Assume-Guarantee Verification for Distributed Systems with Local Specifications

Alessio Lomuscio  
Department of Computing  
Imperial College London  
London, UK

Ben Strulo, Nigel Walker  
BT Innovate  
Adastral Park  
Ipswich, UK

Peng Wu  
Department of Computer Science  
University College London  
London, UK

19 February 2010
Contents

1 Introduction 1

2 Assume-Guarantee Reasoning 3
   2.1 Modules ........................................... 3
   2.2 A Simple Assume-Guarantee Rule .................... 6

3 Bounded Assume-Guarantee Reasoning 8

4 Case Study 10
   4.1 Multi-Path Congestion Control ..................... 10
   4.2 Stability .......................................... 11
   4.3 Computing Assumptions ............................ 12
   4.4 Experiments ....................................... 13

5 Conclusions 14

A Tentative Rule $R_0$ 16

List of Tables

1 Experimental Results for Computing Assumptions ........ 14

List of Figures

1 Büchi Accepting State in Assumptions ................... 12
Assume-Guarantee Verification for Distributed Systems with Local Specifications

Alessio Lomuscio
Department of Computing, Imperial College London, UK

Ben Strulo and Nigel Walker
BT Innovate, Adastral Park, UK

Peng Wu
Department of Computer Science, University College London, UK

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 about global specifications of distributed systems. 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 a sound 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 is one of known approaches to alleviate state explosion in model checking. To verify whether a system satisfies certain property, this approach identifies adequate environment assumptions for individual modules of the system. General assume-guarantee rules have been proposed for safety and liveness properties over the last decade [6, 2, 3, 4]. To name a few, symmetric rules check whether these assumptions may collectively violate the property; while asymmetric rules check whether the assumption for some module can be guaranteed by the rest of modules.

The size of assumptions then becomes a major concern for assume-guarantee reasoning, because large assumptions can still cause scalability issues. The motivation of this paper is to investigate possible ways to reduce the size of assumptions and to reuse assumptions for verifying distributed systems.

Typically a module in a distributed system reacts directly only with a few other modules in its environment; with the rest of the system it reacts only indirectly through intermediate modules. However, the assumptions generated under the general assume-guarantee rules do not exploit this neighbourhood dependency, and so can include much redundant information, such as the expected behaviour of those the module reacts indirectly with. When new modules are
added to the system these can contribute to even more redundant information in the assumptions.

For a system property that can be represented as the conjunction of local specifications on individual modules, these scalability issues can be avoided by generating assumptions with respect to local specifications. A local specification can be regarded as a reflection of the system property on an individual module and its environment. Thus, an assumption generated in this way concerns only modules that constitute the module’s environment.

The main contribution of this paper 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 rule $R_1$ that we prove to be sound for local specifications. Through a counterexample, we show that this simple rule is not complete because it only exploits the direct dependency between modules.

We then extend rule $R_1$ towards completeness. This leads to a bounded assume-guarantee rule $R^\pi$ that we prove to be sound and complete for local specifications. This rule triggers a bounded assume-guarantee reasoning approach, in which the dependency between modules is exploited incrementally.

Suppose the number of hops (or the distance) from one module to another is measured by the minimal number of intermediate modules, through which the former module can react indirectly with the latter one. The approach runs guarantee checking recursively. It first checks whether assumptions can be guaranteed by neighbour modules (which individual modules react directly with in zero hop). Then, in each round, the number of hops is increased by one so as to include the modules in one more hop for checking. Since the number of modules is finite in the system, the procedure will terminate eventually with two possible verdicts: the system satisfies the property if all assumptions are guaranteed; conversely, the system does not satisfy the property if, for some module, no assumption exists that can be guaranteed by the rest of modules.

Finally, we apply rule $R^\pi$ to verify the stability of an optimisation based congestion control system proposed by Kelly and Voice [7]. The optimisation approach allows a distributed solution for network congestion control. A distributed congestion control system is stable if the system equilibrates at certain network-wide flow configuration. This means that each source in the system reaches a stable flow configuration on the routes available to the source. We inherit the compositional structure of this dynamic system and analyse its stability by reasoning about individual sources for local stability. The case study shows that an instantiation of rule $R^\pi$ for system stability can be applied for reasoning about any number of sources, any initial flow configuration, and any topology of bounded degree.

Related Work. General assume-guarantee rules were proposed for safety properties with support of learning based assumption generation [6, 2, 3]. [10, 11] proposed a symbolic approach to learning-based assume-guarantee reasoning. [5, 12] proposed an alphabet refinement technique to reduce the size of assumptions. [4] extended the general assume-guarantee rules to liveness properties, based on the fact that the soundness and completeness of the assume-guarantee rules for safety properties remain intact for liveness properties ($\omega$-regular languages share the required closure properties of regular languages).
On observing possible redundancy behind the general assume-guarantee rules, we present in this paper a bounded assume-guarantee reasoning approach where assumptions are generated with respect to local specifications. The approach can be implemented using symbolic representation, and integrated with learning algorithms for automated assumption generation. On the other hand, those learning algorithms can also benefit from our approach by learning assumptions over local alphabets, instead of the global alphabet.

This paper also extends our previous work [8] by introducing assume-guarantee reasoning to handle the scalability issues, especially in case when asynchronous schemes are concerned for congestion control.

The rest of this paper is organised as follows. We present the simple assume-guarantee rule in Section 2 and prove its soundness and incompleteness. The bounded assume-guarantee rule is presented in Section 3 with proofs of its soundness and completeness. Section 4 illustrates the case study of our bounded assume-guarantee reasoning approach. The conclusions of this work are summarised in Section 5.

## 2 Assume-Guarantee Reasoning

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

### 2.1 Modules

Technically we adopt the basic notion of reactive module [1] to represent distributed 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 $Z$ of variables, let $D^Z$ be the set of all valuation functions on $Z$. For valuation $\rho: Z \rightarrow D$ and $Z' \subseteq Z$, $\rho|_{Z'}: Z' \rightarrow D$ is the restriction of $\rho$ to $Z'$, that is, $(\rho|_{Z'})(x) = \rho(x)$ for any $x \in Z'$.

For valuations $\rho_1: Z_1 \rightarrow D$ and $\rho_2: Z_2 \rightarrow D$, $\rho_1$ and $\rho_2$ are compatible, denoted $\rho_1 \sim \rho_2$, if for any $x \in Z_1 \cap Z_2$, $\rho_1(x) = \rho_2(x)$. For compatible valuations $\rho_1$ and $\rho_2$, $\rho_1 \cup \rho_2 : X_1 \cup X_2 \rightarrow D$ is the extension of $\rho_1$ and $\rho_2$ to $X_1 \cup X_2$, that is, $(\rho_1 \cup \rho_2)(x) = \rho_1(x)$ for $x \in Z_1 - Z_2$, $(\rho_1 \cup \rho_2)(x) = \rho_2(x)$ for $x \in Z_2 - Z_1$ and $(\rho_1 \cup \rho_2)(x) = \rho_1(x) = \rho_2(x)$ for $x \in Z_1 \cap Z_2$.

**Definition 2.1 (Module).** A module $M$ is a tuple $(X, I, Q, T, \lambda, q_0)$, 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 and $X \cap I = \emptyset$;
- $Q$ is a finite set of states;
- $\lambda: Q \rightarrow D^X$ labels each state $q \in Q$ with a valuation $\lambda(q): X \rightarrow D$;
- $T \subseteq Q \times D^I \times Q$ 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 $\alpha: I \rightarrow D$;

$\emptyset$
• $q_0 \in Q$ is the initial state.

An infinite trace $\sigma$ of module $M$ is an infinite sequence $q_0\alpha_0q_1\alpha_1\cdots$ such that $q_i \xrightarrow{\alpha_j} q_{i+1}$ for any $i \geq 0$. Let $\inf(\sigma)$ be the set of all the states that are visited infinitely often in $\sigma$.

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

For $Z \subseteq X$, a stuttering projection of $w$ on $Z$, denoted $w|_Z$, is an infinite word $\rho_0\rho_1\cdots$, where there exists $0 = j_0 < j_1 < \cdots$ such that $\rho_i = \rho_{j_i}|_Z = \rho_{j_{i+1}}|_Z = \cdots = \rho_{j_{i+1}+1}|_Z$ for any $i \geq 0$. As a special case of stuttering projection, the restriction of $w$ on $Z$, denoted $w|_Z$, is the infinite word $\rho_0\rho_1\cdots$, where $\rho_i = \rho|_Z$ 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 $\theta = \alpha_0\alpha_1\cdots$ on the input alphabet $D^I$ is admitted by $M$ if there exists an infinite trace $q_0\alpha_0q_1\alpha_1\cdots$ such that $q_i \in Q$ for any $i \geq 0$. Let $I(M)$ be the set of input words admitted by $M$. Specially, module $M$ is closed if $I = \emptyset$.

We then define the composition operator "\|$" for modules in distributed systems. We choose the notion of composition that explicitly supports asynchrony, because in distributed systems, asynchrony typically arises externally from network communication or scheduling; while general reactive module languages support synchronous composition but leave asynchrony as internal choices of modules themselves.

**Definition 2.2 (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 these two modules, denoted $M_1 \mid M_2$, is a module $(X, I, Q, T, \lambda, q_0)$, where

- $X = X_1 \cup X_2$;
- $I = (I_1 \cup I_2) - X$;
- $Q \subseteq Q_1 \times Q_2$ and $\lambda_1(q_1) \sim \lambda_2(q_2)$ for each state $(q_1, q_2) \in Q$;
- $\lambda : Q \rightarrow D^X$ labels each state $(q_1, q_2) \in Q$ with the valuation $\lambda_1(q_1) \cup \lambda_2(q_2)$;
- $T$ is the smallest transition relation derived by the following composition rules:

  | SYN                  | $q_1 \xrightarrow{\alpha_1} q_1' \quad q_2 \xrightarrow{\alpha_2} q_2' \quad (q_1, q_2) \xrightarrow{\alpha} (q_1', q_2')$ |
  | ASYN_L               | $q_1 \xrightarrow{\alpha_1} q_1' \quad q_2 \xrightarrow{\alpha_2} q_2' \quad (q_1, q_2) \xrightarrow{\alpha} (q_1', q_2')$ |
  | ASYN_R               | $q_1 \xrightarrow{\alpha_1} q_1' \quad q_2 \xrightarrow{\alpha_2} q_2' \quad (q_1, q_2) \xrightarrow{\alpha} (q_1', q_2')$ |

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


\begin{itemize}
  \item \(q_0 = (q_{01}, q_{02}) \in Q\).
\end{itemize}

By rule \textsc{asy} \textsc{L}, (or \textsc{asy} \textsc{R}), only \(M_1\) (or \(M_2\)) evolves; while by rule \textsc{syn}, both \(M_1\) and \(M_2\) evolve simultaneously. For multiple modules, these composition rules can make only one, some or all modules evolve collectively. \(^1\)

Thus, a \textit{closed} distributed 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 \textit{global} alphabet for the system \(M_1|\cdots|M_n\).

Assumptions can then be defined as extended modules with accepting states. Because the property concerned by our case study is represented as a liveness property, we adopt the formalism of Büchi automaton in the definition of assumption. However, the assume-guarantee rules presented in this paper also apply to safety properties (for which assumptions are then defined as finite automata).

\textbf{Definition 2.3 (Assumption).} An assumption \(A\) is a tuple \((X, I, Q, T, \lambda, q_0, F)\), where \(X, I, Q, T, \lambda, q_0\) are the same as in Definition 2.1, and \(F \subseteq Q\) is a finite set of accepting states.

The terminology defined for modules also applies to assumptions. Especially, an infinite word \(\rho_0\rho_1\cdots\) on alphabet \(D^X\) is \textit{accepted} by \(A\) if there exists an infinite trace \(\sigma = q_0\alpha_0q_1\alpha_1\cdots\), referred to as an \textit{accepting trace}, such that \(\inf(\sigma) \cap F \neq \emptyset\) and \(\rho_i = \lambda(q_i)\) for any \(i \geq 0\). The \textit{language} accepted by \(A\), denoted \(L(A)\), is all the infinite words accepted by \(A\).

The definition of composition can be extended to assumptions, too. For assumptions \(A_1 = (X_1, I_1, Q_1, T_1, \lambda_1, q_{01}, F_1)\) and \(A_2 = (X_2, I_2, Q_2, T_2, \lambda_2, q_{02}, F_2)\), compositions \(M_1|A_2\) and \(A_1|A_2\) are modules \((X, I, Q, T, \lambda, q_0, F)\) and \((X, I, Q, T, \lambda, q_0, F')\), respectively, where \(X, I, Q, T, \lambda, q_0\) are the same as in Definition 2.2 and

\begin{itemize}
  \item \(F = \{(q_1, q_2) \in Q \mid q_2 \in F_2\}\);
  \item \(F' = \{(q_1, q_2) \in Q \mid q_1 \in F_1, q_2 \in F_2\}\).
\end{itemize}

This allows to extend the general assume-guarantee rules for local specifications (upon local alphabets).

Then, the following definition relates an assumption with modules that exclusively control the state variables concerned by the assumption.

\textbf{Definition 2.4 (Guarantee).} For the composition of \(k\) modules \(M_i = (X_i, I_i, Q_i, T_i, \lambda_i, q_{0i})\) and an assumption \(A = (X_A, I_A, Q_A, T_A, \lambda_A, q_{0A}, F_A)\) such that

\begin{itemize}
  \item \(X_i \cap X_j = \emptyset\) for any \(1 \leq i, j \leq k\);
  \item \(X_A \subseteq \bigcup_{i=1}^{k} X_i\),
\end{itemize}

suppose \(M_{i_1}, \ldots, M_{i_{k'}}, (1 \leq i_1, \ldots, i_{k'} \leq k)\) are all the \(k'\) modules such that \(X_A \cap X_{M_{i_j}} \neq \emptyset\) for \(1 \leq j \leq k'\), then \(M_1|\cdots|M_{k'}\) \textit{guarantees} \(A\), denoted \(M_1|\cdots|M_{k'} \models A\), if for any infinite word \(w\) derived by \(M_1|\cdots|M_{k'}\) and any stuttering projection

\(^1\)This is also why we introduce the notion of stuttering projection when certain part of state variables is specially concerned. Asynchronous composition does not enhance expressiveness. This composition operator can be implemented by general reactive module languages.
$w'$ of $w$ on $\bigcup_{j=1}^{k'} X_{M_{ij}}$ that can be derived by $M_{i_1}|\cdots|M_{i_{k'}}$, $w'|_{X_A}$ is accepted by $A$.

Specially, if $k'=k$, that is, $X_A \cap X_i \neq \emptyset$ for any $1 \leq i \leq k$, $M_1|\cdots|M_k \models A$ simply means that for any infinite word $w$ derived by $M_1|\cdots|M_k$, $w'|_{X_A}$ is accepted by $A$.

### 2.2 A Simple Assume-Guarantee Rule

The basic principle of assume-guarantee reasoning is to decompose a verification task on a system into sub-tasks on individual modules of the system. Herein, we also take into account possible compositional structures of system properties.

Consider the distributed system $M_1|\cdots|M_n$ and a global specification $\psi$ on $X$ that can be represented as the conjunction of local specifications $\phi_i$ on $X_i \cup I_i$ such that $\psi \iff \bigwedge_{i=1}^{n} \phi_i$. General assume-guarantee approach either generates assumptions for each module $M_i$ with respect to the global specification $\psi$ and then checks whether these assumptions may collectively violate it; or generates an assumption for some module $M_i$ with respect to the global specification $\psi$ and then checks whether the assumption can be guaranteed by the rest of modules. The former rules are known as symmetric rules; while the latter are asymmetric rules.

Thus, it is a common practice to generate assumptions with respect to global specifications. However, in distributed systems, individual modules control their state variables typically under inputs from a few other modules. Therefore,

- each assumption $A_i$ for module $M_i$ contains 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 with more modules, each assumption $A_i$ has to be modified to incorporate their state variables. Thus, assumptions for the existing modules can not be reused for verifying the extended system.

We propose to avoid these scalability issues by distributing each local specification $\phi_i$ onto its corresponding module $M_i$. For the general symmetric rules, this results in an tentative rule $R_0$ as follows:

$$R_0 \quad \frac{\forall 1 \leq i \leq n, M_i|A_i \models \phi_i}{M_1|\cdots|M_n \models \bigwedge_{i=1}^{n} \phi_i}$$

where each assumption $A_i = (I_i, X_i, Q'_i, T'_i, \lambda_i, q'_0_i, F_i)$ is to be generated with respect to the local specification $\phi_i$ (on $X_i \cup I_i$), and $coA_i$ is the complement of assumption $A_i$. 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 $X$) has to be concerned by assumption $A_i$.

However, as a side effect, assumption $A_i$ may admit more interactions with module $M_i$ than can be admitted by assumptions generated with respect to the global specification $\psi$, because variables in $X - (X_i \cup I_i)$ are not constrained by the local specification $\phi_i$. Therefore, the tentative rule $R_0$ does not preserve
soundness, though its completeness is not affected by the weaker assumptions. Appendix A presents an counterexample where rule $R_0$ fails.

For module $M_i$, suppose modules $M_{i_1}, \ldots, M_{i_k}$ ($k \geq 1$) are all the $k$ neighbour modules that control its input variables in $I_i$. So, $I_i \subseteq \bigcup_{j=1}^k X_{i_j}$ and $I_i \cap X_{i_j} \neq \emptyset$ for any $1 \leq j \leq k_i$. Informally, module $M_i$ reacts directly with these $k_i$ neighbour modules. Then, inspired by the general asymmetric rules, we present rule $R_1$ as follows:

$$\forall 1 \leq i \leq n, \quad M_i|A_i \models \varphi_i$$

$$\forall 1 \leq i \leq n, \quad M_{i_1} \cdots M_{i_k} \models A_i$$

$$M_1 \cdots M_n \models \bigwedge_{i=1}^n \varphi_i$$

Theorem 2.5 shows the soundness of rule $R_1$ for local specifications.

**Theorem 2.5 (Soundness).** If for any module $M_i$ ($1 \leq i \leq n$), there exists assumption $A_i$ such that $M_i|A_i \models \varphi_i$ and $M_{i_1} \cdots M_{i_k} \models A_i$, then $M_1 \cdots M_n \models \bigwedge_{i=1}^n \varphi_i$.

**Proof.** By contradiction. Consider an infinite word $w = \rho_0 \rho_1 \cdots$ on the global alphabet (that is, each $\rho_i$ is a valuation on $X$) that makes the conclusion fail on some $\varphi_j$ ($1 \leq j \leq n$). Then, since the state variables in $X_j$ is exclusively controlled by $M_{j}$, any stuttering projection $w|_{X \cup I_j}$ 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, by Definition 2.2, there exists a stuttering projection of $w$ on $\bigcup_{i=1}^k X_{i_j}$, denoted $w'$, that is derived by $M_{i_1} \cdots M_{i_k}$. Since $I_j \subseteq \bigcup_{i=1}^k X_{i_j}$ and $M_{i_1} \cdots M_{i_k} \models A_j$, $w'|_{I_j}$ is then accepted by $A_j$. This is a contradiction because $w'|_{I_j}$ is also a stuttering projection of $w$ on $I_j$.

Unfortunately, rule $R_1$ is not complete in general, because 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 to the neighbour modules. For example, consider a system consisting of the following four modules $M_i$ ($1 \leq i \leq 4$):

<table>
<thead>
<tr>
<th>$M_i$</th>
<th>$X_i$</th>
<th>$I_i$</th>
<th>Transition Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>$M_1$</td>
<td>${x_1}$</td>
<td>${x_2, x_3}$</td>
<td>$x'_1 = x_2 - x_3$</td>
</tr>
<tr>
<td>$M_2$</td>
<td>${x_2}$</td>
<td>${x_4}$</td>
<td>$x'_2 = x_2 - x_4$</td>
</tr>
<tr>
<td>$M_3$</td>
<td>${x_3}$</td>
<td>${x_4}$</td>
<td>$x'_3 = x_3 + x_4$</td>
</tr>
<tr>
<td>$M_4$</td>
<td>${x_4}$</td>
<td>${x_2, x_3}$</td>
<td>$x'_4 = \begin{cases} 1 &amp; x_2 &gt; x_3 \text{ and } x_4 &gt; 0 \ -1 &amp; x_2 &lt; x_3 \text{ and } x_4 &lt; 0 \ 0 &amp; \text{otherwise} \end{cases}$</td>
</tr>
</tbody>
</table>

and local specification

$$AFAG \bigwedge_{x \in X \cup I_i} (x' = x)$$
on each module $M_i$, where $x'$ is the next value of variable $x$.

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

$$M_1|M_2|M_3|M_4 \models 4 \bigwedge_{i=1}^4 AFAG \bigwedge_{x \in X \cup I_i} (x' = x)$$
This is because within the system $x_2$ and $x_3$ evolve by converging in a step size $x_4$, until $x_2$ and $x_3$ meet or just cross each other. Then, the system 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 $(x_2, x_3)$ can not lead $M_1$ to stabilising $x_1$, and so can not be accepted by any assumption $A_1$ that satisfies the premise $M_1|A_1 \models AFG_{x \in X_1 \cup I_1} (x' = x)$.

3 Bounded Assume-Guarantee Reasoning

In this section we extend the simple rule $R_1$ towards completeness by exploiting the neighbourhood dependency between modules incrementally. This results in a bounded rule $R^\pi$, which triggers a bounded assume-guarantee reasoning approach.

For the system $M_1|\cdots|M_n$, $D = \{(M_1, M_2) \mid X_2 \cap I_1 \neq \emptyset\}$ is the direct dependency relation between the modules. $(M_1, M_2) \in D$ means that $M_1$ depends on the inputs from (or reacts directly with) $M_2$. Then, the $k$-dependency relation $D^k$ is defined recursively as follows:

$$D^k = \begin{cases} D^{k-1} \cup (D^{k-1} \circ D) & k > 1 \\ D & k = 1 \end{cases}$$

where $D^{k-1} \circ D$ is the composition of $D^{k-1}$ and $D$.

For module $M_i$, let $C^k_i$ be the composition of all the modules $M$ except $M_i$ such that $(M_i, M) \in D^k$. Then, rule $R_1$ can be extend to a general rule $R_k$ as follows:

$$R_k \frac{\forall 1 \leq i \leq n, \quad M_i|A_i \models \varphi_i \qquad \forall 1 \leq i \leq n, \quad C^k_i \models A_i} {M_1|\cdots|M_n \models \bigwedge_{i=1}^n \varphi_i}$$

Informally, for each module $M_i$, rule $R_1$ only checks its neighbour modules; while rule $R_k$ checks all the modules in the range of $k - 1$ hops from module $M_i$. Similarly, it can be proved that rule $R_k$ is sound for any $k \geq 1$.

Theorem 3.1 (Soundness). Given $k \geq 1$, if for any module $M_i$ ($1 \leq i \leq n$), there exists assumption $A_i$ such that $M_i|A_i \models \varphi_i$ and $C^k_i \models A_i$, then $M_1|\cdots|M_n \models \bigwedge_{i=1}^n \varphi_i$.

Proof. By contradiction. Similar to Theorem 2.5.

For module $M_i$, if the modules in $k$ hops can guarantee assumption $A_i$, then such guarantee is preserved by the ones in $k + 1$ hops. This is because assumption $A_i$ has already been guaranteed regardless of the interactions with the additional modules. Based on this observation, Theorem 3.2 relates rule $R_k$ with rule $R_{k+1}$.

Theorem 3.2. For each module $M_i$, let $A_i$ be an assumption such that $M_i|A_i \models \varphi_i$. Then, $C^k_i \models A_i$ implies $C^{k+1}_i \models A_i$. 

8
Proof. Let $N^k_i$ be the set of all the modules that appear in $C^k_i$. Then, by definition of $D^k$, $N^k_i \subseteq N_i^{k+1}$. So, $I_i \subseteq \bigcup_{M_j \in N^k_i} X_j \subseteq \bigcup_{M_j \in N_i^{k+1}} X_j$. For any word $w$ derived by $C_i^{k+1}$, there exists a stuttering projection of $w$ on $\bigcup_{M_j \in N^k_i} X_j$, denoted $w'$, that can be derived by $C^k_i$. Since $C^k_i \models A_i$, $w' \mid_{I_i}$ would be accepted by $A_i$ for any such $w'$.

Since the system consists of a finite number of state variables, there exists the transitive dependency closure $D^\pi$ such that $D = D^\pi + 1$. Theorem 3.3 shows that rule $R_\pi$ is complete for local specifications.

**Theorem 3.3** (Completeness). Given a system $M_1 \cdots | M_n$ and a decomposable specification $\bigwedge_{i=1}^n \varphi_i$ such that $M_1 \cdots | M_n \models \bigwedge_{i=1}^n \varphi_i$. Suppose $D^\pi$ is the transition dependency closure of the system. Then, there exists assumption $A_i$ for each module $M_i$ such that $M_i \models A_i$ and $C^\pi_i \models A_i$.

Proof. By construction. Since $\bigwedge_{i=1}^n \varphi_i$ implies $\varphi_j$ for any $1 \leq j \leq n$. Thus, for each module $M_i$, $C^\pi_i$ could be extended as such assumption $A_i$ by appointing all states in $C^\pi_i$ as accepting states.

As a corollary of theorems 3.1, 3.2 and 3.3, rule $R_\pi$ could be reformed below as rule $R^\pi$, which is also sound and complete for local specifications.

\[
\begin{array}{c}
\forall 1 \leq i \leq n, \quad M_i \models A_i \models \varphi_i \\
\forall 1 \leq i \leq n, \exists 1 \leq k \leq \pi, \quad C^k_i \models A_i
\end{array}
\]

Thus, rule $R^\pi$ triggers an incremental way for compositional verification of distributed systems. For the system $M_1 \cdots | M_n$ and the global specification $\psi$ that is equivalent to the conjunction of $n$ local specifications $\varphi_i$ on $X_i \cup I_i$ ($1 \leq i \leq n$), the verification task for checking whether $M_1 \cdots | M_n \models \psi$ can be decomposed into $n$ parallel sub-tasks. For each module $M_i$,

- Firstly, generate an appropriate assumption $A_i$ with respect to the local specification $\varphi_i$;
- Then, check recursively whether assumption $A_i$ is guaranteed by the rest of modules. This starts by examining the neighbour modules that module $M_i$ react directly with in zero hop. Then, if all the modules in $k \geq 0$ hops can not guarantee assumption $A_i$, then the range of neighbourhood around module $M_i$ is deepen to $k+1$ hops for another round of guarantee checking. Since the number of modules is finite in the system, this procedure will terminate eventually in two cases: either assumption $A_i$ is guaranteed, or otherwise no further peripheral module exists that has not yet been examined.

Herein, assumption $A_i$ has to be an strong enough abstraction for module $M_i$ to make of its environment, not only in order to satisfy local specification $\varphi_i$, but also to be suitable for checking the modules in different ranges around module $M_i$. For the sake of generality and reusability, the preferable option is the weakest assumption that admits as many as possible sequences of inputs.
to module $M_i$ without violating the local specification $\varphi_i$. For module $M_i$, the weakest assumption $WA_i$ is an assumption such that $L(WA_i) \subseteq I(M_i)$ and

1. $M_i|WA_i \models \varphi_i$;

2. $L(A_i) \subseteq L(WA_i)$ for any assumption $A_i$ such that $L(A_i) \subseteq I(M_i)$ and $M_i|A_i \models \varphi_i$.

Remark 1. The notion of the weakest assumption is still based on local specifications. Observe that the weakest assumption $WA_i$ with respect to local specification $\varphi_i$ may be weaker than the weakest assumption with respect to global specification $\psi_i$. The condition $L(WA_i) \subseteq I(M_i)$ explicitly restrict only those infinite words admitted by module $M_i$ can be accepted by $WA_i$. This is rather implicit in the general assume-guarantee rules where all assumptions are generated on the same global alphabet.

4 Case Study

This section will illustrate an application of rule $R^\pi$ to verify the stability of an optimisation based congestion control system. Both the system and the property exhibit compositional structures. Please refer to [8] for more details about the dynamic system and the stability property.

4.1 Multi-Path Congestion Control

This subsection will introduce briefly the fluid-flow congestion control algorithm proposed by Kelly and Voice [7].

Suppose a network in which a number of sources communicate with a number of destinations. Between each source and destination a number of routes have been provisioned. Each route uses a number of links or, more generally, resources, each of which has a finite capacity constraint.

For each source $s$ and route $r$ available to $s$, the trajectory in the flow rate $x_r$ is given rise to as a continuous function of time $t$, subject to the following differential equation:

\[
\frac{d}{dt} x_r(t) = \kappa_r x_r(t) \left( 1 - \frac{y_r(t)}{U'(s(r))(x_s(r)(t))} \right)_{+} x_r(t),
\]

where $\kappa_r$ is a constant and

- $y_r(t)$ is the total cost on route $r$;
- $s(r)$ is the source that transmits along route $r$;
- $x_s(t)$ is the aggregate flow rate on all routes available to source $s$;
- $U_s$ is a utility function of the total flow sent by source $s$ and $U'_s$ is the first-order derivative of $U_s$;
- $(z)_{+}^+ = \min(0, z)$ if $x \leq 0$, otherwise $(z)_{+}^+ = z$.
Thus, each source $s$ adjusts the flow rate $x_r$ on route $r$ based on feedback $y_r$ from the network (indicating congestion). Then, the algorithm presented in [7] is composed of these sources acting synchronously and collectively. The stability of this algorithm has been proved in [7]. Herein, we consider the fully asynchronous variant of the algorithm under the fairness constraint that every source acts infinitely often. This model can represent uncertain delay between distributed sources.

4.2 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, that is, where $\frac{d}{dt}x_r(t) = 0$ for every route $r$. This is represented logically by the following CTL formula

$$AFAG \land (\land_{r \in s_i} x'_r = x_r)$$

where $s_i$ ranges over all the sources, $r \in s_i$ ranges over all the routes available to source $s_i$, and $x'_r$ is the next value of route flow rate $x_r$.

Lagrangian decomposition techniques reduce system stability onto individual modules [9]. Source $s_i$ 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 these routes (either serving or sharing resource with source $s_i$), that is, $\gamma(s_i) = \{ r \mid j \in r \text{ for any } r' \in s_i \text{ and } j \in r' \}$, where $j \in r$ ranges over all the resources used by route $r$. Then, local stability on source $s_i$ is represented by the following CTL formula

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

Observe that

$$(AFAG \land (\land_{r \in s_i} x'_r = x_r)) \iff (\land_i AFAG \land \bigwedge_{r \in \gamma(s_i)} x'_r = x_r)$$

rule $R^n$ can be instantiated for system stability as follows:

$$\forall 1 \leq i \leq n, \quad M_i|A_i \models AFAG \land \bigwedge_{r \in \gamma(s_i)} x'_r = x_r$$

SS

$$\forall 1 \leq i \leq n, \quad \exists 1 \leq k \leq \pi, \quad C^k_i \models A_i$$

$$M_1 | \cdots | M_n \models AFAG \land (\land_{s, r \in s_i} x'_r = x_r)$$

where source $s_i$ is represented by module $M_i$.

The rest of this section will illustrate an application of rule SS to verify the stability of the distributed congestion control algorithm introduced in Section 4.1. We first present in Section 4.3 an algorithm for computing the weakest assumptions; then we show in Section 4.4 how rule SS can be applied for reasoning about any number of sources, any initial flow configuration, and any topology of bounded degree.
4.3 Computing Assumptions

With respect to local stability, assumptions $A_i$ is meant to supply the sequence of inputs to module $M_i$ such that $M_i|A_i$ can eventually converge to certain configuration on $X_i \cup I_i$.

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

1. For each valuation $\alpha$ 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 $p_q \in F_{A_i}$ such that
   - $p_q \notin E_{A_i}$;
   - $\lambda_{A_i}(p_q) = \alpha$;
   - $p \xrightarrow{\lambda_{M_i}(q)} A_i p_q$.

4. $q_{A_{i0}}$ is the initial state, with $\lambda(q_{A_{i0}})$ 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 $A_i$.

![Figure 1: Büchi Accepting State in Assumptions](image)

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 $p_q$ where $\lambda_{A_i}(p_q) = \alpha$, with two additional transitions leading to it, as shown in Figure 1. Apparently, module $M_i$ at state $q$ would remain at this state under constantly repeated inputs $\alpha$, which is exactly what the specification of local stability expects.

Thus, we compute an assumption $A_i$ for module $M_i$ by looking at the module itself, regardless of the underlying topology. Theorem 4.1 shows that the assumption is an appropriate one for our purpose.

**Theorem 4.1.** Assumption $A_i$ generated by the above algorithm for module $M_i$ is the weakest assumption with respect to local specification $\text{AFAG} \wedge_{r \in \gamma(s_i)} x'_r = x_r$. 

12
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 \(q \in Q_M\), admits a self-loop transition under input \(\lambda_{A_i}(p_q)\). Correspondingly, the word accepted through such an accepting trace will end up 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.

We then prove by contradiction that assumption \(A_i\) is the weakest assumption with respect to the local specification. Suppose there exists an assumption \(A_i'\) such that \(L(A_i') \subseteq I(M_i)\) and \(M_i|A_i'\) satisfies the local specification, but there exists an infinite word \(\theta = a_0a_1 \cdots \in L(A_i')\) that is not accepted by \(A_i\).

Then, by this hypothesis and the definition of step 3, \(\theta\) can not be derived by \(A_i\).

Assume \(a_0 \cdots a_k (k \geq 0)\) is the longest prefix that can be derived from \(A_i\). This means that, for any valuation \(\rho\) on \(X_i\), no transition \(p \overset{a_k}{\rightarrow} A_i, p'\) exists such that \(\lambda_{A_i}(p) = a_k\) and \(\lambda_{A_i}(p') = a_{k+1}\). Hence, by the definition of step 2, no transition \(q \overset{a_k}{\rightarrow} M_i, q'\) exists such that for any states \(q, q' \in Q_M\). This conflicts with the hypothesis, which implies \(\theta \in I(M_i)\).

The time complexity of this algorithm is linear to the size of module \(M_i\). The worst run-time is \(O(2|T_M|)\). 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 \(|D||I_i| + |T_M|\) number of states and \(|T_M||D||I_i| + 2|T_M|\) 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 \(I_i\).

### 4.4 Experiments

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 the initial configuration \((u, v)\) for \(u, v \in D\) and the transitions defined by (a discrete instance of) Equation (1). Then, no matter how many sources a network may consist of, each source is generally of the form \(M_{u,v}\), where \(u, v \in D\).

Let \(A_{u_0,v_0}\) be the super assumption generated by the above algorithm for module \(M_{u_0,v_0}\). Then, we first check whether the composition of any two possible neighbour modules can guarantee these assumptions. This amounts to check whether

\[M_{u_1,v_1}|M_{u_1',v_1} = A_{u_0,v_0}\]  \hspace{1cm} (5)

for any \(u_0, v_0, u_1, v_1, u_1', v_1' \in D\).

Technically, we use GOAL [13] to compute the complement \(\text{co}A_{u_0,v_0}\) and then check whether any word derived by \(M_{u_1,v_1}|M_{u_1',v_1}\) is accepted by \(\text{co}A_{u_0,v_0}\). For the domain \(D = [1, 6]\), Table 1 reports the size of each assumption and its complement, represented as Büchi automata in GOAL, and also the time usage for complementation. Note that \(M_{u_0, v_0}\) is equivalent to \(M_{u_0, v_0}\) under permutation.

In total 46656 (= 6^6) instances of (5) need to be checked. It takes on average 0.26 second to run each instance (except the time usage for complementation).
We find that each instance is valid. This means that any word derived by the composition of any two possible modules is accepted by any assumption. Thus, the stability of such system is demonstrated for any number of sources and any initial flow configuration under the given topology.

Furthermore, this experiment can be extended for any topology with bounded degree (that is, 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 $\bigcup_{k=1}^{m} D^k$.

## 5 Conclusions

The paper presents a distributed assume-guarantee rule $R^\pi$ for distributed systems and global specifications consisting of conjunctions of local specifications. Rule $R^\pi$ is both sound and complete for local specifications, yet drawing conclusions on global specifications. Thus, a verification task on a system can be distributed onto individual modules and local specifications. Furthermore, rule $R^\pi$ triggers an incremental approach to perform compositional model checking which exploits the neighbourhood structure of interactions between modules. Each increment explores the consequences of interactions one step deeper 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 module when interacting with its neighbours. Thus, with development, our technique could greatly extend the range of network problems that could yield to a model checking approach.

Rule $R^*$ tackles the scalability issues in assume-guarantee reasoning by exploiting one possible compositional structure (conjunction) of global specifications, and by exploiting the neighbourhood dependency between individual modules. This is rather promising in practice in that verification sub-tasks are localised on individual modules, and the size of state space explored by each sub-task grows only if necessary and in a controlled manner. As future work, we would like to investigate other compositional structures of global specifications [14]. This bounded strategy could be also applicable for general specifications that possibly do not exhibit compositional structures.

References


A Tentative Rule $R_0$

$$
\forall 1 \leq i \leq n, \ M_i|A_i \models \varphi_i
$$

$$
R_0 \frac{\mathcal{L}(coA_1|\cdots|coA_n) = \emptyset}{M_1|\cdots|M_n \models \bigwedge_{i=1}^{n} \varphi_i}
$$

Proof. Unsoundness By construction. Consider a system consisting of the following four modules $M_i$ ($1 \leq i \leq n$):

<table>
<thead>
<tr>
<th>$M_i$</th>
<th>$X_i$</th>
<th>$I_i$</th>
<th>Transition Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>$M_1$</td>
<td>${x_1}$</td>
<td>${x_2, x_3}$</td>
<td>$x_1 = x_2 - x_3$</td>
</tr>
<tr>
<td>$M_2$</td>
<td>${x_2}$</td>
<td>${x_4}$</td>
<td>$x_2 = x_2 - x_4$</td>
</tr>
<tr>
<td>$M_3$</td>
<td>${x_3}$</td>
<td>${x_4}$</td>
<td>$x_3 = x_3 + x_4$</td>
</tr>
<tr>
<td>$M_4$</td>
<td>${x_4}$</td>
<td>${x_2, x_3}$</td>
<td>$x_4' = 1$</td>
</tr>
</tbody>
</table>
and local specification

$$AFAG \; \wedge_{x \in X_i \cup I_i} (x' = x)$$

on each module $M_i$, where $x'$ is the next value of variable $x$.

With the 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

$$M_1|M_2|M_3|M_4 \not\models \wedge_{i=1}^4 AFAG (x'_i = x_i)$$

because with the system $x_2$ and $x_3$ evolve by diverging from each other. However, modules $M_1, M_2$ and $M_3$ all have chance to converge under certain inputs, while obviously module $M_4$ is already in a stable state no matter what inputs may be. So, $\mathcal{L}(coA_4) = \emptyset$. Therefore, both premises hold.

**Completeness** By contradiction. For each module $M_i$, assume $WA_i$ is the weakest assumption with respect to $\varphi_i$. Since $M_1|\cdots|M_n \models \wedge_{i=1}^n \varphi_i$ and $\wedge_{i=1}^n \varphi_i$ implies $\varphi_i$, such $WA_i$ does exist.

Suppose these exists an infinite word $w$ accepted by $coWA_1|\cdots|coWA_n$. Then, 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 $1 \leq j \leq n$, these exists an infinite word $w'_j$ that is accepted by $M_j|coWA_j$ but does not satisfy $\varphi_j$. Hence, $w'_j$ does not satisfy $\wedge_{i=1}^n \varphi_i$ for any $1 \leq j \leq n$. Thus, there exists an infinite word $w'$ that can be derived by $M_1|\cdots|M_n$ and $w'_j$ is a stuttering projection of $w'$ on $X_j \cup I_j$ for any $1 \leq j \leq n$. Therefore, $w'$ does not satisfy $\wedge_{i=1}^n \varphi_i$. This conflicts with the premise $M_1|\cdots|M_n \models \wedge_{i=1}^n \varphi_i$. 

\[\square\]