Parameterized Verification by Probabilistic Abstraction*

Tamarah Arons[1], Amir Pnueli[1], and Lenore Zuck[2]

[1] Weizmann Institute of Science, Rehovot, Israel,
{amir,tamarah}@wisdom.weizmann.ac.il
[2] New York University, New York, zuck@cs.nyu.edu

Abstract. The paper studies automatic verification of liveness properties with probability 1 over parameterized programs that include probabilistic transitions, and proposes two novel approaches to the problem. The first approach is based on a *Planner* that occasionally determines the outcome of a finite sequence of "random" choices, while the other random choices are performed non-deterministically. Using a Planner, a probabilistic protocol can be treated just like a non-probabilistic one and verified as such. The second approach is based on γ*-fairness*, a notion of fairness that is sound and complete for verifying simple temporal properties (whose only temporal operators are $\diamondsuit$ and $\square$) over finite-state systems. The paper presents a symbolic model checker based on γ-fairness. We then show how the network invariant approach can be adapted to accommodate probabilistic protocols. The utility of the Planner approach is demonstrated on a probabilistic mutual exclusion protocol. The utility of the approach of γ-fairness with network invariants is demonstrated on Lehman and Rabin's Courteous Philosophers algorithm.

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 potentially adversarial (though somewhat fair) scheduler, and the nondeterminism associated with the probabilistic actions, that assumes an even-handed coin-tosser.

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 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

* This research was supported in part by NSF grant CCR-0205571, and the John von Neumann Minerva Center for Verification of Reactive Systems.

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 [PZ93] presents sound and complete methodology for establishing P-validity of general temporal properties over probabilistic systems, and [VW86,PZ93] describe explicit-state model checking procedures 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 amenable 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,KPSZ02].)

One of the promising approaches to the uniform verification of parameterized systems is the method of *network invariants*, first mentioned in [BCG86,SG89], further developed in [WL89] (who also coined the name "network invariant"), and elaborated in [KM95] into a working method. In [KP00,KPSZ02] we extended the approach by using a notion of abstraction that takes into account fairness properties. The approach was developed into a working method and implemented on the Weizmann Institute Temporal Logic Verifier TLV [PS96].

Another promising approach to the uniform verification of parameterized systems is the method of *counter abstraction*: Given a parameterized system with finitely many local states, a concrete state is abstracted by counting, for each possible local state, the minimum between 2 and the number of processes with that local state. Traditionally, counter abstraction is used for proving safety properties of parameterized systems (e.g., [Lub84].) More recently ([BLS01,PXZ02]) it was applied also to the verification of liveness properties; [PXZ02] employs explicit abstraction of fairness requirements.

Since many of the probabilistic protocols that have been proposed and studied (e.g., [LR81,Rab82,CLP84]) are parameterized, a naturally arising question is whether we can combine automatic verification tools of parameterized systems with those of probabilistic ones.

In this paper we propose two novel approaches to the problem. The first is based on *Planners* and the second on the notion of γ*-fairness* introduce in [ZPK02]. When activated, a planner pre-determines the results of a the next k consecutive "random" choices, allowing these next choices to be performed in a completely non-deterministic manner. The approach is sound for finite-state systems: if there is a planner such that a temporal property holds over all computations of the (non-probabilistic) program that activates the planner infinitely often, then the property is P-valid. To deal with parameterized systems, we abstract a version of the system which activates the planner infinitely many times.

The notion of γ*-fairness* is a notion of fairness that is sound and complete for verifying simple temporal properties (whose only temporal operators are $\diamondsuit$ and $\square$) over finite-state systems. We devised a *symbolic* model checking algorithm based on γ-fairness that automatically verifies simple temporal properties of finite-state systems. The algorithm was implemented using TLV. We also extended the network invariant method to apply to γ-fairness, obtaining a method for verifying P-validity over parameterized systems.

Whereas we consider only P-validity, the PRISM probabilistic model checker [KNP02], based on Markov chains and processes, allows the user to verify that a property holds with arbitrary probability, and not just probability 1. However, PRISM does not support the uniform verification of parameterized systems, but rather the verification of individual system configurations. Thus, while PRISM was used to automatically verify the [PZ86b] mutual exclusion protocol for $N = 10$, we (in Section 5) automatically verify it for *every* $N > 1$.

The paper is organized as follows: In Section 2 we describe our formal model, *probabilistic discrete systems* (PDS), which is a fair discrete system augmented with probabilistic requirements. We then define the notion of P-validity over PDSs. We also briefly describe the programming language (SPL augmented with probabilistic goto statements) that we use in our examples and its relation to PDS. In Section 3 we introduce our two new methods for proving P-validity over finite-state systems: The Planner approach, and γ-fairness. We also introduce SYMPMC, the symbolic model checker based on γ-fairness. In Section 4 we describe our model for (fully symmetric) parameterized systems, the method of counter abstraction, and the method of network invariants extended to deal with γ-fairness. Section 5 contains two examples: An automatic P-validity proof of the liveness property of parameterized probabilistic mutual exclusion algorithm [PZ86b] that uses a Planner combined with counter-abstraction, and an automatic proof of P-validity of the individual liveness of the Courteous Philosophers algorithm [LR81] using SYMPMC and Network Invariants. The mutual exclusion example is, to our knowledge, the first formal and automatic verification of this protocol (and protocols similar to it.) The Courteous Philosopher example was proven before in [KPSZ02]. However, there we converted the protocol to a non-probabilistic one with compassion requirements, that had to be devised manually and checked separately, and only then applied the network invariant abstraction. Here the abstraction is from a probabilistic, into a probabilistic, protocol. The advantage of this method is that it does not require the user to compile the list of compassion requirements, nor for the compassion requirements to be checked. Since checking the probabilistic requirements directly is more efficient than checking multiple compassion requirements, the run times are significantly faster (speedups of 50 to 90 percent.) We conclude in Section 6.

2 The Framework

As a computational model for reactive systems we take the model of *fair discrete system* (FDS) [KP00], which is a slight variation on the model of *fair transition system* [MP95], and add *probabilistic requirements* that describe the outcomes of probabilistic selections. We first describe the formal model and the notion of *P-validity*—validity with probability 1. We then briefly describe an a simple programming language that allows for probabilistic selections.

2.1 Probabilistic Discrete Systems

In the systems we are dealing with, all probabilities are bounded from below. In addition, the only properties we are concerned with are temporal formulae which hold with probability 1. For simplicity of presentation, we assume that all probabilistic choices are *binary with equal probabilities*. These restrictions can be easily removed without impairing the results or methods presented in this paper.

A *probabilistic discrete system (*PDS*)* $S : \langle V, \Theta, \rho, \mathcal{P}, \mathcal{J}, \mathcal{C} \rangle$ 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. We assume that Σ is finite.
- Θ: An *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 $s' \in \Sigma$.
- $\mathcal{P}$: A finite set of *probabilistic selections*, each is a triplet (r, t_1, t_2) where r, t_1 and t_2 are assertions. Each such triplet denotes that t_1- and t_2-states are the possible outcomes of a probabilistic transition originating at r-states. We require that for every s and s' such that s' is a ρ-successor of s, there is at most one probabilistic selection $(r, t_1, t_2) \in \mathcal{P}$ and one $i \in \{1, 2\}$ such that s is an r-state and s' is a t_i-state. Thus, given two states, there is at most a single choice out of a single probabilistic selection that can lead from one to the other.
- $\mathcal{J}$: A set of *justice* (*weak fairness*) *requirements*, each given as an assertion. They guarantee that every computation has infinitely many J-states, for every $J \in \mathcal{J}$.
- $\mathcal{C}$: A set of *compassion* (*strong fairness*) *requirements*, each is a pair of assertions. They guarantee that every computation has either finitely many p-states or infinitely many q-states, for every $(p, q) \in \mathcal{C}$.

We require that every state $s \in \Sigma$ has *some* transition enabled on it. This is often ensured by requiring ρ to include the disjunct $V = V'$ which represents the *idle* transition.

Let S be an PDS for which the above components have been identified. We define a *computation tree* of S to be an infinite tree whose nodes are labeled by Σ defined as follows:

- *Initiality:* The root of the tree is labeled by an initial state, i.e., by a Θ-state.
- *Consecution:* Consider a node n labeled by a state s. Then one of the following holds:
  1. n has two children, n_1 and n_2, labeled by s_1 and s_2 respectively, such that for some $(r, t_1, t_2) \in \mathcal{P}$, s is an r-state, and s_1 and s_2 are t_1- and t_2-states respectively.
  2. n has a single child n' labeled by s', such that s' is a ρ-successor of s and for no $(r, t_1, t_2) \in \mathcal{P}$ is it the case that s is an r-state and s' is a t_i-state for some $i \in \{1, 2\}$.

Consider an infinite path $\pi : s_0, s_1, \ldots$ in a computation tree. The path π is called *just* if it contains infinitely many occurrences of J-states for each $J \in \mathcal{J}$. Path π is called *compassionate* if, for each $(p, q) \in \mathcal{C}$, π contains only finitely many occurrences of

p-states or π contains infinitely many occurrences of q-states. The path π is called *fair* if it is both just and compassionate.

A computation tree induces a probability measure over all the infinite paths that can be traced in the tree, the edges leading from a node with two children have each probability 0.5, and the others have probability 1. We say that a computation tree is *admissible* if the measure of fair paths in it is 1. Following [PZ93], we say that a temporal property φ is *P-valid* over a computation tree if the measure of paths in the tree 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 S if it is P-valid over every admissible computation tree of S.

Note that when S is non-probabilistic, that is, when $\mathcal{P}$ is empty, then the notion of P-validity over S coincides with the usual notion of validity over S.

2.2 Probabilistic SPL

All our concrete examples are given in SPL (Simple Programming Language), which is used to represent concurrent programs (e.g., [MP95,MAB$^+$94]). Every SPL program can be compiled into a PDS in a straightforward manner. In particular, every statement in an SPL program contributes a disjunct to the transition relation. For example, the assignment statement "$\ell_0\colon \mathtt{x} := \mathtt{y} + \mathtt{1};\ \ell_1\colon$" can be executed when control is at location ℓ_0. When executed, it assigns the value of $y+1$ to the variable x while control moves from ℓ_0 to ℓ_1. This statement contributes to the transition relation, in the PDS that describes the program, the disjunct $\mathit{at\_}\ell_0 \ \wedge\ \mathit{at'\_}\ell_1 \ \wedge\ x' = y+1 \ \wedge\ \mathit{pres}(V - \{x, \pi\})$ (where for a set $U \subseteq V$, $\mathit{pres}(U) = \bigwedge_{u \in U} u' = u$) and nothing to the probabilistic requirements. The predicates $\mathit{at\_}\ell_0$ and $\mathit{at'\_}\ell_1$ stand, respectively, for the assertions $\pi = 0$ and $\pi' = 1$, where π is the control variable denoting the current location within the process to which the statement belongs (program counter).

In order to represent probabilistic selections, we augment SPL by probabilistic goto statements of the form

$$\ell_0\colon\ \textbf{pr\_goto}\ \{\ell_1, \ell_2\}$$

which adds the disjunct $\mathit{at\_}\ell_0 \ \wedge\ (\mathit{at'\_}\ell_1 \vee \mathit{at'\_}\ell_2) \ \wedge\ \mathit{pres}(V - \{\pi\})$ to the transition relation and the triplet $(\mathit{at\_}\ell_0, \mathit{at\_}\ell_1, \mathit{at\_}\ell_2)$ to the set of probabilistic requirements. Note that, since we allow stuttering (idling), we lose no generality by allowing only probabilistic goto's, as opposed to more general probabilistic assignments.

3 Automatic Verification of Finite-State PDSs

Automatic verification of finite-state PDSs has been studied in numerous works e.g., [VW86,PZ86a,KNP02]. Here we propose two new approaches to automatic verification of P-validity of finite-state PDSs, both amenable to dealing with parameterized (infinite-state) systems.

3.1 Automatic Verification using Planners

A *planner* transforms a probabilistic program Q into a non-probabilistic program Q_T by pre-determining the results of a the next k consecutive "random" choices, allowing these next choices to be performed in a completely non-deterministic manner. The transformation is such that, for every temporal formula φ over (the variables of) Q, if φ is valid over Q_T, then it is P-valid over Q. Thus, the planner transformation converts a PDS into an FDS , and reduces P-validity into validity. The transformation is based on the following consequence of the Borel-Cantelli lemma [Fel68]:

> Let $b_1, \ldots, b_k$ be a sequence of values, $b_i \in \{1, 2\}$, for some fixed k. Let σ be a computation of the program Q which makes infinitely many probabilistic selections. Then, with probability 1, σ contains infinitely many segments containing precisely k probabilistic choices in which these choices follow the pattern $b_1, \ldots, b_k$.

The transformation from Q to Q_T can be described as follows: Each probabilistic statement "ℓ: **pr_goto** $\{\ell_1, \ell_2\}$" in Q is transformed into:

$$\begin{array}{ll} \textbf{if } \textit{consult}_\ell > 0 & \\ \textbf{then} & \left[\begin{array}{l} \textit{consult}_\ell := \textit{consult}_\ell - 1 \\ \textbf{if } \textit{planner}_\ell \textbf{ then goto } \ell_1 \textbf{ else goto } \ell_2 \end{array}\right] \\ \textbf{else goto } \{\ell_1, \ell_2\} & \end{array}$$

Thus, whenever counter $\textit{consult}_\ell$ is positive, the program invokes the boolean-valued function $\textit{planner}_\ell$ which determined whether the program should branch to ℓ_1 or to ℓ_2. Each such "counselled" branch decrements the counter $\textit{consult}_\ell$ by 1. When the counter is 0, the branching is purely non-deterministic. The function $\textit{planner}_\ell$ can refer to all available variables. Its particular form depends on the property φ, and it is up to the user of this methodology to design a planner appropriate for the property at hand. This may require some ingenuity and a thorough understanding of the analyzed program.

Finally, we augment the system with a parallel process, the *activator*, that non-deterministically sets all $\textit{consult}_\ell$ variables to a constant value k. We equip this process with a justice requirement that guarantees that it replenishes the counters (activates the planners) infinitely many times. A proof for the soundness of the method is in Appendix A.

Example 1. Consider Program *up-down* in Fig. 1 in which two processes increment and decrement $y \in [0..4]$.

To establish the P-validity of $\varphi : \square\diamondsuit(y = 0)$, we can take $k = 4$ and define both $\textit{planner}_{\ell_0}$ and $\textit{planner}_{m_0}$ to always yield 0, thus consistently choosing the second (decrementing) mode.

3.2 SYMPMC: A Symbolic Probabilistic Model Checker

While the planner strategy is applicable for many systems, there are cases of parameterized systems (defined in Section 4) for which it cannot be applied. (See Section 5.2

local $y \in [0..4]$ **init 2**

$$\left[\begin{array}{l}\textbf{loop forever do}\\ \quad\left[\begin{array}{l}\ell_0 : \textbf{pr-goto}\ \{\ell_1, \ell_2\}\\ \ell_1 : y := \min(y+1, 4);\\ \qquad \text{goto } \ell_0\\ \ell_2 : y := \max(y-1, 0);\\ \qquad \text{goto } \ell_0\end{array}\right]\end{array}\right] \quad || \quad \left[\begin{array}{l}\textbf{loop forever do}\\ \quad\left[\begin{array}{l}m_0 : \textbf{pr-goto}\ \{m_1, m_2\}\\ m_1 : y := \min(y+1, 4);\\ \qquad \text{goto } m_0\\ m_2 : y := \max(y-1, 0);\\ \qquad \text{goto } m_0\end{array}\right]\\ \qquad -P_2-\end{array}\right]$$

$$-P_1-$$

Fig. 1. Program *up-down*

for an example.) As an alternative method we describe SYMPMC, a symbolic model checker that verifies P-validity of *simple* temporal properties—properties whose only temporal operators are $\diamondsuit$ and $\square$—over finite state PDSs.

The motivation leading to SYMPMC has been the idea that, since all the probabilities of a finite PDS are bounded from below, the definition of P-validity that directly deals with measure spaces can be replaced with some simpler notions of fairness. This was first done in [Pnu83], where *extreme-fairness* was introduced, a notion of fairness that is sound and incomplete for proving P-validity. The work in [PZ93] introduced α*-fairness*, which was shown to be sound and complete. The work there also introduced an explicit-state model checker for P-validity that is based on α-fairness. The main drawback of α-fairness is that it requires fairness with respect to *every past* formula. From the model checking procedure of [PZ93] it follows that the only past formulae really needed are those that appear in the normal-form of the property to be verified. However, obtaining the normal-form is non-elementary in the size of the property.

In [ZPK02] we observed that a consequence of the model checking procedure of [PZ93] is that replacing α-fairness by the simpler γ-fairness results in a notion of fairness that is sound and complete for proving *simple* properties over finite-state programs. The definition of γ fairness is as follows:

Assume a PDS $S : \langle V, \Theta, \rho, \mathcal{P}, \mathcal{J}, \mathcal{C}\rangle$ and a path $\pi = s_0, s_1, \ldots$ in a computation tree of S. Let (r, t_1, t_2) be a probabilistic selection. We say that mode (r, t_j), $j \in \{1, 2\}$ is *taken* at position $i \geq 0$ of π if $s_i \models r$ and $s_{i+1} \models t_j$. We say that the selection (r, t_1, t_2) is taken at position i if either (r, t_1) or (r, t_2) is taken at i. Let s be a state of the system. We call i an s-position if $s_i = s$. We say that π is γ*-fair*, if for each state s (and there are only finitely many distinct states) and each probabilistic selection (r, t_1, t_2), either there are only finitely s-positions from which (r, t_1, t_2) is taken, or for every $i = 1, 2$, there are infinitely many s-positions from which (r, t_i) is taken. The following corollary states that the replacement of probability by γ-fairness is sound and complete with respect to P-validity of simple formulae. It is an immediate consequence of [PZ86a].

Corollary 1. *For every finite-state* PDS S *and simple formula* φ, φ *is P-valid over* S *iff* $\sigma \models \varphi$ *for every* γ*-fair computation*

Based on Corollary 1, the explicit-state model checking procedure of [PZ93], and the non-probabilistic feasibility algorithm of [KPR98], we introduce SYMPMC, a symbolic model checker for verifying P-validity of simple properties over finite-state PDSs.

The core of SYMPMC is an algorithm for simple response formulae (formulae of the form $\square(a \rightarrow \diamondsuit b)$, where a and b are assertions) over a finite state PDS. This algorithm is presented in Fig. 2. It checks the P-validity of $\varphi : \square(a \rightarrow \diamondsuit b)$ for assertions a and b. The algorithm returns $\emptyset$ iff φ is γ-valid over S, i.e., if φ holds over all γ-fair computations of S. SYMPMC had been implemented using TLV [PS96].

```
ALGORITHM RESPONSE(S)
var:
  R:      relation  init |ρ| ∩ (‖¬b‖ × ‖¬b‖)
  new:    predicate init (‖Θ‖ ∘ ‖ρ‖*) ∩ ‖¬b‖
  oldR:   relation  init ∅
  old:    predicate init ∅

while (new ≠ old ∨ R ≠ oldR) do
   old := new
   oldR := R
   new := new ∩ (R ∘ new)
   R := R ∩ (new × new)
   foreach J ∈ 𝒥 do
         new :=  new ∩ R*∘ ‖J‖
         R:= R ∩ (new × new)
   foreach (p, q) ∈ 𝒞 do
         new := (new - ‖p‖) ∪
              (new ∩ R*∘ ‖q‖)
         R := R ∩ (new × new)
   foreach R ∈ 𝒫 do
         treat-P-requirement(R)
endwhile
return
   (‖Θ‖ ∘ ‖ρ‖*) ∩ (‖a‖ − ‖b‖) ∩ (R*∘ new)
```

where for a probabilistic requirement $R = (r, t_1, t_2)$,

```
treat-P-req(R):
  var:
    qpred: predicate init Σ
    someq: predicate init ∅
    pbad:  predicate

  for j =1 to 2 do
    qpred := qpred ∩ (R ∘ ‖t_j‖)
    someq := someq ∪ ‖t_j‖
    pbad := ‖r‖ - qpred
    R :=  R ∩
          [(pbad × (Σ - someq))
            ∪ ((Σ - pbad)×Σ)]
```

Fig. 2. Algorithm RESPONSE **for model-checking the P-validity of** $\varphi : \square(a \rightarrow \diamondsuit b)$

The main difference between the algorithm in Fig. 2 and its counterpart in [KPR98] is the treatment of probabilistic requirements (the third "`foreach`" in the `while` loop). For each probabilistic requirement $R = (r, t_1, t_2)$, the procedure `treat-P-req(R)` removes from the graph all states that are not γ-fair with respect to R.

In [APZ02] we prove:

Theorem 1. *For an input* PDS *S, Algorithm* RESPONSE *terminates. For assertions a and b,* RESPONSE *returns V such that $\varphi = \square(a \rightarrow \diamondsuit b)$ is P-valid in S iff $V = \emptyset$.*

By setting a and b to true, Algorithm RESPONSE can be used to check for γ*-feasibility* (whether the system has at least one γ-fair computation). It can also be used to check the validity of simple formulae by composing the system with temporal testers [KPR98]. Thus, SYMPMC can be used for symbolic model checking of whether a simple temporal formula is P-valid over a finite state program.

4 Probabilistic Parameterized Systems

In this section we turn to probabilistic parameterized systems and their automatic verification. We first define the systems that are the scope of this paper and briefly discuss their automatic verification using Planners and SYMPMC.

4.1 Parameterized Systems

We focus on probabilistic parameterized systems that consist of multiple copies of N identical finite-state SPL processes. For each value of $N > 0$, $S(N)$, the PDS that describes the system, is an instantiation of an PDS. Thus, such a system represents an infinite *family* of systems, one for each value of N.

We are interested in properties that hold for every process in the system. Thus, we are interested in liveness properties of the type φ, where φ is a temporal formula referring only to variables local to a single process. The problem of parameterized verification is to show that φ is P-valid over *every* $S(N)$.

4.2 Counter Abstraction

In [PXZ02] we proposed the method of *counter abstraction* for the verification of liveness properties of parameterized systems. A brief overview of the approach for non-probabilistic systems is given here. For details see [PXZ02].

For simplicity of presentation, we assume that the system $S(N)$ has a set X of global shared variables whose size is independent of N, and the only variable local to each process $P[i]$ is the program counter $\pi[i]$. Each global state s of the system $S(N)$ is then an $(N + |X|)$-tuple, describing the location of each process and the values of each $x \in X$. Assume that the program counters range over the set $\{0 \dots L-1\}$.

We define the *counter abstraction* of state s by an $(L + |X|)$-tuple, such that each one of the first $|L|$ elements is the *counter* of the corresponding location, where for a location ℓ, the counter of ℓ, denoted by κ_ℓ, is defined by:

$$\kappa_\ell = \begin{cases} 0 & - \quad \text{there are no processes in location } \ell \\ 1 & - \quad \text{there is exactly one process in location } \ell \\ 2 & - \quad \text{there are two or more processes in location } \ell \end{cases}$$

Properties are similarly abstracted. Thus, for example, the property $\exists i : at\_\ell_\ell[i]$ is abstracted to $\kappa_\ell > 0$. Denote by $\alpha(\varphi)$ the counter-abstraction of the property φ.

As explained in [PXZ02], in order to be able to prove liveness properties it is necessary to carefully abstract the fairness properties. Once this is done, we obtain the abstracted system $\alpha(S)$, and can show that for every liveness property φ, the validity of $\alpha(\varphi)$ over $\alpha(S)$ implies the validity of φ over $S(N)$ for every $N > 1$.

Suppose we have a probabilistic parameterized system $S(N)$ and a temporal property φ we wish to show is P-valid. We can apply a Planner transformation, obtaining a non-probabilistic parameterized system, to which we can then apply counter-abstraction, which will reduce it to an unparameterized finite-state system. If model checking reveals that $\alpha(\varphi)$ is valid for this system, we can safely conclude that φ is P-valid over $S(N)$.

It is important to note that counter-abstraction can be obtained in a fully automatic way. Obviously, model checking techniques can easily check whether an abstracted system satisfies abstract properties. The only step in the process that requires user intervention (and ingenuity) is in crafting the functions used by the Planner.

In Section 5.1 we demonstrate the power of the approach by verifying the liveness of a probabilistic N-process mutual exclusion algorithm. Previous attempts to verify the same protocol were either manual [PZ86a] or automatic for $N \leq 10$ [KNP00].

4.3 Network Invariants

The method of network invariants was first mentioned in [BCG86,SG89], further developed in [WL89] (who also coined the name "network invariant"), and elaborated in [KM95] into a working method. The formulation here follows [KP00] and [KPSZ02], which take into account the fairness properties of the compared systems and support proofs of liveness properties.

In order to apply the method to PDSs, it is necessary to refine the model, so that it allows for "environment" actions. Roughly speaking, the set of variables includes a special subset consisting of the variables *owned* by the system. The system then takes steps, alternating between environment steps that can change all but the owned variables.

It is also necessary to define the *observable behavior* of a system. To that end, the set of variables is assumed to includes a set of *observables*—variables that are externally observable. The observables are denoted by $\mathcal{O}$.

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

Suppose we are given two comparable systems, a *concrete* $\mathcal{D}_C$ and an *abstract* $\mathcal{D}_A$, and wish to establish that $\mathcal{D}_C \sqsubseteq_\gamma \mathcal{D}_A$. 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 $\mathcal{O}_A$.

In Fig. 3, we present a rule for proving that S_A γ-abstracts S_C. The rule 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.

The Abstraction Rule resembles the abstraction rule of [KPSZ02], with the addition of Premise A6. This premise must, in general, be verified for every assignment U to the abstract variables V_A. The following condition suffices to guarantee premise A6 and is met in many abstractions:

> For every abstract requirement $(r, t_1, t_2) \in \mathcal{P}_A$, there exists a concrete requirement $(r^C, t_1^C, t_2^C) \in \mathcal{P}_C$, where $r[\alpha] = r^C$, $t_1[\alpha] = t_1^C$ and $t_2[\alpha] = t_2^C$.

$$
\begin{array}{ll}
\mathbf{A1}.\ \Theta_C \to \exists V_A : \Theta_A[\alpha] & \\
\mathbf{A2}.\ \mathcal{D}_C \models \Box(\rho_C \quad \to \quad \exists V'_A : \rho_A[\alpha][\alpha']) & \\
\mathbf{A3}.\ \mathcal{D}_C \models \Box(\alpha \to \mathcal{O}_C = \mathcal{O}_A) & \\
\mathbf{A4}.\ \mathcal{D}_C \models \Box \Diamond J[\alpha], & \text{for every } J \in \mathcal{J}_A \\
\mathbf{A5}.\ \mathcal{D}_C \models \Box \Diamond p[\alpha] \to \Box \Diamond q[\alpha], & \text{for every } (p,q) \in \mathcal{C}_A \\
\mathbf{A6}.\ \mathcal{D}_C \models \Box \Diamond((V_A = U)[\alpha] \wedge r[\alpha] \wedge \bigcirc \bigvee_{i=1}^{2} t_i[\alpha]) \to & \\
\qquad \bigwedge_{i=1}^{2} \Box \Diamond((V_A = U)[\alpha] \wedge r[\alpha] \wedge \bigcirc t_i[\alpha]), & \\
\qquad \text{for every } (r, t_1, t_1) \in \mathcal{P}_A \text{ and every assignment } U \text{ over } V_A & \\
\hline
\qquad S_C \sqsubseteq_\gamma S_A &
\end{array}
$$

Fig. 3. Abstraction Rule

Given a parameterized system $S(N)$, the network invariant method calls for devising a *network invariant* $\mathcal{I}$ — a finite state PDS, intended to provide an abstraction for the (open) parallel composition of any k processes of the parameterized system. The method then calls for confirming that $\mathcal{I}$ is indeed an abstraction, and model checking that when composed with a single process it satisfies a property of that process. The first step, that of designing $\mathcal{I}$, calls for some ingenuity of the verifier. As we showed in [KPSZ02], the task can be often quite simple. The third step can be achieved using SYMPMC. The second step, confirming that $\mathcal{I}$ is indeed a good abstraction, calls for establishing the two γ-abstractions $(P[1]\| \cdots \|P[m]) \sqsubseteq_\gamma \mathcal{I}$ and $(\mathcal{I}\|P[i]) \sqsubseteq_\gamma \mathcal{I}$, where m is a small constant (usually in the range $[1..3]$) independent of N, and $P[i]$ is a generic copy of the system's process.

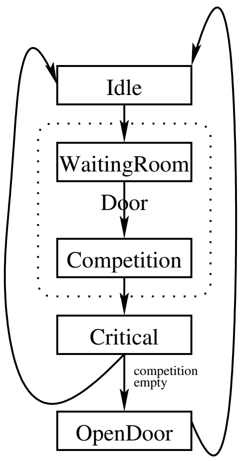
5 Examples

In this section we present two examples, one for each of the methodologies we described in the previous section. To demonstrate the power of the "Planner and Counter abstraction" approach, we take the probabilistic mutual exclusion protocol of [PZ86b]. To demonstrate the power of the "SYMPMC and network invariant" approach, we take the Lehman and Rabin's Courteous Philosopher protocol [LR81].

5.1 Verifying Probabilistic Mutual Exclusion using a Planner

A flow diagram of the protocol, as well as its SPL code, are in Fig. 4. The usual "trying section" consists of two parts: A "waiting room" in which a process waits for a "door" to open in order to be admitted to the "competition", and the competition. Once the door closes, no new process can enter the competition. Processes in the competition flip coins: Losers, those who flip *tails*, wait until there are no winners. A process that flips *heads*, and finds out that it is the only one to have done so (and there are no processes in the critical or exit region), enters the critical section. Otherwise, it waits until all winners join it, and they all proceed to flip coins again. Once a process leaves the critical region, it examines if there are processes in the competition. If there are, it just goes back to its idle state. Otherwise, it opens the door and waits until all processes in the waiting room enter the competition before it goes back to the idle state.

In the original protocol each process has a variable y that can take on eight values, according to the location(s) of the process. We omit it here, and instead present a version that includes locations only, and is amenable to counter-abstraction. Each process i can perform, in a single atomic step, a test consisting of a boolean combination of formulae of the form $\exists j \neq i : \pi[j] \in L$ for $L \subseteq [0..15]$. We denote such a test by *some* $\in L$, and its negation by *none* $\in L$. The only probabilistic transition is at location ℓ_4.



in n : **integer where** $n \geq 2$

$$\mathop{\Big|\Big|}_{i=1}^{n} P[i] ::$$

loop forever do

- ℓ_0 : **non-critical**
- ℓ_1 : **skip**
- ℓ_2 : **await** *none* $\in \{4..13\}$ $\vee$ *some* $\in \{14, 15\}$
- ℓ_3 : **skip**
- ℓ_4 : **pr_goto** $\{\ell_5, \ell_8\}$
- ℓ_5 : **if** *none* $\in \{5, 6, 10..15\}$ **then goto** ℓ_{10}
- ℓ_6 : **skip**
- ℓ_7 : **await** *none* $\in \{5, 6, 10..15\}$ **and then goto** ℓ_4
- ℓ_8 : **await** *none* $\in \{4..7, 10..15\}$
- ℓ_9 : **goto** ℓ_4
- ℓ_{10} : **Critical**
- ℓ_{11} : **if** *none* $\in \{4..9\}$ **goto** ℓ_{13}
- ℓ_{12} : **goto** ℓ_0
- ℓ_{13} : **skip**
- ℓ_{14} : **await** *none* $\in \{2, 3\}$
- ℓ_{15} : **goto** ℓ_0

Fig. 4. A Probabilistic Mutual Exclusion Protocol

The mutual exclusion property of the protocol, stating that it is never the case that two or more processes are in ℓ_{10} at the same time, is easy to model check, for example using the methodology of [APR$^+$01] or counter-abstraction. The liveness property of the protocol is $\forall i : \pi[i] = 1 \rightarrow \diamondsuit(\pi[i] = 10)$, and we wish to show its P-validity for every $N \geq 2$.

We take $k = 1$ and define a planner *planner*$_4$ which determines the result of the next probabilistic choice at ℓ_4 whenever activated. This planner is defined by

planner$_4$: *none* $\in \{5, 6, 10..15\}$

This planner directs the next branch to ℓ_5 if there is no process in any of the locations $H = \{5, 6, 10..15\}$, and to ℓ_6 otherwise.

The intuition behind this planner is that a process can enter the critical section, from location ℓ_5, only if there are no other processes in H-locations. When there are processes in H-locations, we want to reduce the number of processes that are likely to join them, which $\mathit{planner}_4$ accomplishes by returning "0" and thus forcing processes from ℓ_4 to enter ℓ_8. When there are no processes in H-locations, we want the first process that can to enter ℓ_5, so that it can enter the critical section. Thus, $\mathit{planner}_4$ returns "1" in such cases. This planner design can obviously be counter abstracted, hence, we succeeded to use TLV to establish the livelock-freedom property of the protocol given by: $\square(\exists i : \pi[i] = 1 \rightarrow \diamondsuit(\exists i : \pi[i] = 10))$.

Once livelock freedom has been established, it is the structure of the protocol that guarantees individual liveness, by restricting the number of times a process in the competition can overtake another — once a process i is trying to access the critical section (enters $\ell_{2..9}$), every other process can enter the critical section at most twice before i does, which trivially implies the individual accessibility or the protocol [3].

We also established the individual liveness property of the protocol directly using TLV and "counter abstraction save one" ([PXZ02]) using the same planner. See *http://www.cs.nyu.edu/zuck/pubs/pme* for TLV code.

5.2 Verifying the Courteous Philosophers using SYMPMC

The success of the planner strategy in parameterized systems depends on having a *single* strategy for random draws that will allow *every* process to achieve its liveness property. The [LR81] Courteous Philosophers Algorithm is an example where we cannot use a planner: any planner strategy that allows one philosopher to eat may preclude its neighbours from eating. Hence, to automatically verify The [LR81] Courteous Philosophers Algorithm, we use the network invariant approach.

The network invariant we obtained is essentially the one derived in [KPSZ02] and we omit it here for space reasons. There is, however, a crucial difference: In [KPSZ02] we replaced the probabilistic requirements by compassion requirements. Here, with the aid of the revised Abstraction Rule and SYMPMC, we could work directly with γ-fairness and did not need to (manually) devise adequate compassion requirements replacing the probabilistic choices. Thus, the resulting network invariant is significantly simpler and execution time is much shorter.

6 Conclusion and Future Work

The paper deals with the problem of automatic proof of P-validity of liveness properties over parameterized systems. We started with a discussion of the non-parameterized case, and described two new approaches to the problem: Planners that convert a probabilistic system into a non-probabilistic one and allow one to treat P-validity as regular validity, and model checking over γ-fair computations, which is sound and complete for simple temporal properties. We then outlined the two approaches of automatic verification of liveness properties of parameterized systems, counter abstraction and network invariants, and showed how the network invariant method can be combined with

[3] Bounded overtaking property is a safety property and thus it can be established by ignoring the probabilistic transitions of the protocol.

SYMPMC. We demonstrated our techniques by providing automatic proofs for two non-trivial protocols. The first by Planner & counter-abstraction, the second by SYMPMC & (extended) network invariants.

Strictly speaking, neither method combination we used in our examples is *fully automatic*, they both require user input. On one hand the design of a Planner or a Network Invariant may require user ingenuity; on the other hand, most systems are verified by their own designers, who have a pretty good intuition about the appropriate Planner/network invariant.

We are currently working on extending counter-abstraction with γ-fairness. If successful, this will provide a fully automatic proofs of P-validity of parameterized system for the cases that the method of counter-abstraction is applicable.

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^{th} *Intl. Conference on Computer Aided Verification (CAV'01), volume 2102 of*Lect. Notes in Comp. Sci.*, Springer-Verlag*, pages 221–234, 2001.

[APZ02] T. Arons, A. Pnueli, and L. Zuck. Verification by probabilistic abstraction. Weizmann Institute of Science Technical Report, 2002.

[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.

[BLS01] K. Baukus, Y. Lakhnesche, and K. Stahl. Verification of parameterized protocols. Journal of Universal Computer Science, 7(2):141–158, 2001.

[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.

[Fel68] W. Feller. *An Introduction to Probability Theory and its Applicaitons*, volume 1. John Wiley & Sons, 3 edition, 1968.

[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.

[KNP00] M. Kwiatkowska, G. Norman, and D. Parker. Verifying randomized distributed algorithms with prism. In *Proc. of the Workshop on Advances in Verification (WAVe) 2000*. 2000.

[KNP02] M. Kwiatkowska, G. Norman, and D. Parker. Prism: Probabilistic symbolic model checker. In TOOLS 2002, volume 2324 of LNCS, 2002.

[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.

[KPR98] Y. Kesten, A. Pnueli, and L. Raviv. Algorithmic verification of linear temporal logic specifications. In K.G. Larsen, S. Skyum, and G. Winskel, editors, *Proc. 25th Int. Colloq. Aut. Lang. Prog.*, volume 1443 of *LNCS*, pages 1–16. Springer-Verlag, 1998.

[KPSZ02] Y. Kesten, A. Pnueli, E. Shahar, and L. Zuck. Network invariants in action. In *Proceedings of Concur'02*, volume 2421 of *LNCS*. Springer-Verlag, 2002.

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

[Lub84] B.D. Lubachevsky. An approach to automating the verification of compact parallel coordination programs. Acta Infromatica, 21, 1984.

[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.

[Pnu83] A. Pnueli. On the extremely fair treatment of probabilistic algorithms. In *Proc. 15th ACM Symp. Theory of Comp.*, pages 278–290, 1983.

[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.

[PXZ02] A. Pnueli, J. Xu, and L. Zuck. The $(0, 1, \infty)$ counter abstraction. In *Proc.* 14^{th} *Intl. Conference on Computer Aided Verification (CAV'02), volume 2404 of*Lect. Notes in Comp. Sci.*, Springer-Verlag*, 2002. *http://www.cs.nyu.edu/˜zuck/pubs/cav02.ps*.

[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. *Information and Computation*, 103(1):1–29, 1993.

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

[SG89] Z. Shtadler and O. Grumberg. Network grammars, communication behaviors and automatic verification. In J. Sifakis, editor, *Automatic Verification Methods for Finite State Systems*, volume 407 of *LNCS*, pages 151–165. Springer-Verlag, 1989.

[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 *LNCS*, pages 68–80. Springer-Verlag, 1989.

[ZPK02] L. Zuck, A. Pnueli, and Y. Kesten. Automatic verification of probabilistic free choice. In *Proc. of the* 3^{rd} *workshop on Verification, Model Checking, and Abstract Interpretation*, volume 2294 of *LNCS*, 2002.

A Soundness of the Planner Approach

We prove the soundness of the planner method by reducing its soundness to that of α-fairness, which was established in [PZ86a].

Let $Q : \langle V, \Theta, \rho, \mathcal{P}, \mathcal{J}, \mathcal{C} \rangle$ be a PDS. We make two simplifying assumptions: (1) $\mathcal{P}$ contains exactly one requirement, $R = (r, t_1, t_2)$, and (2) there is only one location from which R is taken. The proof below is easy to generalize once the two assumptions are removed, at the cost of additional indices.

Let $\sigma = s_0, \ldots$ be a fair computation of Q and let ψ be a past temporal formula, that is, a formula whose only temporal operators are past operators. Computation σ is α-fair with respect to ψ if either R is taken only finitely many times from ψ-prefixes in σ, or each mode of R is taken infinitely many times from ψ-prefixes in σ. For a set $\mathcal{X}$ of temporal formula, σ is $\mathcal{X}$-fair if it is α-fair with respect to every $\psi \in \mathcal{X}$.

An immediate corollary of the analysis presented in [PZ86a] is:

Corollary 2. *. Let $\mathcal{X}$ be a set of past formulae and φ be a temporal formula. Then:*

$$\sigma \models \varphi \text{ for every } \mathcal{X}\text{-fair } \sigma \quad \Longrightarrow \quad \varphi \text{ is P-valid over } Q.$$

Consider now the PDS Q_T obtained from Q by a transforming it with a Planner *planner*, which is set to make k consecutive "planned" choices. We proceed to construct a set of past formulae $\mathcal{X}$ such that every $\mathcal{X}$-fair computation of Q is a V-projection of some computation of Q_T. Thus, for every temporal formula φ over V, if all Q_T computations satisfy φ, then all $\mathcal{X}$-fair computations of Q also satisfy φ. It then follows from Corollary 2 that φ is P-valid. It thus remains to construct $\mathcal{X}$ and show that every $\mathcal{X}$-fair computation of Q has a computation of Q_T with the same V-projection.

For $i = 1, 2$, let p_i be the state assertion characterizing the r-states for which *planner* recommends choosing i. Define *chose* : $(t_1 \vee t_2) \wedge \ominus r$. Formula *chose* characterizes all the states immediately following a probabilistic choice. Next, define $gc : \bigvee_{i=1,2}(t_i \wedge \ominus(r \wedge p_i))$. Formula gc (standing for *good-choice*) characterizes all the states following a probabilistic choice which is compatible with the choice recommended by the planner at this state. Finally, define

$$\begin{array}{lll} \psi_0 & : \mathrm{T} & \\ \psi_{i+1} & : (\neg \mathit{chose}) \, \mathcal{S} \, (gc \ \wedge \ \ominus \psi_i) & \qquad \text{for } i = 0, \ldots, k{-}1 \end{array}$$

Obviously, ψ_i characterizes a point in the computation such that the last i probabilistic choices are compatible with the recommendations of the planner. As the set of past formulae, we take $\mathcal{X} : \{\psi_0, \ldots, \psi_k\}$.

It remains to show that every $\mathcal{X}$-fair computation of Q has a computation of Q_T with the same V-projection. Let σ be a $\mathcal{X}$-fair computation of Q, and assume that σ has infinitely many R-transitions. By induction on $i = 1, \ldots, k$, (which is similar to the proof of the Borel-Cantelli Lemma) it follows that infinitely many times, σ makes k consecutive choices which are compatible with the recommendation of the planner. It is easy now to construct a computation of Q_T in which the planner is activated whenever σ is to start a sequences of good choices, and is activated nowhere else.

The soundness of the Planner strategy follows.