Computer Science 2620, Springer-Verlag, 2003, pp. 343–357.

[16] W. Nam, R. Alur, Learning-based symbolic assume-guarantee reasoning with automatic decomposition, in: Proc. 4th International Symposium on Automated Technology for Verification and Analysis (ATVA'06), Lecture Notes in Computer Science 4218, Springer-Verlag, 2006, pp. 170–185.

[17] W. Nam, P. Madhusudan, R. Alur, Automatic symbolic compositional verification by learning assumptions, Formal Methods in System Design 32 (3) (2008) 207–234.

[18] S. Chaki, A. Gurfinkel, Automated assume-guarantee reasoning for omega-regular systems and specifications, Innovations in Systems and Software Engineering 7 (2) (2011) 131–139. doi:10.1007/s11334-011-0148-1.

[19] M. Abadi, L. Lamport, Conjoining specifications, ACM Transactions on Programming Languages and Systems 17 (3) (1995) 507–535.

[20] A. Lomuscio, B. Strulo, N. Walker, P. Wu, Assume-guarantee reasoning with local specifications, in: Proceedings of the 12th International Conference on Formal Engineering Methods (ICFEM'10), Vol. 6447 of Lecture Notes in Computer Science, Springer Berlin / Heidelberg, 2010, pp. 204–219.

[21] R. Alur, T. A. Henzinger, Reactive modules, Formal Methods in System Design 15 (1999) 7–48.

[22] R. Alur, T. A. Henzinger, F. Mang, S. Qadeer, S. K. Rajamani, S. Tasiran, MOCHA: Modularity in model checking, in: Proc. 10th International Conference on Computer-aided Verification (CAV'98), Lecture Notes in Computer Science 1427, Springer-Verlag, 1998, pp. 521–525.

[23] A. Cimatti, E. M. Clarke, E. Giunchiglia, F. Giunchiglia, M. Pistore, M. Roveri, R. Sebastiani, A. Tacchella., NuSMV 2: An opensource tool for symbolic model checking, in: Proc. 14th International Conference on Computer-Aided Verification (CAV'02), Lecture Notes in Computer Science 2404, Springer-Verlag, 2002, pp. 241–268.

[24] S. Demri, Linear-time temporal logics with presburger constraints: an overview, Journal of Applied Non-Classical Logics 16 (3-4) (2006) 311–348.

[25] A. Pnueli, The temporal logic of programs, in: Proceedings of the 18th Annual Symposium on Foundations of Computer Science, IEEE Computer Society, Washington, DC, USA, 1977, pp. 46–57. doi:10.1109/SFCS.1977.32.

[26] J. Edmund M. Clarke, O. Grumberg, D. A. Peled, Model Checking, MIT Press, 1999.

[27] M. Huth, M. Ryan, Logic in Computer Science: Modelling and Reasoning about Systems, Cambridge University Press, New York, NY, USA, 2004.

[28] Y.-K. Tsay, Y.-F. Chen, M.-H. Tsai, K.-N. Wu, W.-C. Chan, 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), Lecture Notes in Computer Science 4424, Springer-Verlag, 2007, pp. 466–471.

[29] A. Lomuscio, B. Strulo, N. G. Walker, P. Wu, Model checking optimisation based congestion control algorithms, Fundamenta Informaticae 102 (2010) 77–96.

[30] S. H. Low, D. E. Lapsley, Optimization flow control, I: basic algorithm and convergence, IEEE/ACM Transactions on Networking 7 (6) (1999) 861–874.

[31] K. S. Namjoshi, R. J. Trefler, On the completeness of compositional reasoning methods, ACM Transactions on Computational Logic (TOCL) 11 (2010) 16:1–16:22.

[32] C. Yuen, W. Tjioe, 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), Lecture Notes in Computer Science 2057, Springer-Verlag, 2001, pp. 272–287.