Network Invariants in Action ⋆

Yonit Kesten[1], Amir Pnueli[2], Elad Shahar[2], and Lenore Zuck[3]

[1] Ben Gurion University, ykesten@bgumail.bgu.ac.il
[2] Weizmann Institute of Science, {amir,elad}@wisdom.weizmann.ac.il
[3] New York University, New York, zuck@cs.nyu.edu

Abstract. The paper presents the method of *network invariants* for verifying a wide spectrum of LTL properties, including liveness, of parameterized systems. This method can be applied to establish the validity of the property over a system $S(n)$ for every value of the parameter n. The application of the method requires checking abstraction relations between two finite-state systems. We present a proof rule, based on the method of Abstraction Mapping by Abadi and Lamport, which has been implemented on the TLV model checker and incorporates both history and prophecy variables. The effectiveness of the network invariant method is illustrated on several examples, including a deterministic and probabilistic versions of the dining-philosophers problem.

1 Introduction

The emerging interest in embedded systems brought forth a surge of research in *automatic uniform verification* of *parameterized systems*: Given a parameterized system $S(n) : P_1\| \cdots \|P_n$ and a property p, uniform verification attempts to verify $S(n) \models p$ for every $n > 1$. Verification of such systems is known to be undecidable [2]; much of the recent research has been devoted to identifying conditions that enable their automatic verification and abstraction tools to facilitate the task (e.g., [7, 6, 19, 21, 24].)

One of the promising approaches to the uniform verification of parameterized systems is the method of *network invariants*, first mentioned in [3, 25], further developed in [27] (who also coined the name "network invariant"), and elaborated in [13] into a working method. The formulation here follows [10], which is somewhat akin in spirit to both [27] and [13]. A significant improvement of our approach over [27] and [13] and most other works that use abstraction for verification is that our notion of abstraction takes into account the fairness properties of the compared systems. Consequently, our abstraction can support and simplify proofs of liveness properties as well as any other property expressible by LTL.

The main idea of the method is to abstract $n-1$ of the processes, say the composition $P_2 \parallel \cdots \parallel P_n$, into a single finite-state process $\mathcal{I}$, independent of n. We

⋆ This research was supported in part by the John von Neumann Minerva Center for Verification of Reactive Systems, The European Community IST project "Advance", and ONR grant N00014-99-1-0131.

refer to $\mathcal{I}$ as the *network invariant*. If possible, this reduces the parameterized verification problem $(P_1 \parallel \cdots \parallel P_n) \models p$ into the fixed-size verification problem $(P_1 \parallel \mathcal{I}) \models p$. In order to show that $\mathcal{I}$ is a correct abstraction of any number of processes (assuming that $P_2, \ldots, P_n$ are all identical except for renaming), it is sufficient to apply an inductive argument, using $P \sqsubseteq \mathcal{I}$ as the induction base, and $(P \parallel \mathcal{I}) \sqsubseteq \mathcal{I}$ as the induction step. These two abstraction proof obligations compare two finite-state systems and can, in principle, be performed algorithmically by a model checker.

Unfortunately this approach is intractable. The obvious way to establish algorithmically that a concrete system S_c is abstracted by the abstract system S_a is by showing that $S_c \cap \bar{S_a}$ admits no computations, where $\bar{S_a}$ is the complement of S_a. Since fair systems are equivalent to Streett automata, the set of states obtained by complementing S_a is usually prohibitively large. Consequently, abstraction is accomplished by establishing a step-by-step simulation relation between a concrete computation and an abstract one, following the *abstraction mapping* method of Abadi and Lamport [1]. This approach has been implemented on the Weizmann Institute Temporal Logic Verifier TLV.

In this paper we present our theory of abstraction which can be used for the verification of arbitrary LTL formulas (including liveness). We introduce the method of network invariants and the abstraction proof rule used for discharging the abstraction proof obligations. The method is then illustrated over several examples of parameterized systems for which we uniformly verify their essential safety and liveness properties.

The two examples we study deal with a solution to the dining philosophers problem. In the first example, we consider the case in which each of the philosophers follows a deterministic protocol. The property we establish is individual accessibility, that is, every hungry philosopher eventually eats. We present two network invariants for this problem. The first invariant is a carefully designed abstraction of the two end processes in a string of philosophers. While the design of this invariant took a long time to develop, its proof obligations were very straightforward to establish. At the other end of the spectrum, we present a very natural invariant, which is just a string of 3 philosophers. The main proof obligation here was to show that a string of 4 philosophers is abstracted by a string of 3. This required an abstraction mapping which uses prophecy variables. We then turn to a solution of the dining philosophers which is a variant of the probabilistic "Courteous Philosophers" protocol of [14]. For this protocol, we also succeeded in automatically verifying individual accessibility using a two-halves abstraction. As a third example, we considered the distributed termination algorithm of [5]. This algorithm also considers a ring of processes. However, unlike the two preceding cases, in addition to the communication with its two close neighbors, a process also maintains a communication of a different kind with (potentially) every other process. In order to make this problem amenable to treatment by the network-invariants method, we had first to abstract this "unconstrained" model of communication into a simpler representation. Details about the successful and effective application of the method to this case study are provided in [12]. [1]

The variety of examples of application of the network invariant method to automatic verification of liveness properties of parameterized system, demonstrates the power of the methodology and the wide range of its applicability.

Many methods have been proposed for the uniform verification of parameterized systems. These include methods based on explicit induction ([26, 24]) network invariants that can be viewed as implicit induction ([13], [27], [8], [15]), methods that can be viewed as abstraction and approximation of network invariants ([3], [25], [4]), and other methods that can be viewed as based on abstraction ([9], [7]).

Our approach to verification by network invariants has been presented first in [10]. The work in [10] was based on a significantly weaker proof rule than the one we present here. For example, the proof rule presented there could not handle prophecy variables and was therefore inherently incomplete. Also, some of the solutions presented in [10] used the construct of *chaos* in cases where it was not necessary, leading to overly cumbersome abstractions. Relative to [10], this paper presents a more powerful rule and additional interesting examples, such as the verification of a probabilistic protocol.

2 Fair Discrete Systems

As a computational model for reactive systems we take the model of *fair discrete systems* (FDS) [10], which is a slight variation on the model of *fair transition system* [18]. Under this model, a system $\mathcal{D}:\langle V, \mathcal{O}, W, \Theta, \rho, \mathcal{J}, \mathcal{C}\rangle$ consists of the following components:

- V: A finite set of typed *system variables*, containing data and control variables. A state s is an assignment of type-compatible values to the system variables V. For a set of variables $U \subseteq V$, we denote by $s[U]$ the values assigned by state s to the variables U. The set of states over V is denoted by Σ. In this paper, we assume that Σ is finite.
- $\mathcal{O} \subseteq V$: A subset of *observable variables*. These are the variables which can be externally observed.
- $W \subseteq V$: A subset of *owned* variables. These are variables which only the system itself can modify. All other variables can also be modified by steps of the environment.
- Θ: The *initial condition* – an *assertion* (first-order state formula) characterizing the initial states.
- ρ: A *transition relation* – an assertion $\rho(V, V')$, relating the values V of the variables in state $s \in \Sigma$ to the values V' in a ρ-successor state $s' \in \Sigma$.
- $\mathcal{J}$: A set of *justice* (*weak fairness*) *requirements*. Each justice requirement $J \in \mathcal{J}$ is an assertion, intended to guarantee that every computation contains infinitely many J-states (states satisfying J.)

[1] The TLV code of all examples presented in this paper can be found in www.wisdom.weizmann.ac.il/ verify/publications/2002/KPZ02.html#explanation.

- $\mathcal{C}$: A set of *compassion* (*strong fairness*) *requirements*. Each compassion requirement is a pair $\langle p, q\rangle \in \mathcal{C}$ of assertions, intended to guarantee that every computation containing infinitely many p-states also contains infinitely many q-states.

We require that every state $s \in \Sigma$ has at least one ρ-successor. This is often ensured by including in ρ the *idling* disjunct $V = V'$ (also called the *stuttering* step). In such cases, every state s is its own ρ-successor. A system is said to be *closed* if $W = V$, i.e., all variables are owned by the system. Otherwise, the system is said to be *open*. Let $\sigma\colon s_0, s_1, s_2, \ldots$, be an infinite sequence of states, φ be an assertion and $j \geq 0$ be a natural number. We say that s_j is a *φ-state* if it satisfies φ, and we say that j is a *φ-position* of σ if s_j is a φ-state.

Let $\mathcal{D}$ be an FDS as above. We define an (*open*) *run* of $\mathcal{D}$ to be an infinite sequence of states $\sigma\colon s_0, s_1, s_2, \ldots$, satisfying the following requirements:

- *Initiality:* s_0 is initial, i.e., $s_0 \models \Theta$.
- *Consecution:* For each $j = 0, 1, \ldots$,
  - $s_{2j+1}[W] = s_{2j}[W]$. That is, s_{2j+1} and s_{2j} agree on the interpretation of the owned variables W.
  - s_{2j+2} is a ρ-successor of s_{2j+1}.

Thus, an open run of a system consists of a strict interleaving of system with environment actions, where the system action has to satisfy the transition relation ρ, while the environment step is only required to preserve the values of the owned variables. We say that a run of $\mathcal{D}$ is a *computation* if it satisfies the following requirements:

- *Justice:* For each $J \in \mathcal{J}$, σ contains infinitely many J-positions
- *Compassion:* For each $\langle p, q\rangle \in \mathcal{C}$, if σ contains infinitely many p-positions, it must also contain infinitely many q-positions.

We denote by $\mathcal{C}omp(\mathcal{D})$ the set of all computations of $\mathcal{D}$.

Systems $\mathcal{D}_1$ and $\mathcal{D}_2$ are *compatible* if their sets of owned variables are disjoint, and the intersection of their variables is observable in both systems. For compatible systems $\mathcal{D}_1$ and $\mathcal{D}_2$, we define their *asynchronous parallel composition*, denoted by $\mathcal{D}_1 \| \mathcal{D}_2$, as the FDS whose sets of variables, observable variables, owned variables, justice, and compassion sets are the unions of the corresponding sets in the two systems, whose initial condition is the conjunction of the initial conditions, and whose transition relation is the disjunction of the two transition relations. Thus, a step in an execution of the composed system is a step of system $\mathcal{D}_1$, or a step of system $\mathcal{D}_2$, or an environment step. We also provide a *restriction* operation, which moves a specified variable to the category of owned variables and makes it non-observable. We denote by [**restrict** x **in** $\mathcal{D}$] the system obtained by restricting variable x in system $\mathcal{D}$.

An *observation* of $\mathcal{D}$ is a projection of a $\mathcal{D}$-computation onto $\mathcal{O}$. We denote by $Obs(\mathcal{D})$ the set of all observations of $\mathcal{D}$. Systems $\mathcal{D}_C$ and $\mathcal{D}_A$ are said to be *comparable* if they have the same sets of observable variables, i.e., $\mathcal{O}_{C} = \mathcal{O}_{A}$ or, alternatively, if there is a 1-1 correspondence between $\mathcal{O}_C$ and $\mathcal{O}_A$. System $\mathcal{D}_A$ is said to be an *abstraction* of the comparable system $\mathcal{D}_C$, denoted $\mathcal{D}_C \sqsubseteq \mathcal{D}_A$, if $Obs(\mathcal{D}_C) \subseteq Obs(\mathcal{D}_A)$. The abstraction relation is reflexive, transitive, and compositional, that is, whenever $\mathcal{D}_C \sqsubseteq \mathcal{D}_A$ then $(\mathcal{D}_C \| Q) \sqsubseteq (\mathcal{D}_A \| Q)$. It is also *property restricting*. That is, if $\mathcal{D}_C \sqsubseteq \mathcal{D}_A$ then $\mathcal{D}_A \models p$ implies that $\mathcal{D}_C \models p$ (see [10] for more details).

All our concrete examples are given in SPL (Simple Programming Language), which is used to represent concurrent programs (e.g., [18, 16]). Every SPL program can be compiled into an FDS in a straightforward manner. In particular, every statement in an SPL program contributes a disjunct to the transition relation. For example, the assignment statement "ℓ_0: $y := x + 1$; ℓ_1:" contributes to ρ the disjunct

$$\rho_{\ell_0}: \quad at\_\ell_0 \ \wedge \ at\_\ell_1' \ \wedge \ y' = x + 1 \ \wedge \ x' = x.$$

The predicates at_ℓ_0 and at_ℓ_1' stand, respectively, for the assertions $\pi_i = 0$ and $\pi_i' = 1$, where π_i is the control variable denoting the current location within the process to which the statement belongs. Every variable declared in an SPL program is specified as having one of the modes *in*, *out*, *in-out*, or *local*. All but the local variables are observable. The non-input (out and local) variables are owned, while the input variables (in and in-out) are not owned.

Properties are specified in propositional linear time temporal logic (LTL) over the states of $\mathcal{D}$ (see [17], [18] for LTL). A property φ is *valid* over $\mathcal{D}$ if $\sigma \models \varphi$ for every $\sigma \in Comp(\mathcal{D})$.

3 Verification by Abstract Network Invariants

We define a *binary process* $Q(\vec{x};\vec{y})$ to be a process with two ordered sequences of observable variables $\vec{x}$ and $\vec{y}$. When $\vec{x}$ and $\vec{y}$ consist of a single variable we use the notation $Q(x;y)$. Two binary processes Q and R can be composed to yield another binary process, using the *modular composition* operator $\circ$ defined by

$$(Q \circ R)(\vec{x};\vec{z}) \quad = \quad [\textbf{restrict } \vec{y} \textbf{ in } Q(\vec{x};\vec{y}) \parallel R(\vec{y};\vec{z})]$$

Binary processes $P_1, \ldots, P_m$ can be composed into a closed ring structure (having no observables) defined by

$$(P_1 \circ \cdots \circ P_m \circ) = [\textbf{restrict } \vec{x}_1, \ldots, \vec{x}_m \textbf{ in } \quad P_1(\vec{x}_1;\vec{x}_2) \parallel \cdots \parallel P_m(\vec{x}_m;\vec{x}_1)]$$

The dangling $\circ$ denotes that process P_m is composed with P_1. In this work we deal with parameterized systems of the form $P(n) = (P_1 \circ \cdots \circ P_n \circ)$, where each P_i is a finite state binary process. Such a system represents an infinite *family* of systems (one for each value of n). Our objective is to verify uniformly (i.e., for every value of $n > 1$) that property p is valid. For simplicity of presentation, assume that the property p only refers to the observable variables of P_1 and that processes $P_1, \ldots, P_{n-1}$ are identical (up to renaming) and can be represented by the generic binary process Q. That is, $P_1(\vec{x};\vec{y}) = \cdots = P_{n-1}(\vec{x};\vec{y}) = Q(\vec{x};\vec{y})$. The *network invariants method* can be summarized as follows:

1. Devise a *network invariant* $\mathcal{I} = \mathcal{I}(\vec{x}, \vec{y})$ which is an FDS intended to provide an abstraction for the (open) modular composition $Q^k = \underbrace{Q \circ \cdots \circ Q}_{k}$ for any $k \geq 2$.
2. Confirm that $\mathcal{I}$ is indeed a network invariant, by establishing that $Q \sqsubseteq \mathcal{I}$ and $(Q \circ \mathcal{I}) \sqsubseteq \mathcal{I}$.
3. Model check $(P_1 \circ \mathcal{I} \circ P_n \circ) \models p$.

As presented here, the rule is adequate for proving properties of P_1. Another typical situation is when we wish to prove properties of a generic P_j for $j < n$. In this case, we model check in step 3 that $(\mathcal{I} \circ P \circ \mathcal{I} \circ P_n \circ) \models p$.

Verification by the network invariants method entails *model checking* (step 3) and *verifying abstraction* (step 2). Most of the available computer aided verification (CAV) tools for LTL are designed to support verification tasks: They accept a system and an LTL formula as input, and check whether the formula is valid over the system.

Based on the *abstraction mapping* of [1], we present in Fig. 1 a proof rule that reduces the abstraction problem into a verification problem. There, we assume two comparable FDS's, a *concrete* $\mathcal{D}_C : \langle V_C, \mathcal{O}_C, W_C, \Theta_C, \rho_C, \mathcal{J}_C, \mathcal{C}_C \rangle$ and an *abstract* $\mathcal{D}_A : \langle V_A, \mathcal{O}_A, W_A, \Theta_A, \rho_A, \mathcal{J}_A, \mathcal{C}_A \rangle$, and we wish to establish that $\mathcal{D}_C \sqsubseteq \mathcal{D}_A$. Without loss of generality, we assume that $V_C \cap V_A = \emptyset$, and that there exists a 1-1 correspondence between the concrete observables $\mathcal{O}_C$ and the abstract observables $\mathcal{O}_A$.

The method assumes the identification of an *abstraction mapping* $\alpha : (U = \mathcal{E}_\alpha(V_C))$ which assigns expressions over the concrete variables to *some* of the abstract variables $U \subseteq V_A$. For an abstract assertion φ, we denote by $\varphi[\alpha]$ the assertion obtained by replacing the variables in U by their concrete expressions. We say that the abstract state S is an α-image of the concrete state s if the values of $\mathcal{E}_\alpha$ in s equal the values of the variables U in S.

A1.	$\Theta_C \rightarrow \exists V_A : \Theta_A[\alpha]$	
A2.	$\mathcal{D}_C \models \Box(\rho_C \rightarrow \exists V'_A : \rho_A[\alpha][\alpha'])$	
A3.	$\mathcal{D}_C \models \Box(\alpha \rightarrow \mathcal{O}_C = \mathcal{O}_A)$	
A4.	$\mathcal{D}_C \models \Box\Diamond J[\alpha]$,	for every $J \in \mathcal{J}_A$
A5.	$\mathcal{D}_C \models \Box\Diamond p[\alpha] \rightarrow \Box\Diamond q[\alpha]$,	for every $(p, q) \in \mathcal{C}_A$
	$\mathcal{D}_C \sqsubseteq \mathcal{D}_A$	

Fig. 1. Rule AL-ABS.

Premise A1 of the rule states that for every initial concrete state s, it is possible to find an initial abstract state $S \models \Theta_A$, such that $\langle s, S \rangle \models \alpha$. The existential quantification allows to choose arbitrary values for the abstract variables not mapped by α, i.e. the variables in $V_A - U$.

Premise A2 states that for every pair of concrete states, s_1 and s_2, such that s_2 is a ρ_C-successor of s_1, and an abstract state S_1 which is a α-image of s_1, it is possible to find an abstract state S_2 such that S_2 is an α-image of s_2 and is also a ρ_{A}-successor of S_1. Together, A1 and A2 guarantee that, for every run $s_0, s_1, \ldots$ of $\mathcal{D}_C$ there exists a run $S_0, S_1, \ldots$ of $\mathcal{D}_A$, such that S_j is an α-image of s_j for every $j \geq 0$. Premise A3 states that whenever an abstract state S is an α-image of a concrete state s, then the values of the corresponding observables in the two states match. Premises A4 and A5 ensure that the abstract fairness requirements (justice and compassion, respectively) hold in any abstract state sequence which is a (point-wise) α-image of a concrete computation. It follows that every α-image of a concrete computation σ obtained by applications of premises A1 and A2 is an abstract computation whose observables match the observables of σ. This leads to the following claim:

Claim 1. *If the premises of rule* AL-ABS *are valid for some choice of* α, *then* $\mathcal{D}_A$ *is an abstraction of* $\mathcal{D}_C$.

Rule AL-ABS has been implemented in the current TLV-BASIC implementation of the abstraction checker within TLV [20].

As explained in [1], a rule such as AL-ABS cannot be complete unless we allow the mapping at position j to refer to concrete states at positions other than j. This is handled in [1] by augmenting the concrete system by *history* and *prophecy* variables. Following this recommendation, we allow the concrete system to be augmented with a set V_H of *history variables* and a set V_P of *prophecy variables*. We assume that the three sets, V_C, V_H, and V_P, are pairwise disjoint. The result is an augmented concrete system $\mathcal{D}_C^* : \langle V_C^*, \mathcal{O}, W_C^*, \Theta_C^*, \rho_C^*, \mathcal{J}_C, \mathcal{C}_C \rangle$, where

$$
\begin{array}{ll}
V_C^* = V_C \cup V_H \cup V_P & W_C^* \;=\; W \cup V_H \\
\Theta_C^* = \Theta_C \wedge \bigwedge_{x \in V_H} (x = f_x(V_C, V_P)) & \\
\rho_C^* = \rho_C \wedge \bigwedge_{x \in V_H} x' = g_x(V_C^*, V_C', V_P') \wedge \bigwedge_{y \in V_P} y = \varphi_y(V_C) &
\end{array}
$$

In these definitions, each f_x and g_x are state functions, while each $\varphi_y(V_C)$ is a future temporal formula referring only to the variables in V_C. Thus, unlike [1], we use *transition relations* to define the values of history variables, and *future* LTL *formulas* to define the values of prophecy variables. The clause $y = \varphi_y(V_C)$ added to the transition relation implies that at any position $j \geq 0$, the value of the boolean variable y is 1 iff the formula $\varphi_y(V_C)$ holds at this position.

It is not difficult to see that the augmentation scheme proposed above is *non-constraining*. Namely, for every computation σ of the original concrete system $\mathcal{D}_C$ there exists a computation σ^* of $\mathcal{D}_C^*$ such that σ and σ^* agree on the values of the variables in V_C. It follows that rule AL-ABS can be applied to $\mathcal{D}_C^*$ an arbitrary non-constraining augmentation of $\mathcal{D}_C$ and if the premises are valid, then so is the conclusion. This extended version of the rule has been implemented within the TLV model checker. Handling of the prophecy variables definitions is performed by constructing an appropriate *temporal tester* [11] for each of the future temporal formulas appearing in the prophecy schemes, and composing it with the concrete system.

The presented version of the rule is formulated for the case that the abstraction mapping is a *function*. We do have extensions of the rule for the more general case that α is a *relation* rather than a function.

4 Deterministic Dining Philosophers

As a first example, we apply the network invariant method to a deterministic solution to the dining philosophers problem (DDP). As originally described by Dijkstra, n philosophers are seated at a round table, with a fork placed in between each two neighbors. Each philosopher alternates between a thinking phase and a phase in which he becomes hungry and wishes to eat. In order to eat, a philosopher needs to acquire the forks on both its sides. A solution to the problem consists of protocols to the philosophers (and, possibly, forks) that guarantees that no two adjacent philosophers eat at the same time (mutual exclusion) and that every hungry philosopher eventually gets to eat (individual accessibility). It is well known that there are no symmetric deterministic solutions to the problem. In this section we explore a deterministic asymmetric solution. In the next section we explore a non-deterministic symmetric solution.

A deterministic solution to the problem that uses semaphores for forks, is given by a modular composition $(Q^{n-1} \circ C \circ)$, where the binary processes $Q(\mathit{left}; \mathit{right})$ and $C(\mathit{left}; \mathit{right})$ are presented in Fig. 2. In this program, $n-1$ philosophers reach first for the fork on their left, and then for their right fork. One philosopher, C, is a *contrary philosopher*, reaching first for its right fork and only later for its left fork.

$$(Q^{n-1} \circ C \circ) \quad \text{where}$$

$$Q(\mathit{left}; \mathit{right}) :: \left[\begin{array}{l} \textbf{loop forever do} \\ \quad \left[\begin{array}{l} \ell_0 : \textbf{NonCritical} \\ \ell_1 : \textbf{request } \mathit{left} \\ \ell_2 : \textbf{request } \mathit{right} \\ \ell_3 : \textbf{Critical} \\ \ell_4 : \textbf{release } \mathit{left} \\ \ell_5 : \textbf{release } \mathit{right} \end{array}\right] \end{array}\right] \qquad C(\mathit{left}; \mathit{right}) :: \left[\begin{array}{l} \textbf{loop forever do} \\ \quad \left[\begin{array}{l} \ell_0 : \textbf{NonCritical} \\ \ell_1 : \textbf{request } \mathit{right} \\ \ell_2 : \textbf{request } \mathit{left} \\ \ell_3 : \textbf{Critical} \\ \ell_4 : \textbf{release } \mathit{right} \\ \ell_5 : \textbf{release } \mathit{left} \end{array}\right] \end{array}\right]$$

Fig. 2. Program DINE-CONTR: solution with one contrary philosopher.

Our goal is to prove the liveness property of individual accessibility (starvation freedom), specified by the formula $\varphi : \quad at\_\ell_1[j] \implies \Diamond\, at\_\ell_3[j]$, for every philosopher $j = 1, \ldots, n$. Our strategy is to construct a network invariant $\mathcal{I}(\mathit{left}; \mathit{right})$ that abstracts a philosophers chain Q^k for any $k \geq 2$. In the following we present two such network invariants.

The "*Two-Halves*" Abstraction

The first network invariant $\mathcal{I}(left; right)$ is presented in Fig. 3 and can be viewed as the parallel composition of two "one-sided" philosophers. The compassion requirement reflects the fact that $\mathcal{I}$ can deadlock at location ℓ_1 only if, from some point on, the fork on the right (*right*) remains continuously unavailable.

$$\mathcal{I}(\mathit{left}; \mathit{right}) :: \quad \begin{bmatrix} \textbf{loop forever do} \\ \begin{bmatrix} \ell_0 : \textbf{request } \mathit{left} \\ \ell_1 : \textbf{release } \mathit{left} \end{bmatrix} \end{bmatrix} \quad \| \quad \begin{bmatrix} \textbf{loop forever do} \\ \begin{bmatrix} m_0 : \textbf{request } \mathit{right} \\ m_1 : \textbf{release } \mathit{right} \end{bmatrix} \end{bmatrix}$$

$$\mathcal{J} : \neg at\_m_1 \quad \mathcal{C} : (\mathit{right}, \neg at\_\ell_1)$$

Fig. 3. The Two-Halves Network Invariant

To establish that $\mathcal{I}$ is a network invariant, we use rule AL-ABS to verify $(Q \circ Q) \sqsubseteq \mathcal{I}$ and $(Q \circ \mathcal{I}) \sqsubseteq \mathcal{I}$, using an obvious abstraction mapping with no augmentation of the concrete system. To show that an arbitrary regular philosopher never starves, we model check

$$(\mathcal{I} \circ Q \circ \mathcal{I} \circ C \circ) \quad \models \quad at\_\ell_1[Q] \implies \Diamond\, at\_\ell_3[Q]$$

where C is a contrary philosopher.

In [10] we presented a similar two-halves abstraction, that contained a special *chaos* state, used as an escape state for both components of the abstraction, whenever an environment fault is detected. As can be seen by the current abstraction, the use of *chaos* is unnecessary.

The "*Four-by-Three*" Abstraction with Prophecy

An alternative network invariant is obtained by taking $\mathcal{I} = Q^3$, i.e. a chain of 3 philosophers. To prove that this is an invariant, it is sufficient to establish the abstraction $Q^4 \sqsubseteq Q^3$, that is, to prove that 3 philosophers can faithfully emulate 4 philosophers.

Let $Q_1 \circ Q_2 \circ Q_3 \circ Q_4$ and $Q_5 \circ Q_6 \circ Q_7$ be the modular composition of four and three regular philosophers. The abstraction mapping is defined such that Q_5 mimics Q_1 and Q_7 mimics Q_4. As to Q_6, it remains idle until $Q_1 \circ Q_2 \circ Q_3 \circ Q_4$ reaches a *deadlock* ($Q_1, \ldots, Q_4$ all remain at location ℓ_2 with their right forks being used), at which point Q_6 joins Q_5 and Q_7 to form a similar deadlock at the abstract level. To sense a guaranteed deadlock, we augment the concrete system with a *prophecy variable* $v \in V_P$ and associate v with the temporal formula $\psi : \Box\, \mathit{deadlock}$. Namely, v is true in all states of the concrete system which satisfy ψ. The new variable v is then used in the definition of the abstraction mapping.

5 Probabilistic Courteous Philosophers

As a second example, we consider Lehman and Rabin's *courteous philosophers* protocol [14]. The protocol gives a symmetric, distributed solution to the dining philosophers problem, by introducing probabilistic transitions. An SPL code of the protocol is presented in Fig. 4.[2] In this protocol, the forks (represented by the array $y[1..n]$) are shared variables that are reset when held and set when on the table. In addition to the forks, adjacent philosophers $P[i]$ and $P[i \oplus 1]$[3] share a $last[i \oplus 1]$ variable which indicates whether $P[i]$ is the last to have eaten between the two. Each philosopher $P[i]$ has two additional variables $signL[i]$ and $signR[i]$ that signal its wish to eat to its left and right neighbors. In order to choose the first fork to be picked, a philosopher flips a coin, represented by the probabilistic statement **goto** $\{0.5 : \ell_2; 0.5 : \ell_5\}$ at location ℓ_1. A philosopher can pick its first fork (ℓ_2 and ℓ_5) only when the neighbor with whom it shares the fork is either not hungry or is the last to have eaten between the two. Once it gains the first fork, the philosopher checks whether its second fork is available. If it is, it proceeds to eat (ℓ_8). Else, it returns the *first* fork (ℓ_4 and ℓ_7) and returns to flipping the coin (ℓ_1). The justice requirements of the system are the obvious ones, and there are no compassion requirements.

In [14], the protocol is claimed to satisfy individual accessibility to ℓ_8 with probability 1, under the obvious justice requirements and appropriate assumptions about the probabilistic choices. To provide an automatic proof, we perform the following transformations. First, as we prove in [28], in order to prove the accessibility property of the protocol, it suffices to consider a non-probabilistic version of it, where the coin flips in location ℓ_1 are replaced by non-deterministic choices, and compassion requirements are added to capture the fairness required of the coin flips. Next, we reduce the state space by eliminating the variables $y[i]$, $signL[i]$ and $signR[i]$, whose values can be uniquely determined by the locations of the relevant processes.

The result is program DINE presented in Fig. 5. Process Q has the interface list $(lloc, last, loc;\ loc, rlast, rloc)$ in which loc (appearing twice) is the process own program counter (location) and $last$ is the variable declared within the process. Variables $lloc$ and $rloc$ are the locations of the left and right neighbors of Q, respectively, while $rlast$ is the $last$ variable declared in the right neighbor of Q. Every process in the program is associated with a set of *justice* requirements and a set of *compassion* requirements. The justice requirements are

$$\begin{array}{l} \{loc \neq 1, \quad loc \neq 3, \quad loc \neq 4, \quad loc \neq 6, \quad loc \neq 7, \quad loc \neq 8, \\ \neg(loc = 2 \ \wedge\ (lloc = 0 \ \vee\ lloc \in \{0..5\} \wedge last \neq 0)), \\ \neg(loc = 5 \ \wedge\ (rloc = 0 \ \vee\ rloc \in \{1,2,5..7\} \wedge rlast \neq 1))\} \end{array}$$

[2] In the protocol, as presented in [14], the instructions appearing at location 8 are not atomic. Making them atomic, as we did in our presentation, does not impair the proof since none of these non-atomic assignments are observable to a single process. It does, however, reduce state space for model-checking.

[3] We define $i \oplus 1 = (i \ mod\ n) + 1$ and $i \ominus 1 = (i - 2\ mod\ n) + 1$, so that $n \oplus 1 = 1$ and $1 \ominus 1 = n$.

$$
\begin{array}{ll}
& \begin{array}{lll}
\textbf{in} & n: & \textbf{integer where } n \geq 2 \\
\textbf{local} & y: & \textbf{array } [1..n] \textbf{ of boolean init } 1 \\
\textbf{local} & signL, signR: & \textbf{array } [1..n] \textbf{ of boolean init } 0 \\
\textbf{local} & last: & \textbf{array } [1..n] \textbf{ of } \{-1,0,1\} \textbf{ init } -1
\end{array} \\
\|_{i=1}^{n} \; P[i] :: & \left[\begin{array}{l}
\textbf{loop forever do} \\
\left[\begin{array}{l}
\ell_0 : \textbf{non-critical} \\
\ell_1 : signL[i] := 1; signR[i] := 1; \textbf{goto } \{0.5 : \ell_2; 0.5 : \ell_5\} \\
\ell_2 : \textbf{await } y[i] \;\wedge\; (\neg signR[i \ominus 1] \;\vee\; last[i] = 1) \\
\quad\quad \textbf{and then } y[i] := 0 \\
\ell_3 : \textbf{If } y[i \oplus 1] = 1 \\
\quad\quad\quad \textbf{then } y[i \oplus 1] := 0\textbf{; goto } \ell_8 \\
\ell_4 : y[i] := 1;\; \textbf{goto } \ell_1 \\
\ell_5 : \textbf{await } y[i \oplus 1] \;\wedge\; (\neg signL[i \oplus 1] \;\vee\; last[i \oplus 1] = 0) \\
\quad\quad \textbf{and then } y[i \oplus 1] := 0 \\
\ell_6 : \textbf{If } y[i] = 1 \\
\quad\quad\quad \textbf{then } y[i] := 0\textbf{; goto } \ell_8 \\
\ell_7 : y[i \oplus 1] := 1;\; \textbf{goto } \ell_1 \\
\ell_8 : \left\langle \begin{array}{l}
\textbf{Critical}; \quad signL[i] := 0; signR[i] := 0 \\
last[i] := 0; last[i \oplus 1] := 1; \quad y[i] := 1; y[i \oplus 1] := 1
\end{array} \right\rangle
\end{array} \right]
\end{array} \right]
\end{array}
$$

Fig. 4. The Courteous Philosophers

$$
\begin{array}{l}
\qquad\qquad\qquad Q^n \qquad \text{where} \\
Q(lloc, last, loc;\; loc, rlast, rloc) :: \\
\quad \left[\begin{array}{l}
\textbf{local } last : [-1..1] \textbf{ init } -1 \\
\textbf{loop forever do} \\
\left[\begin{array}{ll}
\ell_0\text{:} & \textbf{Think} \\
\ell_1\text{:} & \textbf{go to } \{\ell_2,\; \ell_5\} \\
\ell_2\text{:} & \textbf{await } lloc = 0 \;\vee\; lloc \in \{0..5\} \wedge last \neq 0 \\
\ell_3\text{:} & \textbf{if } rloc \in \{0..2, 5..7\} \textbf{ then go to } \ell_8 \\
\ell_4\text{:} & \textbf{go to } \ell_1 \\
\ell_5\text{:} & \textbf{await } rloc = 0 \;\vee\; rloc \in \{1, 2, 5..7\} \wedge rlast \neq 1 \\
\ell_6\text{:} & \textbf{if } lloc \in \{0..5\} \textbf{ then go to } \ell_8 \\
\ell_7\text{:} & \textbf{go to } \ell_1 \\
\ell_8\text{:} & \textbf{Eat}; \quad last := 0; \quad rlast := 1
\end{array} \right]
\end{array} \right]
\end{array}
$$

Fig. 5. Program DINE: Location-based Courteous Philosophers

The justice requirements guarantee that no process can get stuck forever at any of the locations $\ell_1, \ell_3, \ell_4, \ell_6, \ell_7, \ell_8$ or at locations ℓ_2, ℓ_5 when their exit conditions are continuously true.

The role of the compassion requirements is to emulate the probabilistic choice at location ℓ_1. The compassion requirements are:

$$\begin{array}{ll}\{(\mathit{entered\_}\ell_{2,5} \ \wedge \ \mathit{cond}, & \mathit{entered\_}\ell_2 \ \wedge \ \mathit{cond}), \\ \ (\mathit{entered\_}\ell_{2,5} \ \wedge \ \mathit{cond}, & \mathit{entered\_}\ell_5 \ \wedge \ \mathit{cond})\}\end{array}$$

for each choice of *cond* taken from the following set:

$$\begin{array}{l}\{\mathit{rloc} \in \{0,8\},\ \mathit{rloc} \in \{2,3\},\ \mathit{rloc} = 4,\ \mathit{rloc} \in \{5,6\},\ \mathit{rloc} = 7, \\ \ \ \mathit{lloc} \in \{0,8\},\ \mathit{lloc} \in \{2,3\},\ \mathit{lloc} = 4,\ \mathit{lloc} \in \{5,6\},\ \mathit{lloc} = 7\}\end{array}$$

For a location ℓ_i, the predicate $\mathit{entered\_}\ell_i$ characterizes all states in which control has just entered ℓ_i. The above requirements guarantee, for each of the conditions *cond*, that if the choice at location ℓ_1 is taken infinitely many times while *cond* holds, then the computation proceeds infinitely many times from ℓ_1 to ℓ_2 while *cond* holds and infinitely many times from ℓ_1 to ℓ_5 while *cond* holds.

Our goal is to establish the accessibility property for the protocol, specified by: $\mathit{at\_}\ell_1[i] \Longrightarrow \Diamond\, \mathit{at\_}\ell_8[i]$ for every $n \geq 2$ and every $i = 1, \ldots, n$ The proof proceeds in several steps, described below.

No Process Can Get Stuck at Either ℓ_2 or ℓ_5

Consider the closed ring $(Q^n \circ) = (Q_1 \circ \cdots \circ Q_n \circ)$. First, we establish

$$\Box(\mathit{loc} = 5) \quad \Longrightarrow \quad \Diamond\Box(\mathit{rloc} = 5 \ \wedge \ \mathit{rlast} = 1) \tag{1}$$

for every Q_i within the ring. To establish this property, we consider the open composition $Q^2 = Q_1 \circ Q_2$ consisting of two composed processes in an unrestricted environment and model check property (1) for Q_1 within Q^2. From property (1) we conclude by induction the property

$$\Box\ \mathit{at\_}\ell_5[i] \quad \Longrightarrow \quad \forall j : [1..n] : \Diamond\Box(\mathit{at\_}\ell_5[j] \ \wedge \ \mathit{last}[j] = 1), \tag{2}$$

claiming that if process Q_i get stuck at ℓ_5 then, eventually all processes get stuck at ℓ_5 with $\mathit{last}[1] = \cdots = \mathit{last}[n] = 1$. We proceed to show that such a situation is impossible. This is due to the invariance of the assertion

$$\mathit{last}[i] \neq -1 \quad \rightarrow \quad \exists j \neq k : \mathit{last}[j] = 0 \ \wedge \ \mathit{last}[k] = 1 \tag{3}$$

The invariance of this assertion follows from the observation that the only transition which can modify the values of $\mathit{last}[i]$ is the exit from ℓ_8 and any execution of this transition sets $\mathit{last}[i] = Q_i.\mathit{last}$ to 0 and $\mathit{last}[i \oplus 1] = Q_i.\mathit{rlast}$ to 1. We conclude that no process can ever get stuck at ℓ_5. In a completely symmetric way we show that no process can ever get stuck at ℓ_2. This allows us to replace binary process Q by a process R which is identical to Q, except that the justice requirements associated with locations ℓ_2 and ℓ_5 are, respectively, $\mathit{loc} \neq 2$ and $\mathit{loc} \neq 5$.

Proving Accessibility for $(R^n \circ)$

We can now prove accessibility for the parametric system $(R_1 \circ \cdots \circ R_n \circ)$, using the network invariants method.

For the network invariant we use a *Two-Halves* abstraction $\mathcal{I} = \mathit{Left} \circ \mathit{Right}$, where process $\mathit{Left} = \mathit{Left}(\mathit{lloc}, \mathit{last}, \mathit{loc};\ \mathit{loc}, \mathit{rloc})$ is presented in Fig. 6. Process $\mathit{Right} = \mathit{Right}(\mathit{lloc}, \mathit{loc};\ \mathit{loc}, \mathit{rlast}, \mathit{rloc})$ is a mirror image of *Left* (see [12]), communicating with its right neighbor wherever *Left* communicates with its left neighbor. Process *Left* behaves like a regular philosopher with respect to its left fork, but its behavior with respect to the right fork is abstracted by non-deterministic steps (ℓ_3 and ℓ_5). To compensate for the non-determinism and ensure accessibility, the following compassion requirement is added to the process:

$$(\mathit{loc} \in \{2..4\} \ \wedge\ \mathit{rloc} \in \{5..7\}, \quad \mathit{loc} = 8 \ \vee\ \mathit{rloc} = 8)$$

Process *Right* is defined symmetrically.

$\mathit{Left}(\mathit{lloc}, \mathit{last}, \mathit{loc};\ \mathit{loc}, \mathit{rloc}) ::$

```
[ local last : [-1..1] init -1
  loop forever do
    [ ℓ0: Think
      ℓ1: go to {ℓ2, ℓ5}
      ℓ2: await lloc = 0 ∨ lloc ∈ {0..5} ∧ last ≠ 0
      ℓ3: go to {ℓ4, ℓ8}
      ℓ4: go to ℓ1
      ℓ5: go to {ℓ5, ℓ6}
      ℓ6: if lloc ∈ {0..5} then go to ℓ8
      ℓ7: go to ℓ1
      ℓ8: Eat;   last := 0; ]

  JUSTICE
  loc ≠ 1, loc ≠ 2, loc ≠ 3, loc ≠ 4,
  loc ≠ 5, loc ≠ 6, loc ≠ 7, loc ≠ 8
  COMPASSION
  (loc ∈ {2..4} ∧ rloc ∈ {5..7},   loc = 8 ∨ rloc = 8),
  (entered_ℓ2,5 ∧ cond,   entered_ℓ2 ∧ cond)
  (entered_ℓ2,5 ∧ cond,   entered_ℓ5 ∧ cond)
  for cond ∈ {lloc ∈ {0, 8}, lloc ∈ {2, 3}, lloc = 4, lloc ∈ {5, 6}, lloc = 7} ]
```

Fig. 6. Process *Left*

For the first step of establishing that $\mathcal{I}$ is a network invariant, we show that $(R_1 \circ R_2 \circ R_1) \sqsubseteq \mathit{Left} \circ \mathit{Right}$. We use the abstraction mapping α given by

$$\begin{array}{lll} \mathit{Left.lloc} = R_1.\mathit{lloc}, & \mathit{Left.last} = R_1.\mathit{last}, & \mathit{Left.loc} = R_1.\mathit{loc}, \\ \mathit{Right.last} = R_3.\mathit{last}, & & \\ \mathit{Right.loc} = R_3.\mathit{loc}, & \mathit{Right.rlast} = R_3.\mathit{rlast}, & \mathit{Right.rloc} = R_3.\mathit{rloc} \end{array}$$

With this abstraction mapping, it is not difficult to check that premises A1, A2 and A3 hold. In particular, the concrete processes have tests for the statements at locations $\ell_2, \ell_3, \ell_5, \ell_6$ which some of the abstract versions transform into non-deterministic choices. Most of the instances of premises A4 and A5 need not be checked because they are equivalent to their concrete counterparts. The only exception is the abstract compassion requirement

$$(\mathit{Left.loc} \in \{2..4\} \ \wedge \ \mathit{Right.loc} \in \{5..7\}, \quad \mathit{Left.loc} = 8 \ \vee \ \mathit{Left.loc} = 8)$$

which has no concrete counterpart. Consequently, we model check that the system $(R_1 \circ R_2 \circ R_3)$ satisfies

$$\square \diamondsuit (R_1.\mathit{loc} \in \{2..4\} \ \wedge \ R_3.\mathit{loc} \in \{5..7\}) \ \rightarrow \ \square \diamondsuit (R_1.\mathit{loc} = 8 \vee R_3.\mathit{loc} = 8)$$

Next, we have to show that $(R \circ \mathcal{I}) \sqsubseteq \mathcal{I}$. This task calls for establishing that $(R \circ \mathit{Left} \circ \mathit{Right}) \ \sqsubseteq \ (\mathit{Left} \circ \mathit{Right})$. The proof of this abstraction is similar to the previous one, using a similar abstraction mapping.

Finally, we conclude the verification by model checking the accessibility property

$$(R \circ \mathit{Left} \circ \mathit{Right} \circ) \quad \models \quad (R.\mathit{loc} = 1) \quad \Longrightarrow \quad \diamondsuit (R.\mathit{loc} = 8).$$

References

1. M. Abadi and L. Lamport. The existence of refinement mappings. *Theoretical Computer Science*, 82(2):253–284, May 1991.
2. K. R. Apt and D. Kozen. Limits for automatic program verification of finite-state concurrent systems. *Information Processing Letters*, 22(6), 1986.
3. M. Browne, E. Clarke, and O. Grumberg. Reasoning about networks with many finite state processes. *PODC'86*, pages 240–248.
4. E. Clarke, O. Grumberg, and S. Jha. Verifying parametrized networks using abstraction and regular languages. *CONCUR'95*, pages 395–407.
5. E. Dijkstra, W. Feijen, and A. van Gasteren. Derivation of a termination detection algorithm for disrtibued computations. *Info. Proc. Lett.*, 16:217–219, 1983.
6. E. Emerson and V. Kahlon. Reducing model checking of the many to the few. In *CADE-17*, pages 236–255, 2000.
7. E. Emerson and K. Namjoshi. Automatic verification of parameterized synchronous systems. *CAV'96*, LNCS 1102.
8. N. Halbwachs, F. Lagnier, and C. Ratel. An experience in proving regular networks of processes by modular model checking. *Acta Informatica*, 29(6/7):523–543, 1992.
9. C. Ip and D. Dill. Verifying systems with replicated components in Murφ. *CAV'96*, LNCS 1102.
10. Y. Kesten and A. Pnueli. Control and data abstractions: The cornerstones of formal verification. *Software Tools for Technology Transfer*, 2(4):328–342, 2000.

11. Y. Kesten and A. Pnueli. Verification by augmented finitary abstraction. *Information and Computation, a special issue on Compositionality*, 163:203–243, 2000.
12. Y. Kesten, A. Pnueli, E. Shahar, and L. D. Zuck. Network invariant in action. Technical report, The weizmann Institute of Science, 2002.
13. R. P. Kurshan and K. L. McMillan. A structural induction theorem for processes. *Information and Computation*, 117:1–11, 1995.
14. D. Lehmann and M. O. Rabin. On the advantages of free choice: A symmetric and fully distributed solution to the dining philosophers problem. *POPL'81*, pages 133–138.
15. D. Lesens, N. Halbwachs, and P. Raymond. Automatic verification of parameterized linear networks of processes. *POPL'97*.
16. Z. Manna, A. Anuchitanukul, N. Bjørner, A. Browne, E. Chang, M. Colón, L. D. Alfaro, H. Devarajan, H. Sipma, and T. Uribe. STeP: The Stanford Temporal Prover. Stanford, California, 1994.
17. Z. Manna and A. Pnueli. *The Temporal Logic of Reactive and Concurrent Systems: Specification*. Springer Verlag, New York, 1991.
18. Z. Manna and A. Pnueli. *Temporal Verification of Reactive Systems: Safety*. Springer-Verlag, New York, 1995.
19. A. Pnueli, S. Ruah, and L. Zuck. Automatic deductive verification with invisible invariants. *TACAS'01*, LNCS 2031, pages 82–97.
20. A. Pnueli and E. Shahar. A platform for combining deductive with algorithmic verification. *CAV'96*, LNCS 1102, pages 184–195.
21. A. Pnueli, J. Xu, and L. Zuck. Liveness with $(0, 1, \infty)$-counter abstraction. To appear in *CAV'02*.
22. A. Pnueli and L. D. Zuck. Probabilistic verification. *Information and computation*, 103(1):1–29, 1993.
23. A. Roychoudhury, K. N. Kumar, C. Ramakrishnan, I. Ramakrishnan, and S. Smolka. Verification of parameterized systems using logic-program transformations. *TACAS'00*.
24. A. Roychoudhury and I. Ramakrishnan. Automated inductive verification of parameterized protocols. *CAV'01*, LNCS 2102.
25. Z. Shtadler and O. Grumberg. Network grammars, communication behaviors and automatic verification. *CAV'89*, LNCS 407, pages 151–165.
26. A. Sistla and S. German. Reasoning about systems with many processes. *J. ACM*, 39:675–735, 1992.
27. P. Wolper and V. Lovinfosse. Verifying properties of large sets of processes with network invariants. *CAV'89*, LNCS 407, pages 68–80.
28. L. Zuck, A. Pnueli, and Y. Kesten. Automatic verification of free choice. In *Proc. of the 3^{rd} workshop on Verification, Model Checking, and Abstract Interpretation*, LNCS 2294, 2002.