Automatic Verification of Probabilistic Free Choice*

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

[1] New York University, New York zuck@cs.nyu.edu
[2] Weizmann Institute of Science, Rehovot, Israel amir@wisdom.weizmann.ac.il
[3] Ben-Gurion University, Beer-Sheva, Israel yKesten@bgumail.bgu.ac.il

Abstract. We study *automatic* methods for establishing P-validity (validity with probability 1) of *simple* temporal properties over finite-state probabilistic systems. The proposed approach replaces P-validity with validity over a non-probabilistic version of the system, in which probabilistic choices are replaced by non-deterministic choices constrained by *compassion* (strong fairness) requirements. "Simple" properties are temporal properties whose only temporal operators are $\diamondsuit$ (eventually) and its dual $\square$ (always). In general, the appropriate compassion requirements are "global," since they involve global states of the system. Yet, in many cases they can be transformed into "local" requirements, which enables their verification by model checkers. We demonstrate our methodology of translating the problem of P-validity into that of verification of a system with local compassion requirement on the "courteous philosophers" algorithm of [LR81], a parameterized probabilistic system that is notoriously difficult to verify, and outline a verification of the algorithm that was obtained by the TLV model checker.

1 Introduction

Probabilistic elements have been introduced into concurrent systems in the early 1980s to provide solutions (with high probability) to problems that do not have deterministic solutions. Among the pioneers of probabilistic protocols were ([LR81,Rab82]). One of the most challenging problems in the study of probabilistic protocols has been their formal verification. While methodologies for proving safety (invariance) properties still hold for probabilistic protocols, formal verification of their liveness properties has been, and still is, a challenge. The main difficulty stems from the two types of nondeterminism that occur in such programs: Their asynchronous execution, that assumes a hostile (though somewhat fair) scheduler, and the nondeterminism associated with the probabilistic actions, that assumes an even-handed scheduler.

It had been realized that if one only wants to prove that a certain property is *P-valid*, i.e., holds with probability 1 over all executions of a system, this

* 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.

can be accomplished, for finite-state systems, in a manner that is completely independent of the precise probabilities. Decidability of P-validity had been first established in [HSP82] for termination properties over finite-state systems, using a methodology that is graph-theoretic in nature. The work in [PZ86b] extends the [HSP82] method and presents deductive proof rules for proving P-validity for termination properties of finite-state program. The work in [PZ86a,PZ93] presents sound and complete methodology for establishing P-validity of general temporal properties over probabilistic systems, and [VW86,PZ86a,PZ93] describe model checking procedure for the finite-state case.

The emerging interest in embedded systems brought forth a surge of research in automatic verification of parameterized systems, that, having unbounded number of states, are not easily amendable to model checking techniques. In fact, verification of such systems is known to be undecidable [AK86]. Much of the recent research has been devoted to identifying conditions that enable automatic verification of such systems, and abstraction tools to facilitate the task (e.g., [KP00,APR$^+$01,EN95,EN96,EK00,PRZ01].)

Many of the probabilistic protocols that have been proposed and studied (e.g., [LR81,Rab82,PZ86b,CLP84]) are parameterized. An obvious question is therefore whether we can combine verification tools of parameterized systems with those of probabilistic ones. The work in [PZ86b] provides several examples of deductive verification of *parameterized* probabilistic systems, including the *free philosophers* algorithm of [LR81] that guarantees livelock freedom of the system. A verification of the more complex *courteous philosophers* algorithm of [LR81] is in [L85], using a methodology that cannot be automated.

The main additional difficulty encountered when verifying probabilistic programs is "probabilistic fairness" – the fairness requirement over computations that suffices to replace measure-theoretic considerations. In this paper we study the problem of *automatic* verification of P-validity of probabilistic systems using a method that is also applicable to the verification of parameterized probabilistic systems. We show how, for the case of *simple* temporal properties, probabilistic systems can be translated into non-probabilistic systems by replacing the probabilistic fairness with *compassion* (strong fairness.) "Simple" properties are temporal properties whose only temporal operator are $\diamondsuit$ (eventually) and its dual $\square$ (always). While this, of course, impairs the expressive power of the properties proven, it encompasses almost all properties of probabilistic protocols that have been studied. The compassion requirements obtained, however, are many and global (i.e., are with respect to global states.) Consequently, the systems obtained are not easily accommodated by most model checkers, that expect compassion requirements to be local. In many cases, and almost all of those we have studied, it is possible to transform the new compassion requirements into few local ones.

We demonstrate our methodology of translating the problem of P-validity into that of verification of a system with local compassion requirement on the "courteous philosopher" algorithm of [LR81], a parameterized probabilistic system that is notoriously difficult to verify. We describe the automatic verification of the algorithm obtained using TLV [PS96], the Weizmann Institute's programmable model checker.

2 Fair Discrete Systems

As a computational model for reactive systems we take the model of *fair discrete systems* (FDS) [KP00], which is a slight variation on the model of *fair transition system* [MP95]. Under this model, a system $\mathcal{S} : \langle V, \mathcal{O}, W, \Theta, \rho, \mathcal{J}, \mathcal{C} \rangle$ 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 set of 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 an ρ-successor state $s' \in \Sigma$.
- $\mathcal{J}$: A set of *justice* (*weak fairness*) *requirements*. The justice requirement $J \in \mathcal{J}$ is an assertion, intended to guarantee that every computation contains infinitely many J-states (states satisfying J).
- $\mathcal{C}$: 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.

Let $\sigma\colon s_0, s_1, s_2, \ldots,$ be an infinite sequence of states, φ be an assertion, and let $j \geq 0$ be a natural number. We say that j is a φ-position of σ if s_j is a φ-state.

Let $\mathcal{S}$ be an FDS for which the above components have been identified. We define an (*open*) *computation* of $\mathcal{S}$ 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}.

- *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.

According to this definition, system and environment steps strictly interleave. Since both the system and environment allow stuttering steps, this is not a serious restriction.

For an FDS $\mathcal{S}$, we denote by $\mathcal{C}omp(\mathcal{S})$ the set of all computations of $\mathcal{S}$. A *property* is a (next- and previous-free) propositional linear time temporal logic, possibly including past operators, over the states of $\mathcal{S}$. A property φ is *valid* over $\mathcal{S}$ is $\sigma \models \varphi$ for every $\sigma \in Comp(\mathcal{S})$.

Systems $\mathcal{S}_1$ and $\mathcal{S}_2$ are *compatible* if their sets of owned variables are disjoint, and the intersection of their variables is observable in both systems. For compatible systems $\mathcal{S}_1$ and $\mathcal{S}_2$, the *parallel composition* of $\mathcal{S}_1$ and $\mathcal{S}_2$, denoted by $\mathcal{S}_1 \,\|\, \mathcal{S}_2$, is 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 new system is a step of system $\mathcal{S}_1$, or a step of system $\mathcal{S}_2$, or an environment step.

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

All our concrete examples are given in SPL (Simple Programming Language), which is used to represent concurrent programs (e.g., [MP95,MAB$^+$94]). 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

$$\ell_0\colon y := x + 1;\ \ell_1\colon$$

can be executed when control is at location ℓ_0. When executed, it assigns $x+1$ to y while control moves from ℓ_0 to ℓ_1. This statement contributes to ρ the disjunct

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

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.

3 Parameterized Systems and Their Verification

A *parameterized FDS* is a system $\mathcal{S}(N) = P[1] \,\|\, \ldots \,\|\, P[N]$, where the $P[i]$'s are symmetric SPL programs. For each value of $N > 0$, $\mathcal{S}(N)$ is an instantiation of an FDS. We are interested in properties that hold for every process in the system. Because of symmetry, we can express them in terms of one process, say $P[1]$. Thus, we are interested in properties of the type $\varphi(1)$, where $\varphi(1)$ is a temporal formula referring only to variables that are known to $P[1]$. The problem of parameterized verification is to show that $\varphi(1)$ is valid (P-valid) over $\mathcal{S}(N)$ *for every* N. A similar situation exists if we are interested in properties, such as mutual exclusion, which involve two contiguous processes. In this case, we can test these properties by checking whether a property $\psi(1,2)$ holds for the specific processes $P[1]$ and $P[2]$, for every value of $N > 2$.

Parametric verification is known to be undecidable (see, e.g., [AK86]). Recent research has focused on methodologies to identify systems and properties for which the problem of parametric verification is decidable, and, for these systems, to provide for semi- or fully- automatic verification.

One of the main ideas that have been proposed is to identify some number, say N_0, such that validity of $\varphi(1)$ over $\mathcal{S}[N']$ for every $N' \leq N_0$ suffices to establish its validity for all $N' \geq N$ [APR+01,EN95,EN96,EK00,PRZ01].

To prove the liveness property of a parameterized system, we propose a variant of the network invariant strategy of [KP00] (see also [WL89,BCG86,CGJ95], [KM95]). The approach is described by:

1. Divine a *network invariant* $\mathcal{I}$ which is an FDS intended to provide an abstraction for the parallel composition of $P_2 \| \cdots \| P_n$ for any $n \geq c$ for some small constant c.
2. Confirm that $\mathcal{I}$ is indeed a network invariant, by verifying that $P_2 \sqsubseteq \mathcal{I}$ and that $(\mathcal{I} \| P_2) \ \sqsubseteq \ \mathcal{I}$.
3. Model check $P_1 \| \mathcal{I} \models p$.
4. Conclude that $S(N) \models p$ for every $N > 1$.

The crucial step in establishing that a candidate $\mathcal{I}$ is a good network invariant is in proving the refinement ($\sqsubseteq$) relation between systems. Usually, the abstract system has significantly more non-determinism than the concrete system. Thus, showing the every concrete step has a unique abstract step that maps to it (and preserves the value of the observable variables) may be quite complicated. Indeed, it has been our experience that a significant part of the effort of proving refinement is devoted to "guiding" TLV to find the abstract step that maps to a concrete step.

4 Example: Deterministic Dining Philosophers

The purpose of this example is twofold: To show a parameterized system and outline its verification, and to present the problem of the dining philosophers that we look at more carefully in Section 6.

Assume there are $N > 2$ processes (philosophers) arranged in a ring (sitting around a table) numbered counter-clockwise $P_1, \ldots, P_N$. Let $i \oplus 1 = (i \bmod N) + 1$ and $i \ominus 1 = (i - 2 \bmod N) + 1$. These definitions lead to the facts that $N \oplus 1 = 1$

and $1 \ominus 1 = N$. Every two adjacent philosophers, P_i and $P_{i\oplus 1}$, share a common fork, $y[i \oplus 1]$. A description of a typical portion of the table is in Fig. 1. Philosophers spend most of their lives thinking (non-critical), however, occasion-

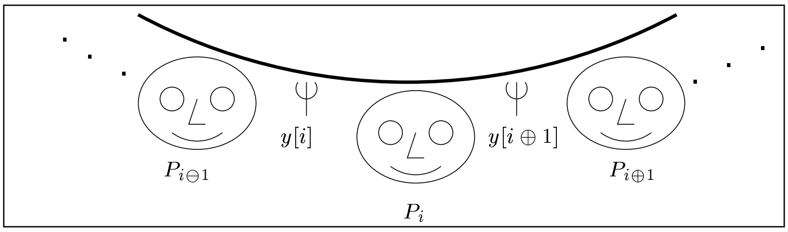

Fig. 1. Part of The Table

ally a philosopher may become hungry. In order to eat, a philosopher needs to obtain both its adjacent forks. A solution to the dining philosophers problem is a program for the philosophers, that guarantees that no two adjacent philosophers eat simultaneously, and that every hungry philosopher eventually eats. It is well known that if the system is fully symmetric, there are no deterministic solutions to the problem. An almost symmetric solution, using semaphores for the forks, is presented in the SPL program described in Fig. 2. The system is "almost symmetric" since all processes, but one, follow the same protocol, while the singled out process ($P[1]$) follows a different protocol: A "regular' philosopher $P[i]$, $i > 1$, reaches first for its left fork and then for its right fork. The "contrary" philosopher $P[1]$ reaches first for its right fork and then for its left fork.

$$
\begin{array}{l}
\quad\quad\quad\quad\quad\quad \textbf{in} \quad n : \textbf{integer where } n \geq 2 \\
\quad\quad\quad\quad\quad\quad \textbf{local } c : \textbf{array } [1..n] \textbf{ where } c = 1 \\
\\
\left\|_{j=2}^{n}\right. P[j] :: \left[\begin{array}{l} \ell_0 : \textbf{loop forever do} \\ \left[\begin{array}{l} \ell_1 : \textbf{NonCritical} \\ \ell_2 : \textbf{request } y[j] \\ \ell_3 : \textbf{request } y[j \oplus 1] \\ \ell_4 : \textbf{Critical} \\ \ell_5 : \textbf{release } y[j] \\ \ell_6 : \textbf{release } y[j \oplus 1] \end{array}\right] \end{array}\right] \;\Big\|\; P[1] :: \left[\begin{array}{l} \ell_0 : \textbf{loop forever do} \\ \left[\begin{array}{l} \ell_1 : \textbf{NonCritical} \\ \ell_2 : \textbf{request } y[2] \\ \ell_3 : \textbf{request } y[1] \\ \ell_4 : \textbf{Critical} \\ \ell_5 : \textbf{release } y[2] \\ \ell_6 : \textbf{release } y[1] \end{array}\right] \end{array}\right]
\end{array}
$$

Fig. 2. A deterministic solution with one contrary philosopher.

The liveness property of the system is:

$$\square(at\_\ell_2[1] \;\rightarrow\; \diamondsuit(at\_\ell_4[1])),$$

We now outline two different (and successful) network invariant strategies used for obtaining the liveness property of the protocol. In both, we view each regular philosopher as a system $P(\mathit{left}, \mathit{right})$, where the semaphores *left* and *right* are the only observables. We seek an invariant $\mathcal{I}(\mathit{left}, \mathit{right})$ which is an abstraction of the philosophers chain

$$S[k] :: \left[\begin{array}{l} \textbf{local } f : \textbf{array}[2..k] \textbf{ of boolean where } f = 1 \\ P(\mathit{left}, f[2]) \parallel P(f[2], f[3]) \parallel \cdots \parallel P(f[k], \mathit{right}) \end{array}\right]$$

for every $k \geq 2$. This means that any $(\mathit{left}, \mathit{right})$-observation of $S[k]$ is matched by a corresponding $(\mathit{left}, \mathit{right})$-observation of $\mathcal{I}(\mathit{left}, \mathit{right})$.

Abstraction 1: The "*two-halves*". Observing how the two border members of $S[k]$ manipulate the obesrvables *left* and *right*, led, after experimentation to an abstraction consisting of the composition of a *left half* philosopher and a *right half* philosopher, as presented in Fig. 3.

$$L(\mathit{left}) :: \left[\begin{array}{l} \ell_0 : \textbf{loop forever do} \\ \left[\begin{array}{l} \ell_1 : \textbf{NonCritical} \\ \ell_2 : \textbf{request } \mathit{left} \\ \ell_3 : \textbf{idle} \\ \ell_4 : \textbf{Critical} \\ \ell_5 : \textbf{release } \mathit{left} \\ \ell_6 : \textbf{skip} \end{array}\right] \end{array}\right] \parallel R(\mathit{right}) :: \left[\begin{array}{l} m_0 : \textbf{loop forever do} \\ \left[\begin{array}{l} m_1 : \textbf{NonCritical} \\ m_2 : \textbf{idle} \\ m_3 : \textbf{request } \mathit{right} \\ m_4 : \textbf{Critical} \\ m_5 : \textbf{skip} \\ m_6 : \textbf{release } \mathit{right} \end{array}\right] \end{array}\right]$$

$$\text{Extra Compassion: } (\mathit{at\_}\ell_3 \ \wedge\ \mathit{right} > 0,\ \mathit{at\_}\ell_4)$$

Fig. 3. The two-halves Network Invariant

The additional compassion requirement reflects the possibility that the leftmost process in $S[2]$ may only deadlock while requesting its right fork (at ℓ_3) if the rightmost process can eventually always holds on to its right fork. To show that an arbitrary regular philosopher never starves, it is suffices to verify

$$(\mathcal{I} \parallel P \parallel \mathcal{I} \parallel R) \quad \models \quad (\mathit{at\_}\ell_2 \ \rightarrow \ \Diamond\, \mathit{at\_}\ell_4)$$

where R is a contrary philosopher and the locations in the property refer to P.

Abstraction 2: The "*four-by-three*". An alternate simpler invariant can be obtained by taking $\mathcal{I} = S[3]$, i.e. a chain of 3 (unmodified) philosophers.
To prove that this is an invariant, it is sufficient to establish

$$(p[1] \parallel p[2] \parallel p[3] \parallel p[4]) \ \sqsubseteq (P[5] \parallel P[6] \parallel P[7])$$

i.e., that 3 philosophers can faithfully emulate 4 philosophers.

This is established by letting $P[5]$ mimic $p[1]$ and $P[7]$ mimic $p[4]$. As to $P[6]$, it can remain idle until it finds out that $S[4]$ is 2 (internal) steps away from a *guaranteed deadlock* (all of $p[1], \ldots, p[4]$ remain stuck at location ℓ_2), at which point $P[6]$ joins $P[5]$ and $P[7]$ in order to form a similar deadlock at the abstract level. This requires the capability of clairvoyance, which has been implemented within TLV.

5 Adding To and Removing Probabilities From FDSs

We describe the formal model for probabilistic discrete systems (PDS) and P-validity. We then show how, when establishing P-validity of simple properties, PDS's can be translated into FDSs.

5.1 PDS: Adding Probabilities to FDSs

A *probabilistic discrete system* (PDS) $\mathcal{S}\colon \langle V, \mathcal{O}, W, \Theta, \rho, \mathcal{J}, \mathcal{C}, \mathcal{P}\rangle$ consists of an FDS $\langle V, \mathcal{O}, W, \Theta, \rho, \mathcal{J}, \mathcal{C}\rangle$ and a *probabilistic fairness condition* $\mathcal{P}$ containing tuples of the form $\langle p;\ \alpha_1 : q_1, \ldots, \alpha_n : q_n\rangle$, where $q_1, \ldots, q_n$ are mutually disjoint state assertions, and $\sum_{i=1}^{n} \alpha_i = 1$. It is also required that p is disjoint of any of the q_i's. Intuitively, the meaning of a probabilistic fairness condition $\langle p;\ \alpha_1 : q_1, \ldots, \alpha_n : q_n\rangle$ is that whenever the system moves from a p-state into a state satisfying $\bigvee_{i=1}^{n} q_i$, it moves into a q_i-state with probability α_i. Let s be a p-state. It is required that s has precisely $n+1$ successors, s (itself) and $s_1, \ldots, s_n$, where s_i satisfies q_i, for every $i = 1, \ldots, n$.

The following definition applies to a PDS with a single probabilistic tuple $\langle p;\ \alpha_1 : q_1, \ldots, \alpha_n : q_n\rangle$. Its generalization to systems with more than one such tuple is straightforward.

An (open) *computation tree* of a PDS is formed as follows:

- The root of the tree is any state which is initial, i.e. satisfies Θ.
- Every node s at level $2j$ has a single descendant which is an environment successor of s,i.e. a state $\widetilde{s}$ such that $\widetilde{s}[W] = s[W]$.
- Every node s at level $2j+1$ which is not a p-state, has a single descendant which is a ρ-successor of s.
- Every node s at level $2j+1$ which is a p-state has either itself s as a single descendant, or the n descendants $s_1, \ldots, s_n$.

Such a tree induces a probability measure over all the infinite paths that can be traced in the tree, where each edge from s to s_i is assigned the probability a_i. A computation tree is called *admissible* if the measure of paths which are just and compassionate is 1.

Following [PZ93], we say that a temporal property φ is *P-valid* over a computation tree T_σ if the measure of paths in that satisfy φ is 1. (See [PZ93] for a detailed description and definition of the measure space.) Similarly, φ is *P-valid over the PDS $\mathcal{S}$* if it is P-valid over every admissible computation tree of $\mathcal{S}$.

Much work has been devoted to replacing the measure space required in the definition of P-validity by "simpler" notion of fairness. In particular, we have been searching for a definition of "x-fairness" of computation, such that φ would be P-valid iff it is satisfied by every x-fair computation of $\mathcal{S}$. Such is the α-fairness of [PZ93]: For a past temporal logic formula χ, a computation $\sigma = s_0, \ldots$ is *α-fair with respect to* χ if, for every probabilistic fairness condition $\langle p; \alpha_1 : q_1, \ldots, \alpha_n : q_n\rangle$,

$$s_0, \ldots, s_j \models \chi \ \wedge\ p \quad \text{and} \quad s_{j+1} \models q_1 \vee \ldots \vee q_n \qquad \text{for infinitely many } j\text{'s}$$

implies

$$s_0, \ldots, s_j \models \chi \wedge p \quad \text{and} \quad s_{j+1} \models q_\ell \qquad\qquad \text{for infinitely many } j\text{'s}$$

for every $\ell = 1, \ldots, n$.

In other words, if the computation reaches a "p to $\bigvee q_i$" transition infinitely many times from χ-prefices, then each mode of the transition should be taken infinitely many times from χ-prefices. A computation if *α-fair* iff it is α-fair with respect to *every* past temporal formula χ. A result of [PZ93] is:

Theorem 1. *A temporal property φ is P-valid over $\mathcal{S}$ iff every α-fair computation of $\mathcal{S}$ satisfies φ.*

While α-fairness is sound and complete, it is hardly satisfactory, since it calls for establishing α-fairness with respect to "every past formula." The work in [PZ93] also presents a model checking procedure of finite state PDSs against temporal specification that do not have the temporal operators $\bigcirc$ and $\mathcal{U}$. That is, the temporal properties whose P-validity is established in the model checking can include all the past operators, but the only future operator they can have is $\diamondsuit$ (and its dual, $\square$). The model checking procedure there involves constructing the closure of the (negation of the) property, building an atom graph where each atom node is a maximal logically-consistent subset of formulae in the closure that correspond to program states, and nodes are connected if they do so in both the tableau of the property and the program itself.

A careful examination of this model checking procedure reveals that in order to establish P-validity over finite state PDSs and $\{\bigcirc, \mathcal{U}\}$-less properties, it suffices to consider computations that are α-fair only with respect to every past formula that appears in the closure of the property. Consequently, we have:

Corollary 1. *Given a finite-state PDS $\mathcal{S}$ and a $\{\bigcirc, \mathcal{U}\}$-less property φ. Then φ is P-valid over $\mathcal{S}$ iff for every past formula χ appearing the in the closure of φ, every $\mathcal{S}$-computation that is α-fair with respect to χ satisfies φ.*

5.2 Removing Probabilities from PDSs

Consider *simple temporal properties* that do no include any of the past, or the future $\bigcirc$ and $\mathcal{U}$ operators. Thus, simple temporal properties include, as their only temporal operators, $\diamondsuit$ and it dual, $\square$. The work in [SZ93] includes an extensive study of this class.

While at first glance it may seem that this class of simple temporal properties is extremely restrictive, it is actually a rather inclusive class, since it accommodates all the safety, and most of the progress properties one usually wants to prove about PDSs. E.g., mutual exclusion usually has the form $\square(\text{trying}_i \rightarrow \diamondsuit\, \text{critical}_i)$ which is a simple temporal property.

The closure of a simple temporal property includes no past formulae. Consequently, we can replace α-fairness by fairness with respect to every state assertion. We call this notion of fairness γ-fairness. Formally, a $\mathcal{S}$-computation σ is

γ-fair if it is α-fair with respect to every state assertion. Note that for a finite-state system, the relevant state assertions are the states themselves (in fact, the γ comes from "global", since this is fairness with respect to the global states.)

We can therefore conclude from Corollary 1:

Corollary 2. *Given a finite PDS $\mathcal{S}$ and a simple temporal property φ. Then φ is P-valid over $\mathcal{S}$ iff every $\mathcal{S}$-computation that is γ-fair satisfies φ.*

Corollary 2 implies that, in order to prove that a simple temporal property is P-valid over a finite PDS, it suffices to prove that it is satisfied over all (just and compassionate) computations of the system where if the probabilistic choice is made infinitely many times from a given state, then each outcome of that probabilistic choice should be taken from that state. Hence, each probabilistic fairness condition $\langle p; \alpha_1 : q_1, \ldots, \alpha_n : q_n \rangle$ can be translated into a *set of compassion* requirements that includes every pair of states $\langle s, s' \rangle$ such that s is a p-state and s' is a q_i-state for some i. For a set $\mathcal{P}$ of probabilistic fairness requirements, denote by $\mathcal{C}(\mathcal{P})$ the set of compassion properties obtained by replacing each condition in $\mathcal{P}$ by the corresponding set of compassion requirements. We then have:

Theorem 2. *Given a finite PDS $\mathcal{S} : \langle V, \Theta, \rho, \mathcal{J}, \mathcal{C}, \mathcal{P} \rangle$. Let $\mathcal{S}' : \langle V, \Theta, \rho, \mathcal{J}, \mathcal{C} \cup \mathcal{C}(\mathcal{P}) \rangle$ be the FDS obtained from $\mathcal{S}$ by translating the probabilistic fairness conditions of $\mathcal{S}$ into compassion requirements. Then for every simple temporal property φ,*

$$\varphi \text{ is P-valid over } \mathcal{S} \qquad \text{iff} \qquad \varphi \text{ is valid over } \mathcal{S}'$$

While Theorem 2 implies that PDSs can be translated into FDSs, a straightforward application of the idea may lead to systems that are not manageable by current model checkers. The reason for this is that the state assertions appearing in the probabilistic fairness conditions are usually local, while the assertions appearing in the new compassion requirements are usually global. For example, in the probabilistic fairness condition, p is usually of the form $at\_\ell_x[i]$ and the q_j's are of the form $at\_\ell_{y_j}[i]$, stating that "from any global state where $P[i]$ is about to take a probabilistic choice whose outcomes are to move it from location x to locations $y_1, \ldots, y_n$, each with a positive probability, it should reach each of these locations from that global state infinitely many times." Each compassion requirement $\langle s, s'_j \rangle$ we obtain has on the left one of the global states where $P[i]$ is in location x, and on the right a set of states where it is in location y_j. We cannot combine the left-hand-side s-s into a single compassion requirement, since this has the undesirable effect of allowing a computation where, e.g., $P[i]$ always gets to y_1 from certain states, and always to y_2 from others. Thus, unless somehow manipulated, we will end up with too many compassion requirements that are global, both are undesirable properties for the purpose of model checking. Note that the situation gets completely out of hand when dealing with parameterized systems. There, the number of compassion requirements one may end up with if not careful is exponential in the size of the code of a single process.

Consequently, a crucial step in establishing P-validity of simple properties using existing model checking technique is then to "localize" the compassion requirements and to minimize their number. While we have no general methodology of doing that, we succeeded to do it for many interesting cases. The most complex case is described in the next section.

6 The [LR81] Dining Philosophers Protocol

In Lehmann and Rabin's *Courteous Philosophers Algorithm* the forks are shared variables that are set when held and reset when on the table. In addition to the forks, adjacent philosophers share a $lastL[i \oplus 1]$ variable, initially -1, and after one of the them eats, denoting whether it's the left (P_i) philosopher that last ate or the right. Each philosopher P_i has additional boolean variables (written by it and read by its immediate neighbours), $signR[i]$ which denotes its wish to eat to its left neighbor ($P_{i \ominus 1}$) and $signL[i]$ which denotes its wish to eat to its right neighbor ($P_{i \oplus 1}$).

The algorithm is a refinement of the "Free Philosophers" algorithm, in the same paper, were each hungry philosopher chooses randomly whether to wait to its left or right fork first, and, after (and if) it obtains it, waits until the other fork is available. The Free Philosophers are guaranteed, however, only that eventually *some* philosopher eats. The Courteous Philosophers are similar to the free ones, the difference being that a courteous philosopher can pick up its first fork only if its partner (on that side) is either not hungry or is the last to have eaten between the two of them. An SPL code of the protocol is described in Fig. 4.[1]

The justice requirements and probabilistic fairness conditions of the system are the obvious ones, and there are no compassion requirements. Since the property we want to establish is the liveness property

$$\square(at\_\ell_1[1] \;\rightarrow\; \diamondsuit\, at\_\ell_8[1]),$$

which is a simple property, we proceed to translate the probabilistic fairness into compassion requirements.

A naive replacement of the probabilistic fairness properties by compassion will lead to roughly 12^N global compassion requirements which is unacceptable. To minimize and localize the requirements, we employed a combination of studying the system (and its deductive proof in [L85]) and experimentation with proving the liveness property for $N = 3, 4$ using TLV. The chain of reductions we went through is as follows.

The deductive proof focuses, at each step, only on two adjacent processes. It seemed therefore reasonable to localize the compassion, and to require it from each processes only with respect to its immediate neighbours. This led to including in the compassion set of each process i the requirements:

$$\langle at\_\ell_1[i] \wedge cond,\; at\_\ell_2[i] \wedge cond\rangle, \quad \langle at\_\ell_1[i] \wedge cond,\; at\_\ell_5[i] \wedge cond\rangle$$

[1] In the protocol, as presented in [LR81], the instructions appearing in lines 9–11 are not atomic. Making them atomic, as we did in our presentation doesn't 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.

$$
\begin{array}{l}
\qquad\qquad \textbf{in} \quad N: \qquad\qquad\qquad \textbf{integer where } N \geq 2 \\
\qquad\qquad \textbf{local } \mathit{signL}, \mathit{signR}, y, : \textbf{array } [1..N] \textbf{ of boolean init } 0 \\
\qquad\qquad \textbf{local } \mathit{lastL}: \qquad\qquad \textbf{array } [1..N] \textbf{ of } \{-1,0,1\} \textbf{ init } -1 \\
\overset{N}{\underset{i=1}{\Big|\Big|}} P[i] :: \left[\begin{array}{l}
\textbf{loop forever do} \\
\left[\begin{array}{ll}
\ell_0: & \textbf{non-critical} \\
\ell_1: & \mathit{signL}[i] := 1; \mathit{signR}[i] := 1; \textbf{goto } \{0.5: \ell_2; 0.5: \ell_5\} \\
\ell_2: & \textbf{await } \neg y[i] \ \wedge \ (\neg \mathit{signR}[i \ominus 1] \ \vee \ \mathit{lastL}[i] = 1) \\
 & \quad \textbf{and then } y[i] := 1 \\
\ell_3: & \textbf{If } y[i \oplus 1] = 0 \\
 & \qquad \textbf{then } y[i \oplus 1] := 1; \textbf{ goto } \ell_8 \\
\ell_4: & y[i] := 0; \ \textbf{goto } \ell_1 \\
\ell_5: & \textbf{await } \neg y[i \oplus 1] \ \wedge \ (\neg \mathit{signL}[i \oplus 1] \ \vee \ \mathit{lastL}[i \oplus 1] = 0) \\
 & \quad \textbf{and then } y[i \oplus 1] := 1 \\
\ell_6: & \textbf{If } y[i] = 0 \\
 & \qquad \textbf{then } y[i] := 1; \textbf{ goto } \ell_8 \\
\ell_7: & y[i \oplus 1] := 0; \ \textbf{goto } \ell_1 \\
\ell_8: & \textbf{Critical} \\
\ell_9: & \mathit{signL}[i] := 0; \mathit{signR}[i] := 0 \\
\ell_{10}: & \mathit{lastL}[i] := 0, \mathit{lastL}[i \oplus 1] := 1 \\
\ell_{11}: & y[i] := 0; y[i \oplus 1] := 0
\end{array}\right]
\end{array}\right]
\end{array}
$$

Fig. 4. The Courteous Philosophers

for every

$$
\mathit{cond} \in \left\{ \begin{array}{l} at\_\ell_{8..11,0..1}[i \ominus 1], at\_\ell_{2,3}[i \ominus 1], at\_\ell_4[i \ominus 1], at\_\ell_{5,6}[i \ominus 1], at\_\ell_7[i \ominus 1] \\ at\_\ell_{8..11,0..1}[i \oplus 1], at\_\ell_{2,3}[i \oplus 1], at\_\ell_4[i \oplus 1], at\_\ell_{5,6}[i \oplus 1], at\_\ell_7[i \oplus 1] \end{array} \right\}
$$

The process of deriving small and local compassion sets is non-algorithmic in nature, however, the result can always be automatically verified.

To provide an automatic proof of the liveness property of the protocol, we first reduced the state space, by eliminating the variables $y[i], \mathit{signL}[i], \mathit{signR}[i]$, whose values can be uniquely determined by the locations of the relevant processes. We also compressed all the actions performed in locations $\ell_8 - \ell_{11}$ into a single statement labeled ℓ_8. All of these reductions do not alter significantly the behavior of the processes but simplify its verification. This leads to the protocol described in Fig. 5.

Using TLV we established the property

$$
\square \, at\_\ell_5[i] \quad \Longrightarrow \quad \diamondsuit \, \square (at\_\ell_5[i \oplus 1] \ \wedge \ \mathit{lastL}[i \oplus 1] = 1) \tag{1}
$$

for every i. From this, by induction around the philosophers ring, we can show that if one process gets stuck at ℓ_5 then all processes eventually get stuck at ℓ_5, with $\mathit{lastL}[1] = \cdots = \mathit{lastL}[N] = 1$. Since the only statement which modifies any of the $\mathit{lastL}[i]$ variables is ℓ_8, which sets $\mathit{lastL}[i]$ to 0, and $\mathit{lastL}[i \oplus 1]$ to 1, we conclude that the situation $\mathit{lastL}[1] = \cdots = \mathit{lastL}[N] = 1$ is unreachable. Therefore, no process can get stuck at location ℓ_5. In a symmetric way, we can show that no process ever gets stuck at location ℓ_2.

$$\begin{array}{l}
\begin{array}{ll} & \textbf{in} \quad N: \quad \textbf{integer where } N \geq 2 \\ & \textbf{local } \mathit{lastL} : \textbf{array } [1..N] \textbf{ of } \{-1,0,1\} \textbf{ init } -1 \end{array} \\
\overset{N}{\underset{i=1}{\Big|\Big|}} P[i] :: \left[\begin{array}{l} \textbf{loop forever do} \\ \left[\begin{array}{l}
\ell_0 : \textbf{non-critical} \\
\ell_1 : \textbf{goto } \{0.5 : \ell_2; 0.5 : \ell_5\} \\
\ell_2 : \textbf{await } at_\ell_0[i \ominus 1] \ \vee \ at_\ell_{0..5}[i \ominus 1] \wedge (\mathit{lastL}[i] \neq 0) \\
\ell_3 : \textbf{if } at_\ell_{1,2,5..7}[i \oplus 1] \textbf{ then go to } \ell_8 \\
\ell_4 : \ \textbf{go to } \ell_1 \\
\ell_5 : \textbf{await } at_\ell_0[i \oplus 1] \ \vee \ at_\ell_{1,2,5..7}[i \oplus 1] \wedge (\mathit{lastL}[i] \neq 1) \\
\ell_6 : \textbf{if } at_\ell_{0..5}[i \ominus 1] \textbf{ then go to } \ell_8 \\
\ell_7 : \ \textbf{go to } \ell_1 \\
\ell_8 : \textbf{Critical}; \ \mathit{lastL}[i] := 0; \ \mathit{lastL}[i \oplus 1] := 1
\end{array}\right] \end{array}\right]
\end{array}$$

Fig. 5. Location-based Courteous Philosophers

This allows us to add the justice properties $\neg at\_\ell_2[i], \neg at\_\ell_5[i]$ to the justice set of each process. Thus, from now on, we restrict our attention to *progressive philosophers* which are guaranteed not to get stuck at either ℓ_2 or ℓ_5.

Next, we follow the ideas developed in Section 4 and view each philosopher $P[i]$ as a system $P(\mathit{lloc}, \mathit{cloc}, \mathit{rloc}, \mathit{clst}, \mathit{rlst})$, whose observables are, respectively, $\mathit{lloc} = P[i \ominus 1].\mathit{loc}$, $\mathit{cloc} = P[i].\mathit{loc}$, $\mathit{rloc} = P[i \oplus 1].\mathit{loc}$, $\mathit{clst} = \mathit{lastL}[i]$, and $\mathit{rlst} = \mathit{lastL}[i \oplus 1]$. We seek an invariant $\mathcal{I}(\mathit{elloc}, \mathit{lloc}, \mathit{rloc}, \mathit{erloc}, \mathit{llst}, \mathit{erlst})$ which is an abstraction of the following philosophers chain $S[k]$:

$$\left[\begin{array}{l}
\begin{array}{lll}
\textbf{in} & \mathit{elloc}, \mathit{erloc} : & [0..8] \\
\textbf{in-out} & \mathit{llst}, \mathit{erlst} : & [-1..1] \textbf{ where } \mathit{llst}, \mathit{erlst} = -1 \\
\textbf{out} & \mathit{lloc}, \mathit{rloc} : & [0..8] \textbf{ where } \mathit{lloc}, \mathit{rloc} = 0 \\
\textbf{local} & \mathit{loc} : & \textbf{array}[2..k-1] \textbf{ of } [0..8] \textbf{ where } \mathit{loc} = 0 \\
 & \mathit{lastL} : & \textbf{array}[2..k] \textbf{ of } [-1..1] \textbf{ where } \mathit{lastL} = -1
\end{array} \\
P(\mathit{elloc}, \mathit{lloc}, \mathit{loc}[2], \mathit{llst}, \mathit{lastL}[2]) \parallel \cdots \parallel P(\mathit{loc}[k-1], \mathit{rloc}, \mathit{erloc}, \mathit{lastL}[k], \mathit{erlst}))
\end{array}\right]$$

for every $k \geq 2$.

Abstraction 1: The "*two-halves*". As in the same abstraction of the deterministic case, we obtain an abstraction consisting of the composition of a *left-half* philosopher and a *right-half* philosopher. This abstraction is presented in Fig. 6.

The additional compassion property $\langle at\_\ell_{2..4} \wedge at\_m_{5..7}, at\_\ell_8 \vee at\_m_8 \rangle$ reflects a remote interaction between the two end processes.

Using TLV, we model checked that the network invariant, so derived, is inductive, and, that properly connected to a full philosopher, the system satisfies the liveness property.

Abstraction 2: The "*five-by-four*". As in the deterministic case, the "$k+1$-by-k" abstraction has the potential of being much simpler. Unlike the deterministic case, this cannot be done for $k = 3$ because of the additional (lastL) variables.

in $\quad$ *elloc, erloc* : [0..8]
in-out *llst, erlst* : $[-1..1]$ **where** *llst, erlst* $= -1$
out *lloc, rloc* : [0..8] **where** *lloc, rloc* $= 0$

loop forever do
- ℓ_0 : **non-critical**
- ℓ_1 : **goto** $\{\ell_2,\ \ell_5\}$
- ℓ_2 : **await** $elloc = 0 \vee elloc \in \{0..5\} \wedge llst \neq 0$
- ℓ_3 : **go to** $\{\ell_4,\ \ell_8\}$
- ℓ_4 : **go to** ℓ_1
- ℓ_5 : **skip**
- ℓ_6 : **if** $elloc \in \{0..5\}$ **then go to** ℓ_8
- ℓ_7 : **go to** ℓ_1
- ℓ_8 : **Critical**; $llst := 0$

||

loop forever do
- m_0 : **non-critical**
- m_1 : **goto** $\{m_2,\ m_5\}$
- m_2 : **skip**
- m_3 : **if** $erloc \in \{1, 2, 5..7\}$ **then go to** m_8
- m_4 : **go to** m_1
- m_5 : **await** $erloc = 0 \vee erloc \in \{1, 2, 5..7\} \wedge erlst \neq 1$
- m_6 : **go to** $\{m_7,\ m_8\}$
- m_7 : **go to** m_1
- m_8 : **Critical**; $erlst := 1$

Extra Compassion: $(at\_\ell_{2..4}\ \wedge\ at\_m_{5..7},\quad at\_\ell_8\ \vee\ at\_m_8)$

Fig. 6. A two-halves abstraction for the Courteous Philosophers

There is a reason to believe that it can be accomplished for $k = 4$. We describe here the ideas that lead us to this belief. However, since model-checking this abstraction requires running nine processes, we have so far failed in checking it in TLV, hence we just sketch the main ideas.

Thus, we take $\mathcal{I} = S[4]$, that is, a chain of 4 (unmodified) philosophers, and prove that

$$(p[1]\ \|\ p[2]\ \|\ p[3]\ \|\ p[4]\ \|\ p[5])\ \sqsubseteq (P[6]\ \|\ P[7]\ \|\ P[8]\ \|\ P[9])$$

i.e., that 4 philosophers can faithfully emulate 5 philosophers (and the observable variables.)

This is established by letting $P[6]$ mimic $p[1]$ and $P[9]$ mimic $p[5]$. The middle processes, $P[7]$ and $P[8]$, remain mostly idle. The only scenario in which $P[7]$ has to move is when $p[1]$ gives up its left fork, i.e., the system moves from a $at\_\ell_3[1]$-state into a $at\_\ell_4[1]$-state. Similarly, $P[8]$ has to move only in the symmetric situation with respect to $p[5]$ (moving from ℓ_6 into to ℓ_7). In each of these cases, it is possible to let the relevant middle process get hold of the fork it shares with its external neighbour, so that to justify the neighbour's failure to obtain the fork. It is also possible to do it in a way the will guarantee compassion of the abstract system.

7 Conclusion and Future Research

In this paper we studied the problem of proving P-validity (validity with probability 1) of "simple" LTL specifications over finite state program. We showed how probabilistic fairness can be replaced by compassion (strong fairness), thus reducing the problem of proving P-validity of probabilistic program to that of verifying (strongly) fair programs. The compassion requirements so obtained are generally global, i.e., are expressed relative to global state of the system. In order to model-check such properties, the compassion requirements must be local, i.e., expressed relative to states of single processes. Once one obtains local compassion properties, establishing P-validity of simple properties can be fully automatic.

We demonstrated our ideas by providing a formal proof for the Courteous Philosophers algorithm of Lehman and Rabin, a notoriously difficult to formally verify. (Indeed, this is the first published verification of it.) The protocol is a somewhat involved protocol that is both parameterized and probabilistic. We are happy to report that we succeeded in obtaining the proofs using the Weizmann programmable model checker TLV.

The main drawback of our method is the ad-hoc manner in which we "localized" the compassion properties. We are attempting to develop better methodologies and tools to assist us in this step. We are also studying more examples, e.g., parameterized probabilistic mutual exclusion protocols.

Another issue, closely related to the work here, is the notion of abstraction and tools for its verification. The research reported here helped us identify some extensions of the notion of abstraction (notably, clairvoyance and stuttering) that can considerably improve our tools. We are currently attempting to find more such extensions. Thus, we hope to soon be able to automatically establish the "five-to-four" abstraction reported at the end of the previous section.

References

[AK86] K. R. Apt and D. Kozen. Limits for automatic program verification of finite-state concurrent systems. *Information Processing Letters*, 22(6), 1986.

[APR+01] T. Arons, A. Pnueli, S. Ruah, J. Xu, and L. Zuck. Parameterized verification with automatically computed inductive assertions. In *Proc.* 13^{rd} *Intl. Conference on Computer Aided Verification (CAV'01), volume 2102 of*Lect. Notes in Comp. Sci., *Springer-Verlag*, pages 221–234, 2001.

[BCG86] M.C. Browne, E.M. Clarke, and O. Grumberg. Reasoning about networks with many finite state processes. In *Proc. 5th ACM Symp. Princ. of Dist. Comp.*, pages 240–248, 1986.

[CGJ95] E.M. Clarke, O. Grumberg, and S. Jha. Verifying parametrized networks using abstraction and regular languages. In *6th International Conference on Concurrency Theory (CONCUR'95)*, pages 395–407, 1995.

[CLP84] S. Cohen, D. Lehmann, and A. Pnueli. Symmetric and economical solutions to the mutual exclusion problem in a distributed system. *Theor. Comp. Sci.*, 34:215–225, 1984.

[EK00] E.A. Emerson and V. Kahlon. Reducing model checking of the many to the few. In *17th International Conference on Automated Deduction (CADE-17)*, pages 236–255, 2000.

[EN95] E. A. Emerson and K. S. Namjoshi. Reasoning about rings. In *Proc. 22th ACM Conf. on Principles of Programming Languages, POPL'95*, San Francisco, 1995.

[EN96] E.A. Emerson and K.S. Namjoshi. Automatic verification of parameterized synchronous systems. In *R. Alur and T. Henzinger, editors,* Proc. 8^{th} Intl. Conference on Computer Aided Verification (CAV'96)*, volume 1102 of* Lect. Notes in Comp. Sci.*, Springer-Verlag*, 1996.

[HSP82] S. Hart, M. Sharir, and A. Pnueli. Termination of probabilistic concurrent programs. In *Proc. 9th ACM Symp. Princ. of Prog. Lang.*, pages 1–6, 1982.

[KM95] R.P. Kurshan and K.L. McMillan. A structural induction theorem for processes. *Information and Computation*, 117:1–11, 1995.

[KP00] Y. Kesten and A. Pnueli. Control and data abstractions: The cornerstones of practical formal verification. *Software Tools for Technology Transfer*, 4(2):328–342, 2000.

[L85] Zuck L. Interim report to PhD Committee. Technical report, Weizmann Institute of Sciences, 1985.

[LR81] D. Lehmann and M.O. Rabin. On the advantages of free choice: A symmetric and fully distibuted solution to the dining philosophers problem. In *Proc. 8th ACM Symp. Princ. of Prog. Lang.*, pages 133–138, 1981.

[MAB+94] Z. Manna, A. Anuchitanukul, N. Bjørner, A. Browne, E. Chang, M. Colón, L. De Alfaro, H. Devarajan, H. Sipma, and T.E. Uribe. STeP: The Stanford Temporal Prover. Technical Report STAN-CS-TR-94-1518, Dept. of Comp. Sci., Stanford University, Stanford, California, 1994.

[MP95] Z. Manna and A. Pnueli. *Temporal Verification of Reactive Systems: Safety.* Springer-Verlag, New York, 1995.

[PRZ01] A. Pnueli, S. Ruah, and L. Zuck. Automatic deductive verification with invisible invariants. In *Proc.* 7^{th} *Intl. Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'01)*, volume 2031, pages 82–97, 2001.

[PS96] A. Pnueli and E. Shahar. A platform for combining deductive with algorithmic verification. In *R. Alur and T. Henzinger, editors,* Proc. 8^{th} Intl. Conference on Computer Aided Verification (CAV'96)*, volume 1102 of* Lect. Notes in Comp. Sci.*, Springer-Verlag*, pages 184–195, 1996.

[PZ86a] A. Pnueli and L. Zuck. Probablistic verification by tableaux. In *Proc. First IEEE Symp. Logic in Comp. Sci.*, pages 322–331, 1986.

[PZ86b] A. Pnueli and L. Zuck. Verification of multiprocess probabilistic protocols. *Distributed Computing*, 1:53–72, 1986.

[PZ93] A. Pnueli and L.D. Zuck. Probabilistic verification. *Inf. and Cont.*, 103(1):1–29, 1993.

[Rab82] M.O. Rabin. The choice coordination problem. *Acta Informatica*, 17:121–134, 1982.

[SZ93] A.P. Sistla and L.D. Zuck. Reasoning in a restricted temporal logic. *Inf. and Cont.*, 102(2):167–195, 1993.

[VW86] M.Y. Vardi and P. Wolper. An automata-theoretic approach to automatic program verification. In *Proc. First IEEE Symp. Logic in Comp. Sci.*, pages 332–344, 1986.

[WL89] P. Wolper and V. Lovinfosse. Verifying properties of large sets of processes with network invariants. In J. Sifakis, editor, *Automatic Verification Methods for Finite State Systems*, volume 407 of *Lect. Notes in Comp. Sci.*, pages 68–80. Springer-Verlag, 1989.