Quantitative Static Analysis of Communication Protocols using Abstract Markov Chains

Abdelraouf Ouadjaout, Antoine Miné

journal manuscript No.
(will be inserted by the editor)

Quantitative Static Analysis of Communication Protocols using Abstract Markov Chains

Abdelraouf Ouadjaout · Antoine Miné

Received: date / Accepted: date

Abstract In this paper we present a static analysis of probabilistic programs to quantify their performance properties by taking into account both the stochastic aspects of the language and those related to the execution environment. More particularly, we are interested in the analysis of communication protocols in lossy networks and we aim at inferring statically parametric bounds of some important metrics such as the expectation of the throughput or the energy consumption. Our analysis is formalized within the theory of abstract interpretation and soundly takes all possible executions into account. We model the concrete executions as a set of Markov chains and we introduce a novel notion of *abstract Markov chains* that provides a finite and symbolic representation to over-approximate the (possibly unbounded) set of concrete behaviors. We show that our proposed formalism is expressive enough to handle both probabilistic and pure non-deterministic choices within the same semantics. Our analysis operates in two steps. The first step is a classic abstract interpretation of the source code, using stock numerical abstract domains and a specific automata domain, in order to extract the abstract Markov chain of the program. The second step extracts from this chain particular invariants about the stationary distribution and computes its symbolic bounds using a parametric Fourier-Motzkin elimination algorithm. We present a prototype implementation of the analysis and we discuss some preliminary experiments on a number of communication protocols. We compare our prototype to the state-of-the-art probabilistic model checker Prism and we highlight the advantages and shortcomings of both approaches.

This work is partially supported by the European Research Council under Consolidator Grant Agreement 681393 – MOPSA.

Abdelraouf Ouadjaout · Antoine Miné
Sorbonne Université, CNRS, Laboratoire d'Informatique de Paris 6, LIP6, F-75005 Paris, France

Antoine Miné
Institut Universitaire de France
E-mail: abdelraouf.ouadjaout@lip6.fr
E-mail: antoine.mine@lip6.fr

1 Introduction

The analysis of probabilistic programs represents a challenging problem. The difficulty comes from the fact that execution traces are characterized by probability distributions that are affected by the behavior of the program, resulting in very complex forms of stochastic processes. In such particular context, programmers are interested in quantitative properties not supported by conventional semantics analysis, such as the inference of expected values of performance metrics or the probability of reaching bug states.

In this work, we propose a novel static analysis for extracting symbolic quantitative information from probabilistic programs. More particularly, we focus on the analysis of communication protocols and we aim at assessing their performance formally. The proposed approach is based on the theory of abstract interpretation [9] that provides a rigorous mathematical framework for developing sound-by-construction static analyses. In the following, we describe informally the main contributions of our work and we illustrate our motivations through some practical examples.

Stationary distributions. Generally, the quantification of performance metrics for such systems is based on computing the *stationary distribution* of the associated random process [13]. It gives the proportion of time spent in every reachable state of the system by considering all possible executions. This information is fundamental to compute the expected value of most common performance metrics. For instance, the throughput represents the average number of transmitted packets per time unit. By identifying the program locations where packets are transmitted and by computing the value of the stationary distribution at these locations, we obtain the proportion of packets sent in one time unit. Many other metrics are based on this distribution, such as the duty cycle (proportion of time where the transceiver is activated) or the goodput (the proportion of successfully transmitted data).

To our knowledge, no existing approach can obtain such information (*i*) *automatically* by analyzing the source code, (*ii*) *soundly* by considering all executions in possibly infinite systems and (*iii*) *symbolically* by expressing the distribution in terms of the protocol parameters. Indeed, most proposed solutions focus on computing probabilities of program assertions [44, 7] or expectation invariants [8, 3]. Only Prism [30], thanks to its extension Param [25], can compute stationary distributions of parametric Markov chains, but it is limited to finite state systems with parametric transition probabilities, whereas we also support systems where the number of states is a (possibly unbounded) parameter.

Example 1 To illustrate this problem, consider the simple wireless protocol shown in Fig. 1a representing a typical backoff-based transmission mechanism used in embedded sensing applications. Assume a star network topology in which a central node collects the readings of a set of surrounding sensor nodes that periodically send their measurements via wireless transmissions. To do that, each sensor node repeatedly activates its sensing device and acquires some readings by calling the `sense` function. To avoid collisions when sending the data, a random backoff is used by sampling a discrete uniform distribution from the range $[1, B]$, where B is an integer parameter of the protocol. The node remains in sleep mode during

```
1  pkt = 0;
2  ack = 0;
3  while(1) {
4    data = sense();
5    //Uniform backoff
6    t = uniform(1, B);
7    wait t;
8    //Transmission with ack
9    if (unicast(data))
10     ack++;
11   pkt++;
12   //Save energy
13   wait S;
14 }
```

(a)

```
15 //Unicast model
16 #define TX_DELAY 1
17 #define RX_DELAY 1
18
19 bool unicast(int data) {
20   //Emulate data transmission
21   wait TX_DELAY;
22   //Emulate ack reception
23   rx = bernoulli();
24   wait RX_DELAY;
25   return rx;
26 }
```

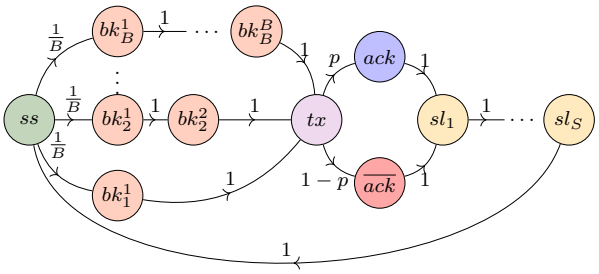
(b)

Fig. 1: (a) Example of a backoff-based transmission protocol. (b) Hardware model of the `unicast` built-in function.

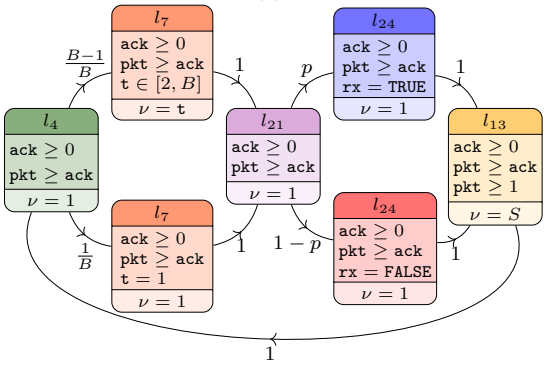
this random period, and after it wakes up, data is transmitted using the `unicast` function. Such functions are generally implemented in hardware by the wireless transceiver, so we give in Fig. 1b its model. Transmission/reception operations are emulated with simple waiting periods; the constant `TX_DELAY` models the transmission delay and the constant `RX_DELAY` models the reception delay. Packet losses are modeled using a Bernoulli distribution, meaning that a packet is transmitted and acknowledged with some parameter probability p, or lost with probability $1-p$. Finally, in order to save energy, the sensor node remains inactive for a duration determined by a parameter S, and then iterates again the same process indefinitely.

A critical task for designers of such systems is to fine-tune the protocol's parameters B and S in order to achieve optimal performance w.r.t. the requirements of the application. Consider for instance that we are interested in the goodput Γ of a sensor; that is, the average number of data packets that are successfully received in one time unit. To study the variation of Γ, system designers generally derive *manually* a mathematical stochastic model of the protocol. In this case, *discrete time Markov chains* are a powerful model embedding many interesting properties that help quantify the performances of our system [43, 23].

We give in Fig. 2a the chain associated with the protocol. Each state of the chain corresponds to a duration of one time unit (e.g. one millisecond). The goodput Γ of the protocol is, therefore, the proportion of time spent in state *ack*, which can be obtained by computing the stationary distribution $\boldsymbol{\pi}$ of the chain. This is done by finding the eigenvector $\boldsymbol{\pi} \stackrel{\text{def}}{=} \langle \boldsymbol{\pi}_{ss}, \boldsymbol{\pi}_{bk_1^1}, \ldots, \boldsymbol{\pi}_{tx}, \boldsymbol{\pi}_{ack}, \boldsymbol{\pi}_{\overline{ack}}, \boldsymbol{\pi}_{sl_1}, \ldots \rangle$ associated with the eigenvalue 1 of the following stochastic matrix:



(a)



(b)

Fig. 2: (a) Discrete time Markov chain of the protocol. ss: sensing state, $\{\, bk_i^j \mid i \in [1, B] \wedge j \in [1, i] \,\}$: backoff states, tx: transmission state, ack: acknowledgment state, $\overline{ack}$: loss state, $\{\, sl_i \mid i \in [1, S] \,\}$: sleep states. (b) Inferred abstract Markov chain.

$$\mathbf{P} = \begin{bmatrix} 0 & \frac{1}{B} & \dots & \frac{1}{B} & 0 & 0 & 0 & 0 \dots 0 \\ 0 & 0 & \dots & 0 & 1 & 0 & 0 & 0 \dots 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \vdots & \vdots \ddots \vdots \\ 0 & 0 & \dots & 0 & 1 & 0 & 0 & 0 \dots 0 \\ 0 & 0 & \dots & 0 & 0 & p & (1-p) & 0 \dots 0 \\ 0 & 0 & \dots & 0 & 0 & 0 & 0 & 1 \dots 0 \\ 0 & 0 & \dots & 0 & 0 & 0 & 0 & 1 \dots 0 \\ 0 & 0 & \dots & 0 & 0 & 0 & 0 & 0 \dots 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \vdots & \vdots \ddots \vdots \\ 1 & 0 & \dots & 0 & 0 & 0 & 0 & 0 \dots 0 \end{bmatrix}$$

which can be done by solving $\boldsymbol{\pi} = \boldsymbol{\pi}\mathbf{P}$ verifying $\sum \boldsymbol{\pi}_i = 1$. Existing verification solutions, such as PARAM [25], can handle symbolic entries within the stochastic matrix $\mathbf{P}$ in order to find parametric solutions. However, to our knowledge, matrices with *parametric structures* (*i.e.* when the size depends on some parameters; in this case B and S) are out of the scope of existing solutions.

Our analysis can find solutions for such problems. When applied on this particular example, it infers in finite time the following bounds of $\boldsymbol{\pi}_{ack}$ in terms of parameters B, S and p:

$$\frac{B^2(p-1) - B(p-3) + 2(p-1)}{3B^2 + 2BS + B + 4} \leq \boldsymbol{\pi}_{ack} \leq \frac{\left(B^2 - B + 2\right)p}{3B^2 + 2BS + B + 4} \tag{1}$$

Since $\Gamma = \boldsymbol{\pi}_{ack}$, this parametric interval is guaranteed to cover all possible values of the goodput.

To obtain the invariant (1), we first construct a computable, finite-size over-approximation of the concrete chain using a novel domain of *abstract Markov chains*. We proceed by abstract interpretation of the program and we obtain the abstract Markov chain shown in Fig. 2b. Each abstract state over-approximates a set of states of the concrete Markov chain by identifying their (*i*) common program location, (*ii*) the invariant of reachable memory environments and (*iii*) the number of time ticks ν spent in such configuration. For instance, the transition $\langle l_4, \texttt{ack} \geq 0 \wedge \texttt{pkt} \geq \texttt{ack}, \nu = 1\rangle \xrightarrow{\frac{1}{B}} \langle l_7, \texttt{ack} \geq 0 \wedge \texttt{pkt} \geq \texttt{ack} \wedge \texttt{t} = 1, \nu = 1\rangle$ represents the case of choosing a backoff window of length 1, while $\langle l_4, \texttt{ack} \geq 0 \wedge \texttt{pkt} \geq \texttt{ack}, \nu = 1\rangle \xrightarrow{\frac{B-1}{B}} \langle l_7, \texttt{ack} \geq 0 \wedge \texttt{pkt} \geq \texttt{ack} \wedge \texttt{t} \in [2, B], \nu = \texttt{t}\rangle$ aggregates the remaining $B - 1$ cases.

Thanks to a novel widening operator, we ensure the finite size of the abstract chain and the convergence of computations in finite time. After convergence, we extract from this abstract chain a number of *distribution invariants* that characterize the boundaries of the stationary distribution vector $\boldsymbol{\pi}$. These invariants are represented as a parametric system of linear inequalities where the unknowns are the entries of $\boldsymbol{\pi}$ partitioned with respect to the abstract states of the abstract chain, and the coefficients are functions of the program parameters. Using a resolution method based on a parametric Fourier-Motzkin elimination, we obtain the invariant (1). ◇

Generalized Lumping. One of the most important challenges that hamper the use of Markov chains in modeling real-life systems is the state space explosion problem. The lumping technique [28] aims to reduce the size of a Markov chain by aggregating states into partitions in a way that allows to establish a link between the quantitative properties the original chain and the lumped one. The main challenge of this approach is to find the appropriate partitioning that preserves (partially) the *Markov property* of the lumped chain [6]. This fundamental property stipulates that the determination of future aggregate states should depend only upon the present aggregate state, not the past ones. This allows us to take benefit from classic results of Markov chains on the lumped process, but limits the application scope of the technique to a narrow range of partitioning policies.

Our extraction and resolution method of distribution invariants – not being limited to the case of communication protocols only – can be considered also as a *generalized lumping technique* of arbitrary Markovian processes. Indeed, our method does not impose any condition on the input chain and can be applied using any partitioning policy, even if the resulting lumped chain violates the Markov property. This represents a key missing property in existing lumping techniques because it decouples the analysis from the partitioning policies, which offers a means to adjust the efficiency/precision tradeoff while keeping the soundness guarantee in all cases.

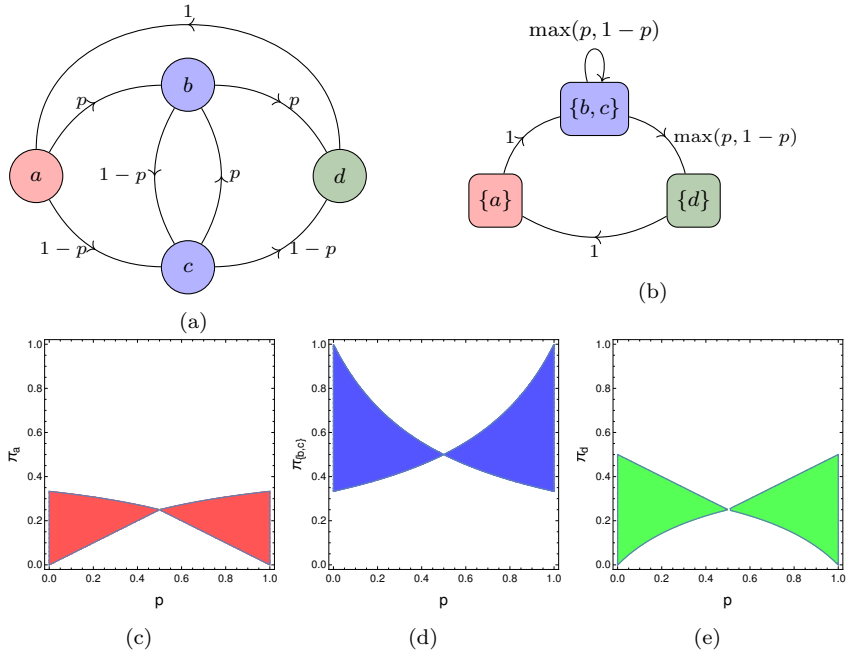


Fig. 3: Generalized Markov chain lumping. (a) Original chain. (b) Lumped chain. (c) Variation bounds of $\boldsymbol{\pi}_a$. (d) Variation bounds of $\boldsymbol{\pi}_b + \boldsymbol{\pi}_c$. (e) Variation bounds of $\boldsymbol{\pi}_d$.

Example 2 Consider the example Markov chain depicted in Fig. 3a and assume the partitioning $P = \{\{a\}, \{b, c\}, \{d\}\}$. We notice that this chain is not lumpable w.r.t. P since the states of the partition $\{b, c\}$ do not preserve the Markov property. Indeed, the probability distributions of b and c for choosing the future partitions are different and cannot be merged into a single valid probability distribution, except for the case when $p = \frac{1}{2}$.

Our method does not impose such restrictions, so we can construct the abstract Markov chain shown in Fig. 3b that respects the partitioning P but violates the Markov property. This is reflected by the non-standard outgoing transition probabilities $\{b, c\} \overset{\max(p,1-p)}{\rightarrow} \{b, c\}$ and $\{b, c\} \overset{\max(p,1-p)}{\rightarrow} \{d\}$, that cannot construct a valid probability distribution since their sum exceeds 1. Informally, such a non-standard transition $A \overset{\omega}{\rightarrow} B$ means that the maximal outgoing probability from a state in partition A to any state in partition B does not exceed the probability ω.

By constructing the distribution invariants of this abstract Markov chain and resolving the obtained parametric linear system, we find the symbolic bounds of $\boldsymbol{\pi}_a$, $\boldsymbol{\pi}_a + \boldsymbol{\pi}_a$ and $\boldsymbol{\pi}_d$ shown in Figs. 3c, 3d and 3e respectively. We notice that the precision of the obtained results, expressed as the distance between the upper and the lower bound, varies depending on the value of p. However, the exact solution is found when $p = \frac{1}{2}$, which corresponds to the case when the chain is lumpabale. ◇

```
1  pkt = 0;
2  ack = 0;
3  while(1) {
4    data = sense();
5    //Uniform backoff
6    t = uniform(1, B);
7    ndwait(t);
8    //Tx with ack
9    if (unicast(data))
10     ack++;
11   pkt++;
12   //Save energy
13   ndwait(S);
14 }
```

(a)

```
15 //Unicast model
16 #define TX_DELAY 1
17 #define RX_DELAY 1
18
19 bool unicast(int data)
       {
20   //Emulate tx
21   wait TX_DELAY;
22   //Emulate ack rx
23   rx = bernoulli();
24   wait RX_DELAY;
25   return rx;
26 }
```

(b)

```
27 //Model of clock drifts
28
29 void ndwait(int t) {
30   if ⍰ {
31     t = t + 1;
32   }
33   wait t;
34 }
```

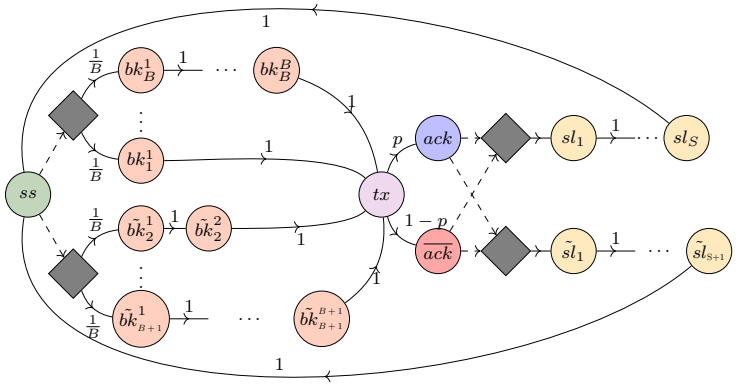
(c)

Fig. 4: (a) Example of a non-deterministic backoff-based transmission protocol. (b) hardware model of the `unicast` built-in function. (c) hardware clock model with non-deterministic drifts.

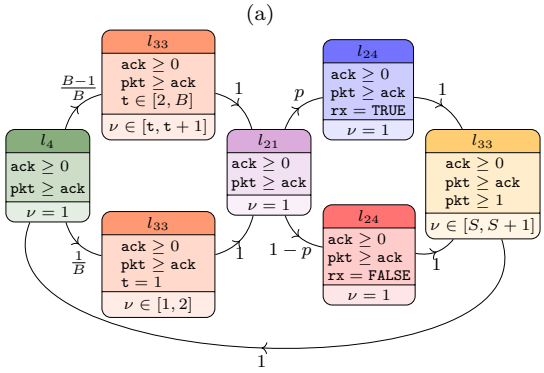
Non-determinism. When the stochastic behavior of the system is not totally known, non-determinism is a valuable tool to overcome this lack of information. However, while probability and non-determinism have been widely studied separately in the literature of program analysis, there exist only few works that can mix them within a same computable semantics [38,11]. The main challenge for such analyses is the difficulty to reason about program traces in terms of (possibly unbounded) sets of heterogeneous probability distributions in order to infer interesting quantitative information.

A well-known stochastic tool supporting both probabilities and non-determinism is the model of *Markov decision processes* (MDP) [42]. Informally, a MDP is an extended Markov chain model in which each state can decide which probability distribution to use before choosing the next state according to it. In other words, at each state of the MDP, a non-deterministic choice from a finite number of transition distributions is allowed, while in (deterministic) Markov chains only one distribution can be used. Since (*i*) an MDP can be viewed as an unbounded set of (possibly infinite) Markov chains as we will see later, and (*ii*) our abstract domain can over-approximate sets of Markov chains of arbitrary sizes, our analysis can be easily extended to handle MDPs, which allows a *natural* semantics formalization for both pure non-determinism and probabilities.

Example 3 Let us go back to our first motivating example in order to introduce non-deterministic choices. Assume that our target embedded system is equipped with a hardware clock that may exhibit occasional drifts, but the distribution of these events is unknown. We model this phenomenon by redefining `wait` using the non-deterministic boolean operator ⍰ as shown in Fig. 4c. For illustration purposes, we use a basic additive drift model that simply increments the clock by one tick in a non-deterministic way. Despite being unrealistic, it simplifies the presentation of the main challenges of this problem. The corresponding MDP is



(a)



(b)

Fig. 5: (a) Markov decision process of the protocol. Diamond nodes represent non-deterministic choices. (b) Associated abstract Markov chain.

depicted in Fig. 5a. We can see that with this small change, the structure of model increased significantly which makes it more difficult to study analytically [1].

The abstract Markov chain inferred by our analysis is – on the other hand – quite similar to the deterministic case, as shown in Fig. 5b. In fact, the structure remained the same while the state invariants have changed according to the introduction of the non-determinism. More particularly, the sojourn time in the backoff and sleep states (identified by the program location l_{33}) reflects such non-determinism with the interval invariants. Using the same resolution method of distribution invariants as the deterministic case, we obtain the new goodput in-

[1] Note that the hardware model of the function `unicast` employs the `wait` primitive since it emulates the physical delay of wireless transmissions, which is not affected by the drifts of the system clock.

variant:

$$\frac{B^2(p-1)(S+1)+2BS+3(p-1)(S+1)}{3B^2(S+1)+2B\left(S^2+3S+1\right)+5(S+1)} \leq \Gamma \leq \frac{\left(B^2+3\right)p(S+1)}{3B^2(S+1)+2B\left(S^2+3S+1\right)+5(S+1)}$$

◇

Contributions. To sum up, we propose a novel static analysis by abstract interpretation based on three main contributions:

1. First, we introduce a novel notion of *abstract Markov chains* that approximates a set of discrete time Markov chains. These abstract chains are inferred automatically by analyzing the source code of the program. For the sake of clarity, we start by limiting the scope of the analysis to probabilistic programs without non-deterministic choices. Thanks to a novel widening algorithm, these chains are guaranteed to have a finite size while covering all possible probabilistic traces of the program.
2. Our second contribution is a result for extracting *distribution invariants* from an abstract Markov chain in the form of a system of parametric linear inequalities for bounding the concrete stationary distribution. Using a parametric-version of the Fourier-Motzkin elimination algorithm, we can infer symbolic and guaranteed bounds of the property of interest.
3. Finally, we extend the previous analysis in order to support programs with non-deterministic choices and we show how we can preserve the soundness of the extracted distribution invariants.

The foundations of our ideas have been previously described in [40]. The present article extends our previous work by the support of non-determinism and the full correctness proof of the distribution invariants. Also, we provide a more comprehensive description of the semantics and a discussion of additional experimental results.

Limitations. Our approach is still in a preliminary development phase and presents some limitations. The analysis supports only discrete probability distributions, such as Bernoulli and discrete uniform distributions. Our model supports symbolic parameters of these distributions, but does not support dynamic modification of the parameter of a Bernoulli distribution during execution. We limit the description herein to a simple C-like language and we do not support yet the analysis of real-world implementations. Finally, we support the analysis of only one node of the network. The interactions via messages with the remaining nodes is not addressed in this work.

Outline. The remaining of the paper is organized as follows. We present in Section 2 the concrete semantics of the deterministic analysis. Section 3 introduces the domain of abstract Markov chains and we detail in Section 4 the method to extract the stationary distribution invariants from an abstract chain and how we can infer symbolic bounds of the property of interest. We show in Section 5 how we can extend the analysis to support non-deterministic programs. The results of the preliminary experiments are presented in Section 6. We discuss the related work in Section 7 and we conclude the paper in Section 8.

$$
\begin{array}{rll}
expr & ::= & \texttt{TRUE} \mid \texttt{FALSE} \mid i \in \mathbb{Z} \qquad \wr constants \wr \\
& \mid & id \qquad \wr identifer \wr \\
& \mid & \boxdot expr \qquad \wr unaray\ operations \wr \\
& \mid & expr \boxdot expr \qquad \wr binary\ operations \wr \\
& \mid & \texttt{bernoulli}_{l \in \mathcal{L}}() \qquad \wr discrete\ Bernoulli\ choice \wr \\
& \mid & \texttt{uniform}_{l \in \mathcal{L}}(expr, expr) \qquad \wr discrete\ uniform\ sample \wr \\
\\
stmt & ::= & id = expr \qquad \wr assignment \wr \\
& \mid & \texttt{wait}_{l \in \mathcal{L}}\ expr \qquad \wr idle\ time \wr \\
& \mid & \texttt{if}\ (expr)\ stmt\ \texttt{else}\ stmt \qquad \wr conditional \wr \\
& \mid & \texttt{while}\ (expr)\ stmt \qquad \wr loop \wr
\end{array}
$$

Fig. 6: Syntax of PSimpl.

2 Concrete Semantics

We consider communication protocols that can be represented as (possibly infinite) discrete time Markov chains. For the clarity of presentation, we target a simple probabilistic language PSimpl with a limited, albeit sufficient, set of features. The language supports sampling from *Bernoulli* and *uniform* distributions, which are widely used in communication protocols. We consider a discrete time scale and we assume that all statements are instantaneous except for a statement `wait`. In the following, we describe the syntax of the language, its concrete semantics and the computation method of the stationary distribution associated to a probabilistic program.

2.1 Language Syntax

We give in Fig. 6 the syntax of PSimpl. We consider boolean and integer expressions, with standard constructs such as boolean/integer constants $c \in \mathcal{V} \stackrel{\text{def}}{=} \mathbb{B} \cup \mathbb{Z}$, variables $id \in \mathcal{X}$ or results of unary/binary operations. PSimpl supports common statements such as assignments, `if` conditionals and `while` loops, in addition to the statement $\texttt{wait}_l\ e$ that models the fact that the program spends e ticks in the current control location l. Probabilistic choices are provided by two built-in functions $\texttt{uniform}_l$ and $\texttt{bernoulli}_l$, where the annotation $l \in \mathcal{L}$ represents the call site location. The function $\texttt{uniform}_l(e_1, e_2)$ draws a random integer value from a discrete uniform distribution over the interval $[e_1, e_2]$. A call to the function $\texttt{bernoulli}_l()$ returns a boolean value according to a Bernoulli distribution with parameter p_l. Note that this parameter is not an argument of the function $\texttt{bernoulli}_l()$ because our analysis does not support dynamic modification of the parameter of Bernoulli distributions at runtime. Nevertheless, p_l is symbolic and can represent any range in $[0, 1]$. To sum up, our analysis can accept as parameter p_l an interval of values, and will give a result that is sound for any input value of p_l within this interval, as long as p_l is not modified during the execution.

2.2 Markov Chains

PSimpl allows defining programs representing discrete time Markov chains over possibly unbounded state spaces. Two key features of the language are important

to achieve that. First, the ability to draw values from probability distributions allows creating probabilistic control flows, similarly to Markov chains. This leads us to the definition of the following notion of *events*:

Definition 1 (Events) The set of all possible random outcomes that can occur during execution defines the set of events:

$$\Xi \stackrel{\text{def}}{=} \{\, b_l, \bar{b}_l \mid l \in \mathcal{L}_{\downarrow \texttt{bernoulli}} \,\} \cup \{\, u_l^{i,a,b} \mid l \in \mathcal{L}_{\downarrow \texttt{uniform}} \wedge i \in [a,b] \,\}$$

where $\mathcal{L}_{\downarrow \texttt{f}} \subseteq \mathcal{L}$ is the set of call site locations of function `f`. Events b_l and $\bar{b}_l$ denote the two outcomes of a call to $\texttt{bernoulli}_l()$. An event $u_l^{i,a,b}$ denotes the ith outcome of a call $\texttt{uniform}_l(e_1, e_2)$, where a and b are the evaluation in the current execution environment of e_1 and e_2 respectively.

The second feature of the language is the function `wait` that expresses time elapse. While communication protocols frequently use waits of more that one time unit, this can be modeled without loss of generality as sequences of waits of one time unit, hence classic Markov chains assume, for simplicity, that the sojourn time in each state is always one. However, an important feature of our language is the ability to use symbolic expressions as parameters of wait, hence, this simplification is no longer possible: we need to explicitly tag each state of our Markov chains with a symbolic, possibly non-unit sojourn time.

Dually, all non-waiting operations in a communication protocol correspond to a change of program state that does not advance time, and is thus not observable at the time scale of Markov chains. Therefore, we adopt a *two-level trace semantics*, as introduced by Radhia Cousot in her thesis [12, Section 2.5.4], that makes a distinction between observable and non-observable transitions. We give here a definition of these two types of traces adapted to our settings:

Definition 2 (Observable states) Let $\mathcal{E} \stackrel{\text{def}}{=} \mathcal{X} \to \mathcal{V}$ be the set of memory environments mapping variables in $\mathcal{X}$ to their values in $\mathcal{V}$. An *observable state* $(l, \rho, \nu) \in \Sigma \stackrel{\text{def}}{=} \mathcal{L} \times \mathcal{E} \times \mathbb{N}^+$ represents the memory environment ρ that the program reaches at location l while spending a sojourn time of ν time ticks.

Definition 3 (Scenarios) A sequence of non-observable transitions is called a *scenario* and is defined as $\omega \in \Omega \stackrel{\text{def}}{=} \Xi^*$ expressing sequences of random events that occur between two observable states. In the sequel, we denote by ε the empty scenario word.

Definition 4 (Observable traces) The observable traces are the set $\mathcal{T}_\Sigma^\Omega \stackrel{\text{def}}{=} \{\, \sigma_0 \stackrel{\omega_1}{\to} \sigma_1 \stackrel{\omega_2}{\to} \cdots \mid \sigma_i \in \Sigma \wedge \omega_i \in \Omega \,\} \cup \{\, \epsilon \,\}$ composed of transitions among observable states labeled with scenarios. An empty observable trace is denoted by ϵ.

2.3 Semantics Domain

The concrete semantics domain of our analysis is defined as $\mathcal{D} \stackrel{\text{def}}{=} \wp(\mathcal{T}_\Sigma^\Omega \times \mathcal{E} \times \Omega)$. An element $(\tau, \rho, \omega) \in \mathcal{T}_\Sigma^\Omega \times \mathcal{E} \times \Omega$ encodes the set of traces reaching a given program location and is composed of three parts: (i) the observable trace $\tau \in \mathcal{T}_\Sigma^\Omega$ containing the past transitions of the Markov chain before the current time tick,

$$
\begin{array}{lcl}
\mathbb{E}\llbracket\, c \in \mathcal{V}\,\rrbracket R & \stackrel{\text{def}}{=} & \{\,(c,\omega) \mid (\tau,\rho,\omega) \in R\,\} \\
\mathbb{E}\llbracket\, id \in \mathcal{X}\,\rrbracket R & \stackrel{\text{def}}{=} & \{\,(\rho(id),\omega) \mid (\tau,\rho,\omega) \in R\,\} \\
\mathbb{E}\llbracket\, \boxdot e\,\rrbracket R & \stackrel{\text{def}}{=} & \{\,(\boxdot v,\omega_1) \mid (\tau,\rho,\omega) \in R \wedge (v,\omega_1) \in \mathbb{E}\llbracket\, e\,\rrbracket\{\,(\tau,\rho,\omega)\,\}\,\} \\
\mathbb{E}\llbracket\, e_1 \boxdot e_2\,\rrbracket R & \stackrel{\text{def}}{=} & \{\,(v_1 \boxdot v_2,\omega_2) \mid (\tau,\rho,\omega) \in R \wedge \\
 & & \quad (v_1,\omega_1) \in \mathbb{E}\llbracket\, e_1\,\rrbracket\{\,(\tau,\rho,\omega)\,\} \wedge (v_2,\omega_2) \in \mathbb{E}\llbracket\, e_2\,\rrbracket\{\,(\tau,\rho,\omega_1)\,\}\,\} \\
\mathbb{E}\llbracket\, \texttt{bernoulli}_l()\,\rrbracket R & \stackrel{\text{def}}{=} & \{\,(b,\omega.\xi) \mid (\tau,\rho,\omega) \in R \wedge (b,\xi) \in \{\,(\texttt{TRUE},b_l),(\texttt{FALSE},\bar{b}_l)\,\}\,\} \\
\mathbb{E}\llbracket\, \texttt{uniform}_l(e_1,e_2)\,\rrbracket R & \stackrel{\text{def}}{=} & \{\,(n,\omega_2.u_l^{n,a,b}) \mid (\tau,\rho,\omega) \in R \wedge (a,\omega_1) \in \mathbb{E}\llbracket\, e_1\,\rrbracket\{\,(\tau,\rho,\omega)\,\} \wedge \\
 & & \quad (b,\omega_2) \in \mathbb{E}\llbracket\, e_2\,\rrbracket\{\,(\tau,\rho,\omega_1)\,\} \wedge n \in [a,b]\,\}
\end{array}
$$

Fig. 7: Concrete semantics of expressions.

$$
\begin{array}{lcl}
\mathbb{S}\llbracket\, x = e\,\rrbracket R & \stackrel{\text{def}}{=} & \{\,(\tau,\rho[x \mapsto v],\omega_1) \mid (\tau,\rho,\omega) \in R \wedge (v,\omega_1) \in \mathbb{E}\llbracket\, e\,\rrbracket\{\,(\tau,\rho,\omega)\,\}\,\} \\
\mathbb{S}\llbracket\, \texttt{if}\ (e)\ s_1\ \texttt{else}\ s_2\,\rrbracket R & \stackrel{\text{def}}{=} & (\mathbb{S}\llbracket\, s_1\,\rrbracket \circ \mathit{filter}(e))R \cup (\mathbb{S}\llbracket\, s_2\,\rrbracket \circ \mathit{filter}(\neg e))R \\
\mathbb{S}\llbracket\, \texttt{while}\ (e)\ s\,\rrbracket R & \stackrel{\text{def}}{=} & \mathit{filter}(\neg e)(\mathrm{lfp}\ \lambda X.\, R \cup (\mathbb{S}\llbracket\, s\,\rrbracket \circ \mathit{filter}(e))X) \\
\mathbb{S}\llbracket\, \texttt{wait}_l\ e\,\rrbracket R & \stackrel{\text{def}}{=} & \{(\tau \xrightarrow{\omega_1} (l,\rho,\nu),\rho,\varepsilon) \mid (\tau,\rho,\omega) \in R \wedge (\nu,\omega_1) \in \mathbb{E}\llbracket\, e\,\rrbracket\{\,(\tau,\rho,\omega)\,\}\} \\
\text{where:} & & \\
\mathit{filter}(e)R & \stackrel{\text{def}}{=} & \{\,(\tau,\rho,\omega_1) \mid (\tau,\rho,\omega) \in R \wedge (\texttt{TRUE},\omega_1) \in \mathbb{E}\llbracket\, e\,\rrbracket\{\,(\tau,\rho,\omega)\,\}\,\}
\end{array}
$$

Fig. 8: Concrete semantics of statements.

(ii) the current memory environment $\rho \in \mathcal{E}$, and (iii) the partial scenario $\omega \in \Omega$ of non-observable random events that occurred between the last tick and the current execution moment.

To obtain the set of all traces of a program, we proceed by induction on its abstract syntax tree using a set of concrete evaluation functions $\mathbb{E}\llbracket\, . \,\rrbracket \in \mathcal{D} \to \wp(\mathcal{V} \times \Omega)$ for expressions and a set of concrete transfer functions $\mathbb{S}\llbracket\, . \,\rrbracket \in \mathcal{D} \to \mathcal{D}$ for statements as follows:

Expressions. We give in Fig. 7 the concrete semantics of expression evaluation over a concrete element $R \in \mathcal{D}$. Since expressions do not generate new time ticks but may involve probabilistic events, evaluation functions $\mathbb{E}\llbracket\, . \,\rrbracket \in \mathcal{D} \to \wp(\mathcal{V} \times \Omega)$ return a set of evaluated values along with the updated scenarios. Two cases are particularly interesting. The semantics of a call $\texttt{bernoulli}_l()$ is to fork the current partial scenarios ω depending on the result of the function. We append the event b_l in the true case, or the event $\bar{b}_l$ in the false case and we return the corresponding truth value. For the expression $\texttt{uniform}_l(e_1,e_2)$, we also fork the partial scenarios, but the difference is that the number of branches depends on the evaluations of e_1 and e_2 in the current memory environment. More precisely, the number of forks corresponds to the number of integer points between the values of e_1 and e_2.

Statements. The semantics of statements is shown in Fig. 8. Most cases have a standard definition. The assignment statement updates the current memory environment by mapping the left-hand variable to the evaluation of the expression. For the `if` statement, we filter the current environments depending on the evaluation of the condition, and we analyze each branch independently before merging

the results. Also, a loop statement is formalized as a fixpoint on the sequences of body evaluation with a filter to extract the iterations violating the loop condition. Finally, the semantics of the statement $\texttt{wait}_l\ e$ is to extend the observable traces with a new transition to a state where the sojourn time is equal to the evaluation of the expression e. The label of this new transition is simply the computed partial scenario, which is reset to the empty word ε since we keep track of events traces only between two `wait` statements.

2.4 Stationary Distributions

Informally, the stationary distribution of a discrete time Markov chain, also called limiting or steady-state distribution, is the probability vector giving the proportion of time that the chain will spend in each state after running for a sufficiently long period. However, in the case where a program $\mathsf{P} \in \mathit{stmt}$ contains uninitialized parameters in a given initial set of states $I \subseteq \mathcal{E}$, the resulting traces $\mathbb{S}\,[\![\,\mathsf{P}\,]\!]$ may represent several distinct Markov chains. More precisely, each initial environment $\rho \in I$ corresponds to exactly one Markov chain. Therefore, we will not obtain a single stationary distribution corresponding to P, but a parametric stationary distribution function that maps initial values of parameters to the distribution of the corresponding chain.

Let us define the extraction function $\mathbb{M}\,[\![\,\mathsf{P}\,]\!] \in \mathcal{E} \to \wp(\mathcal{T}_\Sigma^\Omega)$ that computes the traces of a single Markov chain as the output of $\mathbb{S}\,[\![\,\mathsf{P}\,]\!]$ on a given initial environment and an empty trace:

$$\mathbb{M}\,[\![\,\mathsf{P}\,]\!]\rho \stackrel{\text{def}}{=} \{\,\tau \mid \exists(\tau, \rho', \omega) \in \mathbb{S}\,[\![\,\mathsf{P}\,]\!]\{\,(\epsilon, \rho, \varepsilon)\,\}\,\}$$

Let $\boldsymbol{\pi}\,[\![\,\mathsf{P}\,]\!] \in \mathcal{E} \to \Sigma \to [0,1]$ be the probability vector representing the stationary distribution of the Markov chain corresponding to a given initial environment. To find this distribution, we first define the notion of *scenario probability*:

Definition 5 (Scenario probability) The function $\Pr\,[\![\,\omega\,]\!] \in \mathcal{E} \to [0,1]$ gives the probability of a scenario $\omega \in \Omega$ by combining the probabilities of its individual events. Let p_l be a symbolic variable representing the probability parameter of a Bernoulli distribution at call site $l \in L$. We define $\Pr\,[\![\,\omega\,]\!]$ by structural induction as follows:

$$\Pr\,[\![\,\omega\,]\!]\rho \stackrel{\text{def}}{=} \begin{cases} 1 & \text{if } \omega = \varepsilon \\ \rho[p_l] & \text{if } \omega = b_l \\ 1 - \rho[p_l] & \text{if } \omega = \bar{b}_l \\ \frac{1}{b-a+1} & \text{if } \omega = u_l^{i,a,b} \\ \Pr\,[\![\,\omega\,]\!]\rho \Pr\,[\![\,\xi\,]\!]\,\rho & \text{if } \omega = \omega\xi \end{cases}$$

Empty scenario ε has probability 1 since it represents a deterministic choice. The probability of a Bernoulli outcome b_l is the evaluation of the associated parameter variable p_l in the current environment ρ. Outcomes $u_l^{i,a,b}$ of a uniform distribution are equiprobable over the interval $[a,b]$. Finally, the probability of a composed sequence $\omega\xi$ is the joint probability of ω and ξ.

Afterwards, we construct a non-standard stochastic matrix that characterizes the transitions between observable states:

Definition 6 (Non-standard stochastic matrix) We denote by $\mathbb{P}[\![\,\mathsf{P}\,]\!] \in \mathcal{E} \to \Sigma \times \Sigma \to \mathbb{R}^+$ the square matrix function defined as:

$$\mathbb{P}[\![\,\mathsf{P}\,]\!] \stackrel{\text{def}}{=} \lambda\rho.\, \lambda(s_i, s_j).\, \frac{\nu_j}{\nu_i} \sum_{s_i \stackrel{\omega}{\to} s_j \in \mathbb{M}[\![\,\mathsf{P}\,]\!]\rho} \Pr[\![\,\omega\,]\!]\rho$$

where ν_i and ν_j denote the sojourn time in states s_i and s_j respectively.

Note that this definition differs slightly from the classic construction of the stochastic transition matrix of discrete time Markov chains. This is due to the fact that states in our model embed the information of sojourn time ν, which is assumed to be equal to one time unit in classic Markov chains.

In order to compute the family of distributions $\{\, \boldsymbol{\pi}[\![\,\mathsf{P}\,]\!]\rho \mid \rho \in I \,\}$ for a set of initial environments $I \subseteq \mathcal{E}$, as for the classic matrix, we solve the system:

$$\forall \rho \in I : \begin{cases} \forall s_i \in \Sigma : \boldsymbol{\pi}[\![\,\mathsf{P}\,]\!](\rho)(s_i) = \sum_{s_j \in \Sigma} \boldsymbol{\pi}[\![\,\mathsf{P}\,]\!](\rho)(s_j) \times \mathbb{P}[\![\,\mathsf{P}\,]\!](\rho)(s_j, s_i) \\ \sum_{s \in \Sigma} \boldsymbol{\pi}[\![\,\mathsf{P}\,]\!](\rho)(s) = 1 \end{cases} \tag{2}$$

Since the reachable state space in Σ can be unbounded, both $\mathbb{P}[\![\,\mathsf{P}\,]\!]$ and $\boldsymbol{\pi}[\![\,\mathsf{P}\,]\!]$ may not be computable. In addition, system designers are generally interested in analyzing the system for wide ranges of parameter settings I, which makes the problem more difficult. In the following, we propose a computable abstraction of Markov chains to over-approximate the traces $\{\, \mathbb{M}[\![\,\mathsf{P}\,]\!]\rho \mid \rho \in I \,\}$ for any initial setting. Afterwards, we show how we can construct a finite stochastic matrix using information provided by our abstract chain, that helps infer symbolic, guaranteed bounds of all distributions $\{\, \boldsymbol{\pi}[\![\,\mathsf{P}\,]\!]\rho \mid \rho \in I \,\}$.

3 Abstract Semantics

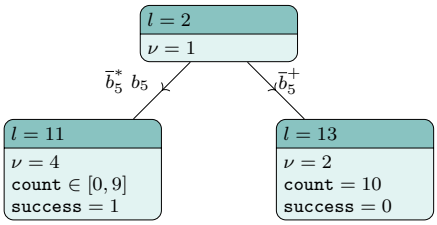
In order to analyze a program statically, we need a computable abstraction of the concrete semantics domain $\mathcal{D}$. The basic idea is to first partition the set of observable program states $\wp(\mathcal{L} \times \mathcal{E} \times \mathbb{N})$ with respect to the program locations, resulting into the intermediate abstraction $\mathcal{L} \to \wp(\mathcal{E} \times \mathbb{N})$. For each location, the set of associated environments is then abstracted with a stock numerical domain $\mathfrak{N}^\sharp$, by considering the sojourn time as a program variable ν. We obtain the abstract states domain $\Sigma^\sharp \stackrel{\text{def}}{=} \mathcal{L} \to \mathfrak{N}^\sharp$. As a consequence of this partitioning, observable states at the same program location will be merged. Therefore, we obtain a special structure in which observable abstract states are connected through possibly multiple scenarios coming from the merged concrete states.

Example 4 We illustrate this fact in Fig. 9a depicting a more complex probabilistic modeling of the previous `sense()` function using a bounded geometric distribution that works as follows. We start by warming up the sensing device during one tick. After that, we check whether the sensor detects some external activity (high temperature, sound noise, *etc.*) and we perform this check for at most 10 times. We assume that these external activities follow a Bernoulli distribution. At the end, we perform some processing during 4 ticks in case of detection and 2 ticks in case of non-detection.

We can see in Fig. 9b that between the observable program locations 2 and 11 many scenarios are possible, which are abstracted with the regular expression

```
1  void* sense() {
2    warmup(); //~ wait 1
3    int count = 0, success = 0;
4    do {
5      if (check()) //~ bernoulli()
6        success = 1;
7      else
8        count++;
9    } while (count < 10 && !success);
10   if (success)
11     return doDetection(); //~ wait 4
12   else
13     return doAbsence(); //~ wait 2
14 }
```

(a)



(b)

Fig. 9: (a) A simple probabilistic model for the `sense()` function. (b) An abstraction of observable traces represented as a hierarchical automaton.

$\overline{b}_5^*\ b_5$ that encodes the pattern of having a number of Bernoulli failure outcomes at line 5 before a successful one. However, between lines 2 and 13, we can have only a sequence of failures, which is expressed as $\overline{b}_5^+$. ◇

The presence of these multi-word transitions leads to a *hierarchical automata structure* representing an automaton over an alphabet of automata. On the one hand, one automata structure is used to encode the transitions between observable abstract states. On the other hand, for each observable transition, another automata structure is used to encode the regular expressions of scenarios connecting the endpoints of the transition.

In the section, we formalize this structure through our novel domain of abstract Markov chains. For modularity reasons, we begin by presenting a generic data structure, called *abstract automata*, for representing languages over an abstract alphabet. Afterwards, we employ this data structure to instantiate two abstract domains that will be composed into a two-level hierarchy. At a bottom level, we develop an abstract scenario domain as an automaton over an alphabet of abstract probability events to over-approximate traces of non-observable states. On top of it, we build our abstract Markov chain domain as an automaton over the alphabet of abstract scenarios.

3.1 Abstract Automata

Le Gall et al. proposed a lattice automata domain [31] to represent words over an abstract alphabet having a lattice structure. We extend this domain to support also abstraction at the state level by merging states into abstract states, which is important to approximate Markov chains having an infinite state space.

Let $\mathfrak{A}^\sharp$ be an abstract alphabet domain and $\mathfrak{S}^\sharp$ an abstract state domain. We assume that $\mathfrak{A}^\sharp$ is an abstraction of some concrete alphabet symbols $\mathfrak{A}$, having a concretization function $\gamma_{\mathfrak{A}^\sharp} \in \mathfrak{A}^\sharp \to \wp(\mathfrak{A})$, a partial order $\sqsubseteq_{\mathfrak{A}^\sharp}$, a join operator $\sqcup_{\mathfrak{A}^\sharp}$, a meet operator $\sqcap_{\mathfrak{A}^\sharp}$, a least element $\bot_{\mathfrak{A}^\sharp}$ and a widening operator $\triangledown_{\mathfrak{A}^\sharp}$. Similarly,

$\mathfrak{S}^\sharp$ is assumed to be an abstraction of some concrete states $\mathfrak{S}$ equipped with a concretization function $\gamma_{\mathfrak{S}^\sharp} \in \mathfrak{S}^\sharp \to \wp(\mathfrak{S})$, a partial order $\sqsubseteq_{\mathfrak{S}^\sharp}$, a join operator $\sqcup_{\mathfrak{S}^\sharp}$, a least element $\bot_{\mathfrak{S}^\sharp}$ and a widening operator $\triangledown_{\mathfrak{S}^\sharp}$.

We define the functor domain $\mathscr{A}(\mathfrak{A}^\sharp, \mathfrak{S}^\sharp)$ of abstract automata over alphabet $\mathfrak{A}^\sharp$ and states $\mathfrak{S}^\sharp$ as follows:

Definition 7 (Abstract automata) An abstract automaton $A \in \mathscr{A}(\mathfrak{A}^\sharp, \mathfrak{S}^\sharp)$ is a finite state automaton $A = (S, s_0^\sharp, F, \Delta)$, where $S \subseteq \mathfrak{S}^\sharp$ is the set of states, $s_0^\sharp \in S$ is the initial state, $F \subseteq S$ is the set of final states and $\Delta \subseteq S \times \mathfrak{A}^\sharp \times S$ is the transition relation. The meaning of A is provided by two concretization functions:

1. The sets of accepted traces abstracted by A is given by the concretization function $\gamma^{\mathbf{T}} \in \mathscr{A}(\mathfrak{A}^\sharp, \mathfrak{S}^\sharp) \to \wp(\mathcal{T}_{\mathfrak{S}}^{\mathfrak{A}})$ defined by:

$$\gamma^{\mathbf{T}}_{\mathscr{A}}(A) \stackrel{\text{def}}{=} \{\, s_0 \xrightarrow{a_0} s_1 \dots \xrightarrow{a_{n-1}} s_n \mid \exists s_0^\sharp \xrightarrow{a_0^\sharp} s_1^\sharp \dots \xrightarrow{a_{n-1}^\sharp} s_n^\sharp \in \mathbf{T}(A) : \forall i \in [0,n] : s_i \in \gamma_{\mathfrak{S}^\sharp}(s_i^\sharp) \wedge \forall i \in [0,n) : a_i \in \gamma_{\mathfrak{A}^\sharp}(a_i^\sharp) \,\}$$

   where $\mathbf{T}(A) \stackrel{\text{def}}{=} \{\, s_0^\sharp \xrightarrow{a_0^\sharp} s_1^\sharp \dots \xrightarrow{a_{n-1}^\sharp} s_n^\sharp \mid \forall i \in [0,n) : (s_i^\sharp, a_i^\sharp, s_{i+1}^\sharp) \in \Delta \wedge s_n^\sharp \in F \,\}$ gives the set of traces accepted by A.
2. The set of accepted words abstracted by A is given by the concretization function $\gamma^{\mathbf{L}} \in \mathscr{A}(\mathfrak{A}^\sharp, \mathfrak{S}^\sharp) \to \wp(\mathfrak{A}^*)$ defined by:

$$\gamma^{\mathbf{L}}_{\mathscr{A}}(A) \stackrel{\text{def}}{=} \{\, a_0 a_1 \dots a_{n-1} \mid \exists a_0^\sharp a_1^\sharp \dots a_{n-1}^\sharp \in \mathbf{L}(A), \forall i \in [0,n) : a_i \in \gamma_{\mathfrak{A}^\sharp}(a_i^\sharp) \,\}$$

   where $\mathbf{L}(A) \stackrel{\text{def}}{=} \{\, a_0^\sharp a_1^\sharp \dots a_{n-1}^\sharp \mid \exists s_0^\sharp \xrightarrow{a_0^\sharp} s_1^\sharp \dots \xrightarrow{a_{n-1}^\sharp} s_n^\sharp \in \mathbf{T}(A) \,\}$ gives the set of words accepted by A.

This dual view of traces vs. words is important in our semantics since scenarios are considered as words (sequence of events) and observable traces as traces. Let us now define some important operators of the functor domain $\mathscr{A}$. In the following, we denote by $A = (S, s_0^\sharp, F, \Delta)$, $A_1 = (S_1, s_0^\sharp, F_1, \Delta_1)$ and $A_2 = (Q_2, q_0^\sharp, F_2, \Delta_2)$ three instances of $\mathscr{A}(\mathfrak{A}^\sharp, \mathfrak{S}^\sharp)$.

3.1.1 Order

To compare two abstract automata, we define the following simulation relation that extends the classic simulation concept found in transition systems by considering the abstraction in alphabet and states:

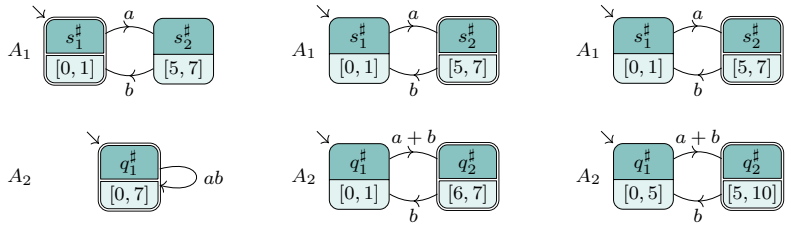
Definition 8 (Simulation relation) A binary relation $\mathcal{S}_{A_1}^{A_2} \subseteq \mathfrak{S}^\sharp \times \mathfrak{S}^\sharp$ is a simulation between A_1 and A_2 *iff* $\forall (s_1^\sharp, q_1^\sharp) \in \mathcal{S}_{A_1}^{A_2}$ we have $s_1^\sharp \sqsubseteq_{\mathfrak{S}^\sharp} q_1^\sharp$ and:

$$\forall s_1^\sharp \xrightarrow{a_1^\sharp} s_2^\sharp \in \Delta_1, \exists q_1^\sharp \xrightarrow{a_2^\sharp} q_2^\sharp \in \Delta_2 : \left(a_1^\sharp \sqsubseteq_{\mathfrak{A}^\sharp} a_2^\sharp\right) \wedge \left((s_2^\sharp, q_2^\sharp) \in \mathcal{S}_{A_1}^{A_2}\right)$$

Definition 9 (Partial order) Let $\preccurlyeq_{A_1}^{A_2}$ denotes the smallest simulation relation between A_1 and A_2 verifying $(s_0^\sharp, q_0^\sharp) \in \preccurlyeq_{A_1}^{A_2}$. We define the partial order relation $\sqsubseteq_{\mathscr{A}}$ as:

$$A_1 \sqsubseteq_{\mathscr{A}} A_2 \Leftrightarrow \preccurlyeq_{A_1}^{A_2} \neq \varnothing \wedge \forall (s^\sharp, q^\sharp) \in \preccurlyeq_{A_1}^{A_2} : s^\sharp \in F_1 \Rightarrow q^\sharp \in F_2$$

which means that A_2 should simulate and accept every accepted trace in A_1.



(a) $A_1 \not\sqsubseteq_{\mathscr{A}} A_2$ (b) $A_1 \not\sqsubseteq_{\mathscr{A}} A_2$ (c) $A_1 \sqsubseteq_{\mathscr{A}} A_2$

Fig. 10: Examples of order comparison between abstract automata.

Example 5 Consider the abstract automata shown in Fig. 10. For illustration purpose, we use the integer intervals domain as state abstraction, and the set of regular expressions over two symbols $\{a, b\}$ as alphabet abstraction.

In the first case (a), no simulation relation exists between A_1 and A_2 since the transition $\langle s_1^\sharp : [0,1]\rangle \xrightarrow{a} \langle s_2^\sharp : [5,7]\rangle$ cannot be simulated by the transition $\langle q_1^\sharp : [0,7]\rangle \xrightarrow{ab} \langle q_1^\sharp : [0,7]\rangle$ given that $a \not\sqsubseteq^\sharp_{\mathfrak{A}} ab$ violates the condition of Definition 8. In case (b), $A_1 \not\sqsubseteq_{\mathscr{A}} A_2$ because the transition $\langle s_1^\sharp : [0,1]\rangle \xrightarrow{a} \langle s_2^\sharp : [5,7]\rangle$ in A_1 cannot be simulated by the transition $\langle q_1^\sharp : [0,1]\rangle \xrightarrow{a+b} \langle q_2^\sharp : [6,7]\rangle$ since $[5,7] \not\sqsubseteq^\sharp_{\mathfrak{S}} [6,7]$. Note that in both cases (a) and (b), $\mathbf{L}(A_1) \subseteq \mathbf{L}(A_2)$, but this is not sufficient to verify the order relation $\sqsubseteq_{\mathscr{A}}$, in contrast to the automata domain proposed by Le Gall et al. [31]. Finally, in case (c), the condition of Definition (8) is fulfilled for every transition in A_1, which implies that $A_1 \sqsubseteq_{\mathscr{A}} A_2$. ◇

3.1.2 Join

To compute the union of two abstract automata A_1 and A_2, we need to extend the simulation-based traversal in a way to consider all traces of both automata, including those violating the simulation condition (*i.e.* traces belonging to one automaton only). To do so, we introduce the concept of *product relation* that builds a transition relation defined over the Cartesian product $\mathfrak{S}^\sharp \times \mathfrak{S}^\sharp$ that over-approximates the transitions that can be performed simultaneously by A_1 and A_2. A naive approximation is to map every couple transitions $s_1^\sharp \xrightarrow{a_1} s_2^\sharp \in \Delta_1$ and $q_1^\sharp \xrightarrow{a_2} q_2^\sharp \in \Delta_2$ into $(s_1^\sharp, q_1^\sharp) \xrightarrow{a_1 \sqcup_{\mathfrak{A}^\sharp} a_2} (s_2^\sharp, q_2^\sharp)$. While being sound, this approximation is too coarse and we can gain in precision by separating singular transitions where we can guarantee that both automata cannot move simultaneously. Note that detecting singular transitions is not always possible since the abstract alphabet domain $\mathfrak{A}^\sharp$ may lack a complement operator necessary to extract them precisely. Nevertheless, we propose a heuristic that can detect singularity in a number of situations, while always preserving soundness.

Example 6 The intuition behind the heuristic is depicted in Fig. 11. In the first case (a), singularity between the input transitions $s_1^\sharp \xrightarrow{ab^*} s_2^\sharp$ and $q_1^\sharp \xrightarrow{a^*b} q_2^\sharp$ cannot be decided because the intersection $ab^* \sqcap_{\mathfrak{A}^\sharp} a^*b$ is nonempty. Consequently, both transitions are combined into a single over-approximated product transition that accepts the merged alphabet symbol $ab^* + a^*b$. However, in the second case (b), no

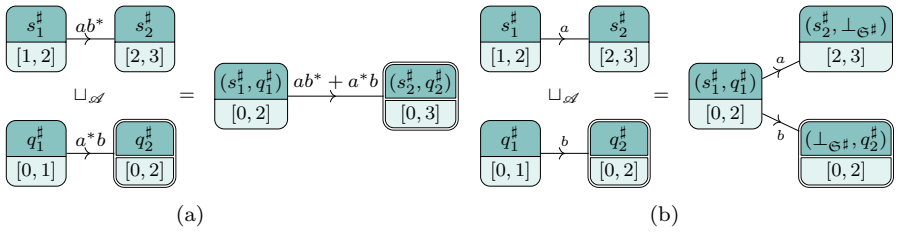


Fig. 11: Cases of constructing a product transition.

intersection exists between the transitions $s_1^\sharp \xrightarrow{a} s_2^\sharp$ and $q_1^\sharp \xrightarrow{b} q_2^\sharp$. This means that both input automata – when in states $s_1^\sharp$ and $q_1^\sharp$ respectively – cannot perform a simultaneous transition, which is expressed as two singular transitions from $(s_1^\sharp, q_1^\sharp)$ to $(s_2^\sharp, \bot_{\mathfrak{S}^\sharp})$ and $(\bot_{\mathfrak{S}^\sharp}, q_2^\sharp)$. ◇

Note that comparing alphabet symbols is not the only means to detect singular transitions. Indeed, in some situations, destination states $s_2^\sharp$ and $q_2^\sharp$ should be kept separated in order to preserve some of *semantic* precision of the analysis. To illustrate this point, let us consider the computation of the goodput of a protocol. In order to obtain a precise quantification of this metric, it is necessary to avoid merging states encapsulating different situations of packet transmission status (reception, loss). To do so, we assume that the abstract states domain $\mathfrak{S}^\sharp$ is provided with some equivalence relation $\equiv_{\mathfrak{S}^\sharp}$ that partitions the states into a finite set of equivalence classes depending on the property of interest. Using this information, we define our product relation as follows:

Definition 10 (Product relation) A binary relation $\mathcal{P}_{A_1}^{A_2} \subseteq \mathfrak{S}^\sharp \times \mathfrak{S}^\sharp$ is a product of A_1 and A_2 *iff* $\forall (s_1^\sharp, q_1^\sharp) \in \mathcal{P}_{A_1}^{A_2}$ we have $s_1^\sharp \equiv_{\mathfrak{S}^\sharp} q_1^\sharp$ and:

$$\begin{cases} (s_2^\sharp, q_2^\sharp) \in \mathcal{P}_{A_1}^{A_2} & \text{if } \exists s_1^\sharp \xrightarrow{a_1^\sharp} s_2^\sharp \in \Delta_1, \exists q_1^\sharp \xrightarrow{a_2^\sharp} q_2^\sharp \in \Delta_2 : \left(s_2^\sharp \equiv_{\mathfrak{S}^\sharp} q_2^\sharp\right) \wedge \left(a_1^\sharp \sqcap_{\mathfrak{A}^\sharp} a_2^\sharp \neq \bot_{\mathfrak{A}^\sharp}\right) \\ (s_2^\sharp, \bot_{\mathfrak{S}^\sharp}) \in \mathcal{P}_{A_1}^{A_2} & \text{if } \exists s_1^\sharp \xrightarrow{a_1^\sharp} s_2^\sharp \in \Delta_1, \forall q_1^\sharp \xrightarrow{a_2^\sharp} q_2^\sharp \in \Delta_2 : \left(s_2^\sharp \not\equiv_{\mathfrak{S}^\sharp} q_2^\sharp\right) \vee \left(a_1^\sharp \sqcap_{\mathfrak{A}^\sharp} a_2^\sharp = \bot_{\mathfrak{A}^\sharp}\right) \\ (\bot_{\mathfrak{S}^\sharp}, q_2^\sharp) \in \mathcal{P}_{A_1}^{A_2} & \text{if } \exists q_1^\sharp \xrightarrow{a_2^\sharp} q_2^\sharp \in \Delta_2, \forall s_1^\sharp \xrightarrow{a_1^\sharp} s_2^\sharp \in \Delta_1 : \left(s_2^\sharp \not\equiv_{\mathfrak{S}^\sharp} q_2^\sharp\right) \vee \left(a_1^\sharp \sqcap_{\mathfrak{A}^\sharp} a_2^\sharp = \bot_{\mathfrak{A}^\sharp}\right) \end{cases}$$

with the convention that $s^\sharp \equiv_{\mathfrak{S}^\sharp} \bot_{\mathfrak{S}^\sharp}, \forall s^\sharp \in \mathfrak{S}^\sharp$.

Definition 11 (Join) Let $\circledast_{A_1}^{A_2}$ denote the smallest product relation containing $(s_0^\sharp, q_0^\sharp)$. We define the structure of the join automaton $(J, j_0^\sharp, \Delta, F) = A_1 \sqcup_{\mathscr{A}} A_2$ as follows:

$$\begin{cases} J & \stackrel{\text{def}}{=} \{\, s^\sharp \sqcup_{\mathfrak{S}^\sharp} q^\sharp \mid (s^\sharp, q^\sharp) \in \circledast_{A_1}^{A_2} \,\} \\ j_0^\sharp & \stackrel{\text{def}}{=} s_0^\sharp \sqcup_{\mathfrak{S}^\sharp} q_0^\sharp \\ \Delta & \stackrel{\text{def}}{=} \{\, j_1^\sharp \xrightarrow{a^\sharp} j_2^\sharp \mid \\ & \qquad \exists (s_1^\sharp, q_1^\sharp) \in \circledast_{A_1}^{A_2}, (s_2^\sharp, q_2^\sharp) \in \circledast_{A_1}^{A_2} : \\ & \qquad \exists s_1^\sharp \xrightarrow{a_1^\sharp} s_2^\sharp \in \Delta_1, q_1^\sharp \xrightarrow{a_2^\sharp} q_2^\sharp \in \Delta_2 : \\ & \qquad j_1^\sharp = s_1^\sharp \sqcup_{\mathfrak{S}^\sharp} q_1^\sharp \wedge a^\sharp = a_1^\sharp \sqcup_{\mathfrak{A}} a_2^\sharp \wedge j_2^\sharp = s_2^\sharp \sqcup_{\mathfrak{S}^\sharp} q_2^\sharp \,\} \\ F & \stackrel{\text{def}}{=} \{\, s^\sharp \sqcup_{\mathfrak{S}^\sharp} q^\sharp \mid (s^\sharp, q^\sharp) \in \circledast_{A_1}^{A_2} \wedge (s^\sharp \in F_1 \vee q^\sharp \in F_2) \,\} \end{cases}$$

In other words, we simply map each product state $(s^\sharp, q^\sharp) \in \divideontimes_{A_1}^{A_2}$ to $s^\sharp \sqcup_{\mathfrak{S}^\sharp} q^\sharp$. The final states are the subsets of these images where at least $s^\sharp$ or $q^\sharp$ is final.

3.1.3 Append

We introduce also the append operator $\oplus_\phi \in \mathscr{A}(\mathfrak{A}^\sharp, \mathfrak{S}^\sharp) \times \mathfrak{A}^\sharp \to \mathscr{A}(\mathfrak{A}^\sharp, \mathfrak{S}^\sharp)$ that extends a given abstract automaton with a set of new outgoing transitions. In addition to the abstract alphabet symbol that will decorate the new transitions, the append operator requires an additional parameter $\phi \in \mathfrak{S}^\sharp \to \mathfrak{S}^\sharp$ that encodes the effect of the transition at the state level. By doing so, the functor $\mathscr{A}$ externalizes the definition of the semantics of transitions, which is left to the instantiating domain.

Formally, we define the append operator as follows:

$$
\begin{aligned}
&A \oplus_\phi a^\sharp \stackrel{\text{def}}{=} \\
&\quad \text{let } F' = \{\, \phi(s^\sharp) \mid s^\sharp \in F \,\} \text{ in} \\
&\quad (S \cup F', s_0^\sharp, F', \Delta \cup \{\, s^\sharp \xrightarrow{a^\sharp} \phi(s^\sharp) \mid s^\sharp \in F \,\})
\end{aligned}
$$

which means that from every final state $s^\sharp \in F$ of A, a new edge is created, labeled with with $a^\sharp$, that leads to a new final state computed as the image of $s^\sharp$ through the parameter transfer function ϕ that annotates the operator $\oplus_\phi$.

3.1.4 Widening

Finally, we present a widening operator to avoid growing an automaton indefinitely during loop iterations. The original lattice automata domain [31] proposed a widening operator, inspired from [49,18], that employs a bisimulation-based minimization to merge similar states by comparing their transitions at some given depth. However, it assumes that the abstract alphabet domain is provided with an equivalence relation that partitions the symbols into a finite set of equivalence classes. We believe that it is more meaningful to perform this partitioning on the abstract states as explained earlier for the computation of the product relation. Therefore, we employ a different approach inspired from graph widening [48,32,50]. We compare the result of successive loop iterations and we try to detect the increment transitions to extrapolate them by creating cycles. However, existing graph widenings are limited to finite alphabets and may not ensure the convergence on ascending chains, so we propose an extension to alleviate these shortcomings.

The proposed algorithm is executed in two phases. Firstly, we perform a *structural widening* to extrapolate the language recognized by the input automata and we ignore for the moment the abstract states. We show in Fig. 12 the main steps of this widening. Assume that A_1 and A_2 are the results of two successive iterations. Without loss of generality, we assume that $A_1 \sqsubseteq_{\mathscr{A}} A_2$ (if this is not the case, we replace A_2 by $A_1 \sqcup_{\mathscr{A}} A_2$). First, we compare A_1 and A_2 in order to extract the increment transitions using the following function:

$$
\begin{aligned}
\texttt{incr}(A_1, A_2) \stackrel{\text{def}}{=} \{\, (s_1^\sharp, q_1^\sharp \xrightarrow{a_2^\sharp} q_2^\sharp) \mid &\left((s_1^\sharp, q_1^\sharp) \in \divideontimes_{A_1}^{A_2}\right) \wedge \left(s_1^\sharp, q_1^\sharp \neq \bot_{\mathfrak{S}^\sharp}\right) \wedge \\
&\exists q_1^\sharp \xrightarrow{a_2^\sharp} q_2^\sharp \in \Delta_2, \forall s_1^\sharp \xrightarrow{a_1^\sharp} s_2^\sharp \in \Delta_1 : \left(s_2^\sharp \not\equiv_{\mathfrak{S}^\sharp} q_2^\sharp\right) \vee \left(a_2^\sharp \not\sqsubseteq_{\mathfrak{A}^\sharp} a_1^\sharp\right) \,\}
\end{aligned}
$$

Input : Two abstract automata A_1 and A_2
Output : Widened abstract automaton A

$A : (S, s_0^\sharp, F, \Delta) \leftarrow A_1$;
⟅ Find the increment transitions ⟆
$\delta \leftarrow \texttt{incr}(A, A_2)$;
while $\delta \neq \varnothing$ **do**
  ⟅ Select one increment transition ⟆
  $(s_1^\sharp, q_1^\sharp \xrightarrow{a_2^\sharp} q_2^\sharp) \leftarrow \texttt{head}(\delta)$;
  ⟅ $q_2^\sharp$ is the default endpoint candidate for the missing transition ⟆
  $s_\equiv^\sharp \leftarrow q_2^\sharp$;
  ⟅ Search for better candidates in A ⟆
  $S_\equiv \leftarrow \{\, s^\sharp \in S \mid s^\sharp \equiv_{\mathfrak{S}^\sharp} q_2^\sharp \,\}$;
  if $S_\equiv \neq \varnothing$ **then**
    $s_\equiv \leftarrow (\texttt{head} \circ \texttt{sort})(\mathcal{I}^{A,A_2}_{q_2^\sharp}, S_\equiv)$;
  end
  $S' \leftarrow S \cup \{\, s_\equiv^\sharp \,\}$;
  ⟅ Add $s_\equiv^\sharp$ to finals if necessary ⟆
  $F' \leftarrow F \cup \left((s_\equiv^\sharp \in F_2)?\{\, s_\equiv^\sharp \,\} : \varnothing\right)$;
  ⟅ Add the missing transition ⟆
  $\Delta' \leftarrow \Delta \cup$
    $\{\, s_1^\sharp \xrightarrow{a^\sharp} s_\equiv^\sharp \mid s_1^\sharp \xrightarrow{a_1^\sharp} s_\equiv^\sharp \in \Delta \wedge a^\sharp = a_1^\sharp \triangledown_{\mathfrak{A}^\sharp} a_2^\sharp \,\} \cup$
    $\{\, s_1^\sharp \xrightarrow{a_2^\sharp} s_\equiv^\sharp \mid \forall a_1^\sharp : s_1^\sharp \xrightarrow{a_1^\sharp} s_\equiv^\sharp \notin \Delta \,\}$;
  $A \leftarrow (S', s_0^\sharp, F', \Delta')$;
  ⟅ Find new increment transitions ⟆
  $\delta \leftarrow \texttt{incr}(A, A_2)$;
end

Fig. 12: Structural widening algorithm for abstract automata.

Essentially, an increment $(s_1^\sharp, q_1^\sharp \xrightarrow{a^\sharp} q_2^\sharp)$ means that A_1 at state $s_1^\sharp$ cannot recognize the symbol $a^\sharp$ while A_2 recognizes it through a transition from $q_1^\sharp$ to $q_2^\sharp$.

Now, we need to extrapolate A_1 in order to recover this difference, which is done by adding the missing word suffix $a_2^\sharp$ while trying not to grow A_1 in size. The basic idea is to sort states in A_1 depending on how they compare to the missing state $q_2^\sharp$. The comparison is performed with the following *similarity index* expressing the proportion of common partial traces that a state shares with $q_2^\sharp$:

$$\mathcal{I}^{A_1,A_2}_{q_2^\sharp}(s^\sharp) \overset{\text{def}}{=} \left\| \{\, a_1^\sharp \dots a_n^\sharp \in \overrightarrow{\mathbf{L}}_{A_2,k}\,(q_2^\sharp) \mid \exists b_1^\sharp \dots b_n^\sharp \in \overrightarrow{\mathbf{L}}_{A_1,k}\,(s^\sharp), \forall i : a_i^\sharp \sqsubseteq_{\mathfrak{A}^\sharp} b_i^\sharp \,\} \right\| + \left\| \{\, a_1^\sharp \dots a_n^\sharp \in \overleftarrow{\mathbf{L}}_{A_2,k}\,(q_2^\sharp) \mid \exists b_1^\sharp \dots b_n^\sharp \in \overleftarrow{\mathbf{L}}_{A_1,k}\,(s^\sharp), \forall i : a_i^\sharp \sqsubseteq_{\mathfrak{A}^\sharp} b_i^\sharp \,\} \right\|$$

where $\overrightarrow{\mathbf{L}}_{A,k}\,(s^\sharp)$ (resp. $\overleftarrow{\mathbf{L}}_{A,k}\,(s^\sharp)$) is the set of words starting from (resp. ending at) $s^\sharp$ of length less than k, where k is a parameter of the analysis. In other words, these two utility functions denote respectively the set of reachable and co-reachable words of a given state at some depth k.

After selecting the state $s_\equiv^\sharp$ with the highest similarity index, we add the missing transitions after widening the alphabet symbol if a transition already exists in

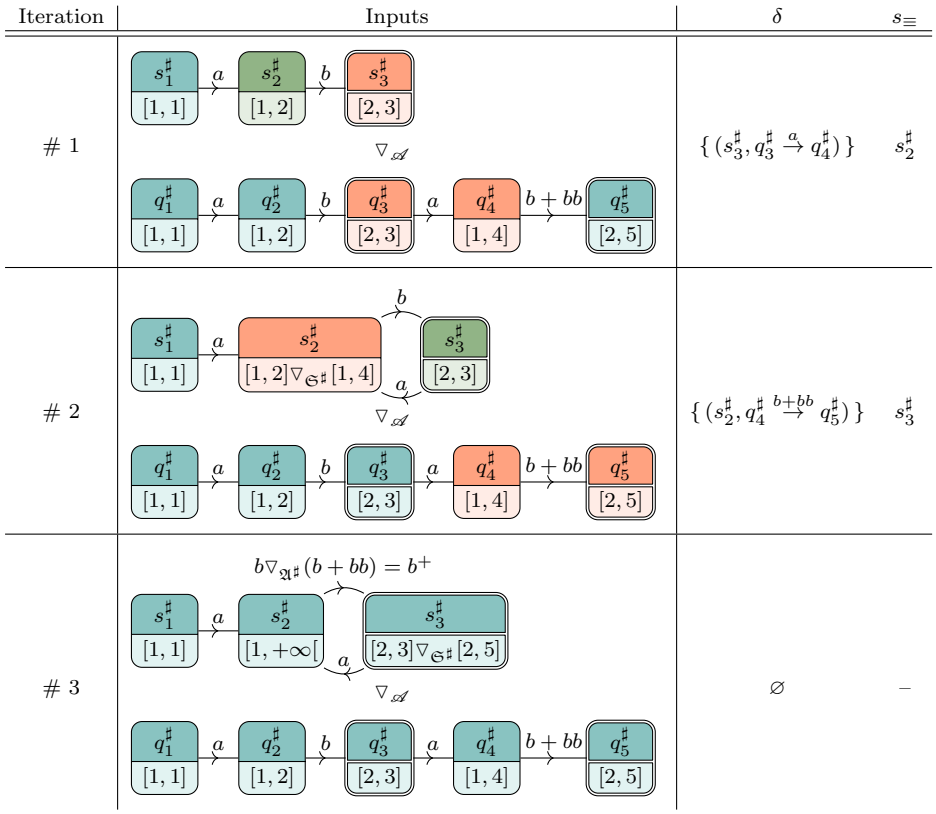


Fig. 13: Example of abstract automata widening.

A. By iterating over all increment transitions, we obtain an automata structure that does not grow indefinitely since we add new states only if no existing one is equivalent. By assuming that the number of equivalence classes of $\equiv_{\mathfrak{S}^\sharp}$ is finite, the widening ensures termination.

After the structural widening, we inspect the states of the resulting automaton to extrapolate them if necessary. We simply compute the simulation relation $\preccurlyeq^A_{A_2}$ between A_2 and the widened automaton A, and we replace every state $s^\sharp \in S$ with $s^\sharp \triangledown_{\mathfrak{S}^\sharp} (s_1^\sharp \sqcup_{\mathfrak{S}^\sharp} s_2^\sharp \sqcup_{\mathfrak{S}^\sharp} \dots)$ where $s_i^\sharp \preccurlyeq^A_{A_2} s^\sharp, \forall i$.

Example 7 The different steps of the proposed widening algorithm are illustrated in Fig. 13 in which we consider two example automata A_1 and A_2, where $A_1 \sqsubseteq_{\mathscr{A}} A_2$. Let us assume that the similarity depth parameter is $k = 1$. During the first iteration, the algorithm detects the increment transition $(s_3^\sharp, q_3^\sharp \xrightarrow{a} q_4^\sharp)$. By comparing the reachable and co-reachable k-words of the states of A_1 to those of $q_4^\sharp$, we obtain the following similarity indices:

$$\mathcal{I}^{A_1,A_2}_{q_4^\sharp}(s_1^\sharp) = 0 \qquad \mathcal{I}^{A_1,A_2}_{q_4^\sharp}(s_2^\sharp) = 2 \qquad \mathcal{I}^{A_1,A_2}_{q_4^\sharp}(s_3^\sharp) = 0$$

Therefore, $s_2^\sharp$ is selected as the most similar state to $q_4^\sharp$ and the missing transition $s_3^\sharp \xrightarrow{a} s_2^\sharp$ is added. Note that we can combine structural and state widening at this point, which allows as to over-approximate $s_2^\sharp$ by $s_2^\sharp \nabla_{\mathfrak{S}^\sharp} (s_2^\sharp \sqcup_{\mathfrak{S}^\sharp} q_4^\sharp)$ to accelerate convergence.

During the second iteration, we use the resulting automaton as the left argument of the widening operator and we iterate the same process. The algorithm selects $(s_2^\sharp, q_4^\sharp \xrightarrow{b+bb} q_5^\sharp)$ as the increment transition and computes the following similarity indices:

$$\mathcal{I}^{A,A_2}_{q_5^\sharp}(s_1^\sharp) = 0 \qquad \mathcal{I}^{A,A_2}_{q_5^\sharp}(s_2^\sharp) = 0 \qquad \mathcal{I}^{A,A_2}_{q_5^\sharp}(s_3^\sharp) = 1$$

The state $s_3^\sharp$ being the most comparable one to $q_5^\sharp$, the automaton A is enriched with the transition $s_2^\sharp \xrightarrow{b+bb} s_3^\sharp$. Since a transition $s_2^\sharp \xrightarrow{b} s_3^\sharp$ already exists, we just need to compute the widening of its alphabets $b \nabla_{\mathfrak{A}^\sharp} (b + bb)$ and endpoint state $s_3^\sharp \nabla_{\mathfrak{S}^\sharp} q_5^\sharp$. By doing so, no increment transition can be found, which means that no more widening iterations are required. ◇

3.2 Abstract Scenarios

Using the functor domain $\mathscr{A}$, we instantiate an abstract scenario domain for approximating words of random events. Two considerations are important to take into account. First, the length of these words may depend on some variables of the program. It is clear that ignoring these relations may lead to imprecise computations of the stationary distribution. Consequently, we enrich the domain with an abstract Parikh vector [41] to count the number of occurrences of random events within accepted words. By using a relational numerical domain, such as octagons [35] or polyhedra [10], we preserve some relationships between the number of events and program variables.

The second consideration is related to the uniform distribution. As shown previously in the concrete evaluation function of Fig. 7, the number of outcomes depends on the bounds provided as argument to the function `uniform`. Since these arguments are evaluated in the running environment, we can have an infinite number of outcomes at a given control location when considering all possible executions.

We perform a simplifying abstraction of the random events Ξ in order to obtain a finite size alphabet and avoid the explosion of the uniform distribution outcomes. Assume that we are analyzing the statement $x = \texttt{uniform}_l(e_1, e_2)$ in abstract environment $\rho^\sharp$. Several abstractions are possible. In this work, we choose to partition the outcomes into a fixed number U of abstract outcomes, where U is a parameter of the analysis. The first $U-1$ partitions represent the concrete individual outcomes $\{\min(e_1 + i - 1, e_2) \mid i \in [1, U-1]\}$, to which we associate the abstract events $\{\, \mathbf{u}_l^i \mid i \in [1, U-1] \,\}$. For the remaining outcomes, we merge them into a single aggregate abstract event $\mathbf{u}_l^{\text{☆}}$.

Formally, we obtain a simple finite set of abstract events $\Xi^\sharp \stackrel{\text{def}}{=} \{\, \mathbf{b}_l, \overline{\mathbf{b}}_l \mid l \in \mathcal{L}_{\downarrow \texttt{bernoulli}} \,\} \cup \{\, \mathbf{u}_l^i, \mathbf{u}_l^{\text{☆}} \mid l \in \mathcal{L}_{\downarrow \texttt{uniform}} \wedge 1 \leq i \leq U-1 \,\}$. For the Parikh vector, we associate to every abstract event $\xi^\sharp \in \Xi^\sharp$ a counter variable $\kappa_{\xi^\sharp} \in \mathbb{N}$ that will be incremented whenever the event $\xi^\sharp$ occurs.

$$\mathbb{S}^\sharp_\Omega[\![\, x = \texttt{bernoulli}_l() \,]\!]\omega^\sharp \stackrel{\text{def}}{=}$$
$$\text{let } \phi_{\texttt{TRUE}} = \lambda(-,\rho^\sharp).\,(l, \mathbb{S}^\sharp_{\mathfrak{N}}[\![\, \kappa_{\mathbf{b}_l} = \kappa_{\mathbf{b}_l} + 1 \,]\!] \circ \mathbb{S}^\sharp_{\mathfrak{N}^\sharp}[\![\, x = \texttt{TRUE} \,]\!]\rho^\sharp) \text{ in}$$
$$\text{let } \phi_{\texttt{FALSE}} = \lambda(-,\rho^\sharp).\,(l, \mathbb{S}^\sharp_{\mathfrak{N}}[\![\, \kappa_{\overline{\mathbf{b}}_l} = \kappa_{\overline{\mathbf{b}}_l} + 1 \,]\!] \circ \mathbb{S}^\sharp_{\mathfrak{N}^\sharp}[\![\, x = \texttt{FALSE} \,]\!]\rho^\sharp) \text{ in}$$
$$(\omega^\sharp \oplus_{\phi_{\texttt{TRUE}}} \{\, \mathbf{b}_l \,\}) \sqcup_{\mathscr{A}} \left(\omega^\sharp \oplus_{\phi_{\texttt{FALSE}}} \{\, \overline{\mathbf{b}}_l \,\}\right)$$

$$\mathbb{S}^\sharp_\Omega[\![\, x = \texttt{uniform}_l(e_1, e_2) \,]\!]\omega^\sharp \stackrel{\text{def}}{=}$$
$$\text{let } \phi_i = \lambda(-,\rho^\sharp).\,(l, \mathbb{S}^\sharp_{\mathfrak{N}}[\![\, \kappa_{\mathbf{u}^i_l} = \kappa_{\mathbf{u}^i_l} + 1 \,]\!] \circ \mathit{filter}^\sharp_{\mathfrak{N}}(x \le e_2) \circ \mathbb{S}^\sharp_{\mathfrak{N}}[\![\, x = e_1 + i - 1 \,]\!]\rho^\sharp) \text{ in}$$
$$\text{let } \phi_{\bigstar} = \lambda(-,\rho^\sharp).\,(l, \mathbb{S}^\sharp_{\mathfrak{N}}[\![\, \kappa_{\mathbf{u}^{\bigstar}_l} = \kappa_{\mathbf{u}^{\bigstar}_l} + 1 \,]\!] \circ \mathit{filter}^\sharp_{\mathfrak{N}}(e_1 + U \le x \le e_2) \circ \mathbb{S}^\sharp_{\mathfrak{N}}[\![\, x = \top \,]\!]\rho^\sharp) \text{ in}$$
$$\left(\bigsqcup_{1 \le i \le U-1} \omega^\sharp \oplus_{\phi_i} \{\, \mathbf{u}^i_l \,\}\right) \sqcup_{\mathscr{A}} (\omega^\sharp \oplus_{\phi_{\bigstar}} \{\, \mathbf{u}^{\bigstar}_l \,\})$$

$$\mathbb{S}^\sharp[\![\, x = \texttt{wait}_l\ e \,]\!](\tau^\sharp, f_{\Omega^\sharp}) \stackrel{\text{def}}{=}$$
$$\text{let } \phi = \lambda(-,\rho^\sharp).\,(l, \mathbb{S}^\sharp_{\mathfrak{N}}[\![\, \nu = e \,]\!]\rho^\sharp) \text{ in}$$
$$\left(\bigsqcup_{s^\sharp \in F(\tau^\sharp)} (\tau^\sharp \downarrow s^\sharp) \oplus_\phi f_{\Omega^\sharp}(s^\sharp), \lambda s^\sharp.\,\varepsilon^\sharp\right)$$

Fig. 14: Some abstract transfer functions.

Therefore, we define the domain of abstract scenarios as $\Omega^\sharp \stackrel{\text{def}}{=} \mathscr{A}(\wp(\Xi^\sharp), \Sigma^\sharp)$, where $\Sigma^\sharp$ is our previous mapping $\mathcal{L} \to \mathfrak{N}^\sharp$ from program locations to a stock numeric abstract domain $\mathfrak{N}^\sharp$. We assume that $\mathfrak{N}^\sharp$ has a concretization function $\gamma_{\mathfrak{N}^\sharp} \in \mathfrak{N}^\sharp \to \wp(\mathcal{E})$, transfer functions $\mathbb{S}^\sharp_{\mathfrak{N}}[\![\,.\,]\!] \in \mathfrak{N}^\sharp \to \mathfrak{N}^\sharp$ of numeric statements and a filtering function $\mathit{filter}^\sharp_{\mathfrak{N}}(e) \in \mathfrak{N}^\sharp \to \mathfrak{N}^\sharp$ that keeps numeric environments that satisfies the condition expression e.

Let us now describe transfer functions $\mathbb{S}^\sharp_\Omega[\![\,.\,]\!] \in \Omega^\sharp \to \Omega^\sharp$ shown in Fig. 14 formalizing the impact of probability distributions on an abstract scenario. To over-approximate the effect of an assignment $x = \texttt{bernoulli}_l()$ on an abstract scenario $\omega^\sharp$, we process each possible outcome of the distribution separately. Let us illustrate with the case of b_l. We extend the input abstract automaton $\omega^\sharp$ with a new outgoing transition annotated with the state transfer function $\phi_{\texttt{TRUE}}$ that computes the new final states of $\omega^\sharp$. In each numeric environment $\rho^\sharp$ of the current final states, $\phi_{\texttt{TRUE}}$ sets variable x to value `TRUE` and increments the Parikh counter κ_{b_l} associated to the outcome b_l. The same process is applied for the outcome $\overline{b}_l$. The final result is obtained by joining the obtained pair of abstract automata.

Let us now consider the assignment $x = \texttt{uniform}_l(e_1, e_2)$. For the case of an outcome $\mathbf{u}^i_l$, we update the variable x with the evaluation of $\min(e_1 + i - 1, e_2)$, which is done by first performing the assignment $x = e_1 + i - 1$ in the environment of a final state of $\omega^\sharp$, and then apply the filter $x \le e_2$ on the result. The Parikh counter $\kappa_{\mathbf{u}^i_l}$ is also incremented. The aggregate outcome $\mathbf{u}^{\bigstar}_l$ is handled by assigning to x the evaluation of the interval $[e_1 + U, e_2]$ and incrementing the counter $\kappa_{\mathbf{u}^{\bigstar}_l}$.

3.3 Abstract Markov Chains

To provide a computable abstraction of the concrete semantics domain $\mathcal{D} = \wp(\mathcal{T}^\Sigma_\Omega \times \mathcal{E} \times \Omega)$ we proceed in two steps. We start by abstracting the set of observ-

able traces $\wp(\mathcal{T}_{\Omega}^{\Sigma})$ with an abstract automaton $\mathcal{T}^\sharp \stackrel{\text{def}}{=} \mathscr{A}(\Omega^\sharp, \Sigma^\sharp)$ defined over the alphabet of abstract scenarios $\Omega^\sharp$ and the abstract state space $\Sigma^\sharp$. To approximate the partial scenarios starting from the last `wait` statement, we may use the domain of abstract scenarios $\Omega^\sharp$. Since the states of an abstract scenario already embed an abstraction of the program environments $\mathcal{E}$, the product $\mathcal{T}^\sharp \times \Omega^\sharp$ constitutes an abstraction of $\mathcal{D}$ having the following concretization function:

$$\gamma(\tau^\sharp, \omega^\sharp) \stackrel{\text{def}}{=} \{\, (\tau, \rho, \omega) \mid \tau \in \gamma^{\mathbf{T}}_{\mathscr{A}}(\tau^\sharp) \wedge \exists (l_1, \rho_1^\sharp) \xrightarrow{\xi_1^\sharp} \dots \xrightarrow{\xi_{n-1}^\sharp} (l_n, \rho_n^\sharp) \in \mathbf{T}(\omega^\sharp) : \\ \rho \in \gamma_{\mathfrak{N}^\sharp}(\rho_n^\sharp) \wedge \omega \in \gamma^{\mathbf{L}}_{\mathscr{A}}(\xi_1^\sharp \dots \xi_{n-1}^\sharp) \,\}$$

This abstraction is sound but may lead to coarse results. Indeed, by choosing a product abstraction $\mathcal{T}^\sharp \times \Omega^\sharp$, we decouple environments from traces: all reachable environments are merged together regardless of the taken trace. We can enhance the precision of the analysis by maintaining some relationships through partitioning: we simply separate environments depending on the final states of the observable traces automaton $\mathcal{T}^\sharp$. More formally, we define our abstract semantics domain as follows:

$$\mathcal{D}^\sharp \stackrel{\text{def}}{=} \mathcal{T}^\sharp \times (\Sigma^\sharp \to \Omega^\sharp)$$

Let $\mathcal{F} \in \mathscr{A}(\mathfrak{A}^\sharp, \mathfrak{S}^\sharp) \to \wp(\mathfrak{S}^\sharp)$ be a function returning the set of final states of an abstract automaton and $A \downarrow s^\sharp$ the projection of an abstract automaton A on a final state $s^\sharp$ (obtained by restricting the final states of A to the singleton $\{s^\sharp\}$). The concretization function of $\mathcal{D}^\sharp$ is given by:

$$\gamma(\tau^\sharp, f_{\Omega^\sharp}) \stackrel{\text{def}}{=} \{\, (\tau, \rho, \omega) \mid \exists s^\sharp \in \mathcal{F}(\tau^\sharp) : \exists (l_1, \rho_1^\sharp) \xrightarrow{\xi_1^\sharp} \dots \xrightarrow{\xi_{n-1}^\sharp} (l_n, \rho_n^\sharp) \in \mathbf{T}(f_{\Omega^\sharp}(s^\sharp)) : \\ \tau \in \gamma^{\mathbf{T}}_{\mathscr{A}}(\tau^\sharp \downarrow s^\sharp) \wedge \rho \in \gamma_{\mathfrak{N}^\sharp}(\rho_n^\sharp) \wedge \omega \in \gamma^{\mathbf{L}}_{\mathscr{A}}(\xi_1^\sharp \dots \xi_{n-1}^\sharp) \,\}$$

For the abstract transfer functions $\mathbb{S}^\sharp[\![\,.\,]\!] \in \mathcal{D}^\sharp \to \mathcal{D}^\sharp$ of $\mathcal{D}^\sharp$, we focus on the case $\texttt{wait}_l\ e$ since it is the only one that modifies the structure of the abstract Markov chain. The definition is shown in Fig. 14. Given a current abstract element $(\tau^\sharp, f_{\Omega^\sharp}) \in \mathcal{D}^\sharp$, the function iterates over every observable final state $s^\sharp \in \mathcal{F}(\tau^\sharp)$ to add a new transition labeled with the associated partial scenario $f_{\Omega^\sharp}(s^\sharp)$ that points to a new observable state with a sojourn time equal to the current evaluation of e. Finally, since the chain is in a new time tick, all entries of the output scenario map are set to $\varepsilon^\sharp$ representing the empty scenario word where all Parikh counters are reset to 0.

We can show that the following soundness condition is preserved:

Theorem 1 (Soundness)

$$(\mathbb{S}[\![\, s \,]\!] \circ \gamma)\,(\tau^\sharp, \omega^\sharp) \subseteq \Big(\gamma \circ \mathbb{S}^\sharp[\![\, s \,]\!]\Big)\,(\tau^\sharp, \omega^\sharp), \forall s \in stmt, \forall (\tau^\sharp, \omega^\sharp) \in \mathcal{D}^\sharp$$

Example 8 Let us go back to our first motivating example in order to illustrate the inference process of the resulting abstract Markov chain shown in Fig. 2b. The intermediate abstract chains of the most important steps are depicted in Fig. 15.

The `sense` statement at line 4 generates a first abstract chain with a single state representing a one-tick time duration for retrieving data from the sensing device. After that, two new transitions are added to over-approximate the unbounded number of outcomes of the `uniform` distribution at line 6. The abstract event $\mathbf{u}_6^1$ represent the case of choosing a backoff window of length 1, and the

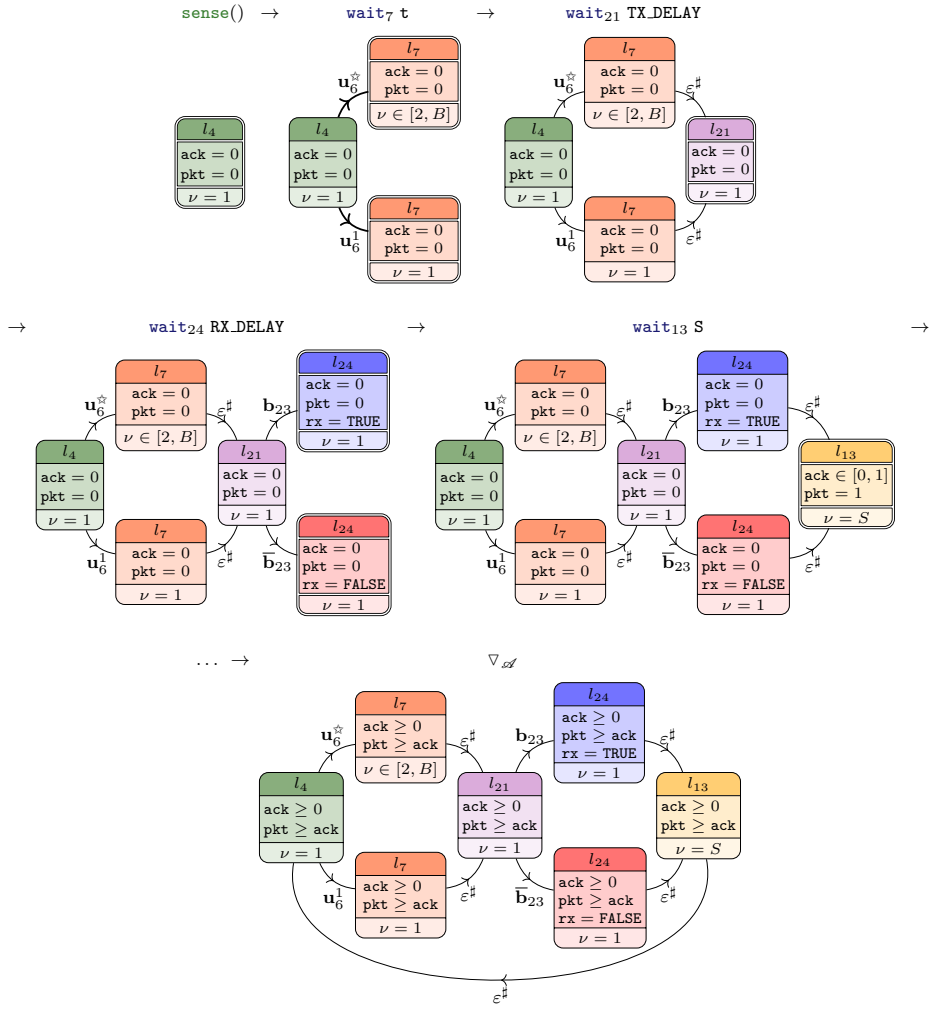


Fig. 15: Analysis iterations of the motivating example.

abstract event $\mathbf{u}_6^{\star}$ over-approximates the remaining cases of lengths in $[2, B]$. Note that it is important to employ relational numerical domains, such as octagons or polyhedra, in order to represent such conditions. Nevertheless, an analysis using a non-relational domains, such as intervals, is still sound but less precise.

When the backoff mechanism terminates, the packet is sent by calling the function `unicast`. The transmission step is translated by our domain as a transition to a one-tick state, since we defined `TX_DELAY` $= 1$. The transition is annotated with the empty abstract scenario $\varepsilon^\sharp$ because no event occurred since the last time tick at line 7. The Bernoulli model of packet loss is represented as two transitions with the events $\mathbf{b}_{23}$ and $\overline{\mathbf{b}}_{23}$ generated by the statement `bernoulli`() at line 23. These

transitions point to two new abstract states with a one-tick sojourn time modeling the time consumed by the reception operation at line 24.

After the `unicast` function returns, the energy saving sleep statement at line 13 is represented by a single state with a sojourn time $\nu = S$. As we iterate the `while` again, the size of the abstract automaton grows but after applying the widening operator $\triangledown_{\mathscr{A}}$ a fixed point is reached that over-approximates the family of concrete Markov chains shown in Fig. 2a. ◇

4 Stationary Distributions

In this section, we present a method for extracting safe bounds of the stationary distribution of an abstract Markov chain. We do so by deriving a *distribution invariant* that establishes a system of parametric linear inequalities over the abstract states. Using the Fourier-Motzkin elimination algorithm, we can find guaranteed bounds of time proportion spent in a given abstract state.

4.1 Distribution Invariants

We begin with some preliminary definitions. For each statement $\texttt{uniform}_l(e_1, e_2)$, we denote by $\overleftarrow{\mathrm{u}}_l \stackrel{\text{def}}{=} e_1$ and $\overrightarrow{\mathrm{u}}_l \stackrel{\text{def}}{=} e_2$ the bounds expressions of the distribution. Also, we define the functions $\min^\sharp [\![e]\!], \max^\sharp [\![e]\!] \in \Sigma^\sharp \to \mathit{expr}$ giving respectively the evaluation of the maximal and minimal values of an expression e in a given abstract state, which is generally provided for free by the underlying numerical domain. In the case of relational domains, the returned bounds can be symbolic. For the sake of simplicity, we write also $\min^\sharp_\star [\![e]\!], \max^\sharp_\star [\![e]\!] \in \mathit{expr}$ to denote respectively the minimal and maximal evaluations over the set of all reachable abstract states. The abstract Markov chain obtained after the analysis of P is given by:

$$\mathbb{M}^\sharp [\![\mathsf{P}]\!] R^\sharp \stackrel{\text{def}}{=} \lambda R^\sharp . \text{ let } (\tau^\sharp, -) = \mathbb{S}^\sharp [\![\mathsf{P}]\!] R^\sharp \text{ in } \tau^\sharp$$

The following definition gives a means to compute the probability of given abstract scenario:

Definition 12 (Abstract scenario probability) The symbolic probability $\Pr^\sharp \left[\!\!\left[\omega^\sharp \right]\!\!\right] \in \mathit{expr}$ of abstract scenarios $\omega^\sharp \in \Omega^\sharp$ is defined by structural induction on its regular expression:

$$\Pr^\sharp \left[\!\!\left[\omega^\sharp \right]\!\!\right] \stackrel{\text{def}}{=} \begin{cases} 1 & \text{if } \omega^\sharp = \varepsilon^\sharp \\ p_l & \text{if } \omega^\sharp = \mathbf{b}_l \\ 1 - p_l & \text{if } \omega^\sharp = \overline{\mathbf{b}}_l \\ \frac{1}{\min^\sharp_\star [\![\overrightarrow{\mathrm{u}}_l]\!] - \max^\sharp_\star [\![\overleftarrow{\mathrm{u}}_l]\!] + 1} & \text{if } \omega^\sharp = \mathbf{u}^i_l \\ \max\left(0, \frac{\max^\sharp_\star [\![\overrightarrow{\mathrm{u}}_l]\!] - \min^\sharp_\star [\![\overleftarrow{\mathrm{u}}_l]\!] + 2 - U}{\min^\sharp_\star [\![\overrightarrow{\mathrm{u}}_l]\!] - \max^\sharp_\star [\![\overleftarrow{\mathrm{u}}_l]\!] + 1}\right) & \text{if } \omega^\sharp = \mathbf{u}^\star_l \\ \Pr^\sharp \left[\!\!\left[\omega^\sharp_0 \right]\!\!\right] \times \Pr^\sharp \left[\!\!\left[\xi^\sharp \right]\!\!\right] & \text{if } \omega^\sharp = \omega^\sharp_0 \xi^\sharp \\ \Pr^\sharp \left[\!\!\left[\omega^\sharp_1 \right]\!\!\right] + \Pr^\sharp \left[\!\!\left[\omega^\sharp_2 \right]\!\!\right] & \text{if } \omega^\sharp = \omega^\sharp_1 + \omega^\sharp_2 \end{cases}$$

By combining the sojourn and probability invariants embedded in the abstract chain, we construct an abstract transition matrix that characterizes completely the stochastic properties of the program inside one finite data structure:

Definition 13 (Abstract transition matrix) The abstract transition matrix $\mathbb{P}^\sharp [\![\mathsf{P}]\!] \in \mathcal{D}^\sharp \to \Sigma^\sharp \times \Sigma^\sharp \to \mathit{expr}$ is a square symbolic matrix defined as:

$$\mathbb{P}^\sharp [\![\mathsf{P}]\!] \stackrel{\text{def}}{=} \lambda R^\sharp.\, \lambda(s_i^\sharp, s_j^\sharp).\, \frac{\max^\sharp [\![\nu]\!] s_j^\sharp}{\min^\sharp [\![\nu]\!] s_i^\sharp} \sum_{s_i^\sharp \xrightarrow{\omega^\sharp} s_j^\sharp \in \mathbb{M}^\sharp [\![\mathsf{P}]\!] R^\sharp} \mathrm{Pr}^\sharp \left[\!\!\left[\omega^\sharp \right]\!\!\right]$$

Example 9 Let P be our first motivating example shown in Fig. 1a and let $I^\sharp$ be the initial empty abstract state. Let $S = \langle \mathsf{ss}, \mathsf{bk}_1, \mathsf{bk}_\star, \mathsf{tx}, \mathsf{ack}, \overline{\mathsf{ack}}, \mathsf{sl} \rangle$ be the vector of abstract states of the resulting chain shown in Fig. 2b. To obtain the matrix $\mathrm{Pr}^\sharp [\![\mathsf{P}]\!] I^\sharp$, we iterate over all the transitions of the abstract chain. Consider for example the case of the transition $\mathsf{ss} \xrightarrow{\mathbf{u}_5^\star} \mathsf{bk}_\star$. First, we apply Definition 12 to compute the transitions probability $\mathrm{Pr}^\sharp \left[\!\!\left[\mathbf{u}_5^\star \right]\!\!\right] = \frac{B-1}{B}$. Afterwards, we extract the sojourn time bounds $\max^\sharp [\![\nu]\!](\mathsf{bk}_\star) = B$ and $\min^\sharp [\![\nu]\!](\mathsf{ss}) = 1$ from the embedded numeric environments. Finally, we apply Definition (13) to obtain the matrix cell $\mathbb{P}^\sharp [\![\mathsf{P}]\!](I^\sharp)(\mathsf{ss}, \mathsf{bk}_\star) = \frac{B(B-1)}{B} = B - 1$. By iterating the same process for all transitions we obtain:

$$\mathbb{P}^\sharp [\![\mathsf{P}]\!] I^\sharp = \begin{pmatrix} 0 & \frac{1}{B} & B-1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & \frac{1}{2} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & p_{23} & 1-p_{23} & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & S \\ 0 & 0 & 0 & 0 & 0 & 0 & S \\ \frac{1}{S} & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix} \tag{3}$$

◇

Let us now introduce the concept of *abstract stationary distribution* that gives the proportion of time spent in every abstract state. In the sequel, we denote $I^\sharp \in \mathcal{D}^\sharp$ an abstraction of the initial states.

Definition 14 (Abstract stationary distribution) The abstract stationary distribution of an abstract chain $\mathbb{M}^\sharp [\![\mathsf{P}]\!] I^\sharp$ is defined as the symbolic vector $\boldsymbol{\pi}^\sharp [\![\mathsf{P}]\!] I^\sharp \in \Sigma^\sharp \to \mathit{expr}$ verifying:

$$\forall \rho \in \{\, \rho \mid (-, \rho, -) \in \gamma(I^\sharp) \,\}, \forall s^\sharp \in \Sigma^\sharp :$$
$$\mathbb{E}_{\mathcal{E}} \left[\!\!\left[\boldsymbol{\pi}^\sharp [\![\mathsf{P}]\!](I^\sharp)(s^\sharp) \right]\!\!\right] \rho = \sum_{s \in \gamma_\Sigma(s^\sharp)} \boldsymbol{\pi} [\![\mathsf{P}]\!](\rho)(s)$$

where $\mathbb{E}_{\mathcal{E}} [\![\,.\,]\!] \in \mathcal{E} \to \mathcal{V}$ is the classic evaluation of numeric expressions.

It is important to note that since spurious concrete states $s \in \gamma_\Sigma(s^\sharp)$ have a null concrete stationary probability $\boldsymbol{\pi} [\![\mathsf{P}]\!](\rho)(s)$, the abstract stationary probability $\boldsymbol{\pi}^\sharp [\![\mathsf{P}]\!](I^\sharp)(s^\sharp)$ represents the exact sum of the stationary probabilities of the real concrete states abstracted by $s^\sharp$. Therefore, any lower and/or upper bounds that can be found about $\boldsymbol{\pi}^\sharp [\![\mathsf{P}]\!](I^\sharp)(s^\sharp)$ are also valid for the concrete states abstracted by $s^\sharp$. To compute such bounds, we use $\mathbb{P}^\sharp [\![\mathsf{P}]\!] I^\sharp$ with the following result:

Theorem 2 (Distribution invariant)

$$\forall \rho \in \{\, \rho \mid (-, \rho, -) \in \gamma(I^\sharp) \,\} :$$
$$\begin{cases} \forall s_i^\sharp \in \Sigma^\sharp : \mathbb{E}_{\mathcal{E}} \left[\!\!\left[\boldsymbol{\pi}^\sharp \, [\![\mathsf{P}]\!] (I^\sharp)(s_i^\sharp) \right]\!\!\right] \rho \leq \sum\limits_{s_j^\sharp \in \Sigma^\sharp} \mathbb{E}_{\mathcal{E}} \left[\!\!\left[\boldsymbol{\pi}^\sharp \, [\![\mathsf{P}]\!] (I^\sharp)(s_j^\sharp) \times \mathbb{P}^\sharp \, [\![\mathsf{P}]\!] (I^\sharp)(s_j^\sharp, s_i^\sharp) \right]\!\!\right] \rho \\ \sum\limits_{s^\sharp \in \Sigma^\sharp} \mathbb{E}_{\mathcal{E}} \left[\!\!\left[\boldsymbol{\pi}^\sharp \, [\![\mathsf{P}]\!] (I^\sharp)(s^\sharp) \right]\!\!\right] \rho = 1 \end{cases}$$

Proof See Appendix A.

Simply stated, this theorem allows us to establish two important invariants. The first one is a weak form of the Markov property and can be informally written in vector algebra as $\boldsymbol{\pi}^\sharp \, [\![\, . \,]\!] \leq \boldsymbol{\pi}^\sharp \, [\![\, . \,]\!] \mathbb{P}^\sharp \, [\![\, . \,]\!]$, which means that the probability of being at some abstract state is always upper-bounded by the sum of the probabilities of moving from previous states. The upper-bound is guaranteed by the sound over-approximation of abstract scenarios and sojourn times. The second invariant is the normalization condition that stipulates that the abstract states cover the entire space of concrete states, and therefore, the sum of the probabilities of abstract states in a given parameter valuation should be equal to 1 (since spurious states introduced by concretization have a null concrete probability).

Example 10 By applying Theorem 2 on the abstract stochastic matrix (3), we obtain the following parametric system of linear inequalities:

$$\begin{cases} \pi^\sharp_{\mathsf{ss}} \leq \frac{1}{S} \pi^\sharp_{\mathsf{sl}} \\ \pi^\sharp_{\mathsf{bk}_1} \leq \frac{1}{B} \pi^\sharp_{\mathsf{ss}} \\ \pi^\sharp_{\mathsf{bk}_\star} \leq (B-1) \pi^\sharp_{\mathsf{ss}} \\ \pi^\sharp_{\mathsf{tx}} \leq \pi^\sharp_{\mathsf{bk}_1} + \frac{1}{2} \pi^\sharp_{\mathsf{bk}_\star} \\ \pi^\sharp_{\mathsf{ack}} \leq p \pi^\sharp_{\mathsf{tx}} \\ \pi^\sharp_{\overline{\mathsf{ack}}} \leq (1-p) \pi^\sharp_{\mathsf{tx}} \\ \pi^\sharp_{\mathsf{sl}} \leq S \pi^\sharp_{\mathsf{ack}} + S \pi^\sharp_{\overline{\mathsf{ack}}} \\ \pi^\sharp_{\mathsf{ss}} + \pi^\sharp_{\mathsf{bk}_1} + \pi^\sharp_{\mathsf{bk}_\star} + \pi^\sharp_{\mathsf{tx}} + \pi^\sharp_{\mathsf{ack}} + \pi^\sharp_{\overline{\mathsf{ack}}} + \pi^\sharp_{\mathsf{sl}} = 1 \end{cases}$$

where the vector $\langle \pi^\sharp_{\mathsf{ss}}, \pi^\sharp_{\mathsf{bk}_1}, \pi^\sharp_{\mathsf{bk}_\star}, \pi^\sharp_{\mathsf{tx}}, \pi^\sharp_{\mathsf{ack}}, \pi^\sharp_{\overline{\mathsf{ack}}}, \pi^\sharp_{\mathsf{sl}} \rangle$ is the vector of unknown limiting probabilities. ◇

The obtained system of parametric linear inequalities can be used to find safe bounds of the property of interest. Without loss of generality, assume that the time proportion of this property is associated with the stationary probability of some state $s^\sharp$. To compute a safe symbolic range of $\boldsymbol{\pi}^\sharp \, [\![\mathsf{P}]\!] (I^\sharp)(s^\sharp)$, we just have to perform a projection of the linear system that keeps only $\boldsymbol{\pi}^\sharp \, [\![\mathsf{P}]\!] (I^\sharp)(s^\sharp)$ and removes the other unknowns while preserving all constraints. Many off-the-shelf symbolic environments, such as Sage and Mathematica, can solve such problems symbolically. However, in practice we have experienced very low performances even for small scale inequality systems. To overcome this problem, we have implemented a parametric Fourier-Motzkin projection algorithm [24,47] that performs better than tested symbolic environments.

Input : Inequalities $I_0 = \{c_j + \sum_{1 \le i \le n} a_{i,j} x_i \le 0 \mid 1 \le j \le m\}$
Input : Constraints C_0
Output : Set $\mathcal{I}$ of inequalities-constraints tuples $\langle C, I \rangle$

$\mathcal{I} \leftarrow \{\langle C_0, I_0 \rangle\}$;
for $i = 1$ **to** $n - 1$ **do**
  ⟅ Eliminate unknown x_i ⟆
  $\mathcal{I}' \leftarrow \varnothing$;
  foreach $\langle C, I \rangle \in \mathcal{I}$ **do**
    ⟅ Decompose I depending on the sign of the coefficient of x_i ⟆
    $I^+ \leftarrow \{\langle c_j + \sum_{i \le k \le n} a_{k,j} x_k \le 0 \rangle \in I \mid a_{i,j} > 0\}$;
    $I^- \leftarrow \{\langle c_j + \sum_{i \le k \le n} a_{k,j} x_k \le 0 \rangle \in I \mid a_{i,j} < 0\}$;
    $I^0 \leftarrow \{\langle c_j + \sum_{i \le k \le n} a_{k,j} x_k \le 0 \rangle \in I \mid a_{i,j} = 0\}$;
    $I^? \leftarrow I \setminus (I^+ \cup I^- \cup I^0)$;
    ⟅ For each possible sign combination in $I^?$, we generate a new case ⟆
    $P \leftarrow \{\langle I^+, I^-, I^0, C \rangle\}$;
    foreach $\langle c_j + \sum_{i \le k \le n} a_{k,j} x_k \le 0 \rangle \in I^?$ **do**
      $x \leftarrow \langle c_j + \sum_{i \le k \le n} a_{k,j} x_k \le 0 \rangle$;
      $P \leftarrow \{\langle I^+ \cup \{x\}, I^-, I^0, C \wedge a_{i,j} > 0 \rangle \mid \langle I^+, I^-, I^0, C \rangle \in P \wedge \mathbf{check}(C \wedge a_{i,j} > 0)\} \cup$
        $\{\langle I^+, I^- \cup \{x\}, I^0, C \wedge a_{i,j} < 0 \rangle \mid \langle I^+, I^-, I^0, C \rangle \in P \wedge \mathbf{check}(C \wedge a_{i,j} < 0)\} \cup$
        $\{\langle I^+, I^-, I^0 \cup \{x\}, C \wedge a_{i,j} = 0 \rangle \mid \langle I^+, I^-, I^0, C \rangle \in P \wedge \mathbf{check}(C \wedge a_{i,j} = 0)\}$;
    end
    ⟅ We can apply now the classic Fourier-Motzkin elimination on each case ⟆
    foreach $\langle I^+, I^-, I^0, C \rangle \in P$ **do**
      $I \leftarrow I^0$;
      foreach $\langle c^+ + \sum_{i \le k \le n} a_k^+ x_k \le 0 \rangle \in I^+$ **do**
        foreach $\langle c^- + \sum_{i \le k \le n} a_k^- x_k \le 0 \rangle \in I^-$ **do**
          $I \leftarrow I \cup \{\langle (c^- a_i^+ - c^+ a_i^-) + \sum_{i+1 \le k \le n} (a_k^- a_i^+ - a_k^+ a_i^-) x_k \le 0 \rangle\}$;
        end
      end
      $\mathcal{I}' \leftarrow \mathcal{I}' \cup \{\langle C, I \rangle\}$;
    end
  end
  $\mathcal{I} \leftarrow \mathcal{I}'$;
end

Fig. 16: Algorithm of the parametric Fourier-Motzkin elimination.

4.2 Parametric Fourier-Motzkin Algorithm

We give in Fig. 16 the algorithm of the Fourier-Motzkin elimination. We have as inputs a set I_0 of m parametric inequalities of the form $\{c_j + \sum_{1 \le i \le n} a_{i,j} x_i \le 0 \mid 1 \le j \le m\}$ where each x_i is an unknown and each c_j and $a_{i,j}$ are parametric coefficients of arbitrary form. Additionally, we also provide a (possibly empty) set of constraints C_0 that gives initial information about the parameters (for example, a parameter p_l of a Bernoulli distribution is always in the range $[0, 1]$). The aim of the algorithm is to return a set of constraints equivalent to I_0 where all variables were eliminated except a single one (assume x_n). To do so, the algorithm eliminates the other variables sequentially. At each iteration, a variable is eliminated and we obtain a set of parametric solutions $\{\langle C, I \rangle\}$ where I are a set of linear constraints on the remaining unknowns and C are the conditions on the parameters for obtaining the solution I.

For the sake of clarity, let us first describe the classical non-parametric version of the algorithm. To eliminate a variable x_i, we examine its coefficients $a_{i,j}$ in I and we partition the inequalities depending on the sign of these coefficients (lines 5 – 8). The idea is that when having two inequalities $\langle c_{j_1} + \sum_{i \leq k \leq n} a_{k,j_1} x_k \leq 0 \rangle$ and $\langle c_{j_2} + \sum_{i \leq k \leq n} a_{k,j_2} x_k \leq 0 \rangle$ where the signs of the target coefficients a_{i,j_1} and a_{i,j_2} are different and not null (let say $a_{i,j_1} > 0$ and $a_{i,j_2} < 0$), we can generate a new inequality $\langle a_{i,j_1} c_{j_2} - a_{i,j_2} c_{j_1} + \sum_{i \leq k \leq n} (a_{i,j_1} a_{k,j_2} - a_{i,j_2} a_{k,j_1}) x_k \leq 0 \rangle$ that is implied by the previous inequalities and where the coefficient of x_i is null. So, if we perform the same merging operation on every couple of sign-opposite inequalities, while keeping the inequalities that have already a null coefficient on x_i, we obtain a set of inequalities equivalent to the previous ones and where x_i has been eliminated (lines 14 – 19).

When the coefficients of the unknowns are not constant, we cannot always determine their signs. Consequently, we collect the set of undetermined inequalities $I^?$ (line 9) and we eliminate the ambiguity by pushing the sign conditions into the parameters constraints C. In other words, we fork the inequalities I into a set of new inequalities: one for every possible sign combination of the undertermined coefficients. For each case, the conditions of the sign combination are accumulated with the current parameters conditions C, and the undetermined inequalities are classified depending on these sign conditions (line 12). To improve precision, the function **check** is used to test the satisfiability of the conjunction of C with the sign condition; cases with invalid formula are rejected. After resolving all coefficient signs, the classic Fourier-Motzkin elimination can be applied.

5 Non-deterministic Semantics

In this section, we explain how we can extend our previous analysis to handle pure non-determinism. We enrich the previous syntax of PSimpl shown in Fig. 6 with a boolean non-deterministic choice operator ⍰, and we show that previous concrete and abstract semantics can be easily adapted to preserve the correctness of the inferred distribution invariants.

5.1 Concrete Semantics

Markov decision processes are a well-known formalism for enriching the model of discrete time Markov chains with non-determinism [38]. Essentially, they represent stochastic processes that perform a non-deterministic choice at each state to select the transition probability distribution to employ. We can view such a system as if it were some adversary, called a *policy scheduler*, that tries to control its behavior [42]. Therefore, for a given policy scheduler that fixes all non-deterministic choices that will be made by the process, the system becomes a discrete time Markov chain.

Let $\mathbb{B}^\infty$ denote infinite sequences of boolean values and :: the append operator on sequences. We define the set of policy schedulers as $\Psi \stackrel{\text{def}}{=} \mathcal{L} \to \mathbb{B}^\infty$ mapping control locations of the operator ⍰ to a sequence of resolved boolean choices. We lift our previous pure probabilistic domain $\mathcal{D}$ to a non-deterministic probabilistic

semantics domain $\hat{\mathcal{D}} \stackrel{\text{def}}{=} \Psi \to \mathcal{D}$ by resolving *a priori* all non-deterministic choices depending on a given policy scheduler.

The concrete semantics of the operator ⍰ is given by the following transfer function:

$$\begin{array}{l}\hat{\mathbb{S}}[\![l : \texttt{x = ?;}]\!]\hat{R} \stackrel{\text{def}}{=} \\ \quad \text{let } \texttt{pred} = \lambda\psi.\, \lambda b.\, \psi[l' \mapsto ((l = l')?(b :: \psi(l)) : (\psi(l')))] \text{ in} \\ \quad \lambda\psi.\, \bigsqcup_{b \in \mathbb{B}} \mathbb{S}[\![l : \texttt{x = } b\texttt{;}]\!]\hat{R}(\texttt{pred}(\psi)(b))\end{array}$$

Essentially, the utility function $\texttt{pred}(\psi)(b)$ creates the predecessor policy scheduler of ψ that extends ψ with the boolean choice b at location l. The deterministic assignment is then applied on the Markov chains associated to the input schedulers that are predecessors of the output schedulers (used for the next resolution of ⍰). For the remaining statements, the corresponding transfer functions are obtained by a straightforward pointwise extension of their deterministic counterpart:

$$\hat{\mathbb{S}}[\![s]\!]\hat{R} \stackrel{\text{def}}{=} \lambda\psi.\, \mathbb{S}[\![s]\!](\hat{R}(\psi))$$

A similar lifting is employed to obtain the concrete stationary distribution $\hat{\boldsymbol{\pi}}[\![.]\!]$ from $\boldsymbol{\pi}[\![.]\!]$.

5.2 Policy Encoding

Since the set of sequences of natural numbers $\mathbb{N} \to \mathbb{N}$ has the same cardinality as $\mathbb{R}$, we can encode each policy scheduler as a distinct real number in $[0, 1]$. By considering this number as a parameter of the program and by a specific syntactical transformation of the operator ⍰, we can build a deterministic probabilistic program equivalent to the original non-deterministic one.

Formally, let $l \in L$ be the program location of some non-deterministic choice expression ⍰. We assume without loss of generality that every expression contains at most one occurrence of the ⍰ operator, and that this operator can appear only in the right-hand-side part of assignments. We attach to every occurrence three fresh auxiliary variables $\beta_l \in \mathbb{B}$, $\mu_l \in \mathbb{R}$ and $\mu_{l,0} \in \mathbb{R}$. The boolean variable β_l stores the result of the non-deterministic choice. The real variable μ_l, initialized by the parameter value $\mu_{l,0} \in [0, 1]$, is used to encode the sequence of values of β_l. We replace every occurrence of ⍰ with the variable β_l, and we add before this occurrence the following code that extracts a new random bit from μ_l, stores it into β_l, and shifts μ_l by one bit:

l : x = ⍰ ; →

```
if (μ_l >= 0.5) {
  β_l = TRUE;
} else {
  β_l = FALSE;
}
μ_l = 2 * fmod(μ_l, 0.5);
x = β_l;
```

where $\texttt{fmod}(a, b)$ is the floating-point remainder of the division a/b. By doing so, each initial value $\mu_{l,0}$ generates an infinite and unique sequence of boolean values.

Since we encoded a non-deterministic program into an parametric deterministic one that is equivalent, all our previous analysis can be applied. Safe bounds for the original program can be obtained by extracting the worst case values when varying the policy-encoding parameter in $[0, 1]$, *i.e.*, by projecting out this parameter.

Abstraction. We use a naive abstraction that allows simple and efficient computations while keeping the analysis sound. Instead of maintaining precise information about the encoding of boolean choices and the auxiliary variables β_l and μ_l, we simply forget their values and we join the results of both non-deterministic branches to cover all possible policies of the process. For instance, we can abstract non-deterministic assignments as follows:

$$\mathbb{S}^\sharp[\![\, \mathtt{x} = ?;]\!](\tau^\sharp, \omega^\sharp) \stackrel{\text{def}}{=} \bigsqcup_{b \in \mathbb{B}} \mathbb{S}^\sharp[\![\, \mathtt{x} = b;]\!](\tau^\sharp, \omega^\sharp)$$

which is similar to the classic abstraction of non-determinism in non-probabilistic programs. Since the auxiliary variables are not referenced in the abstract semantics, no special processing is required to eliminate them and we are guaranteed to cover all possible sequences of non-deterministic choices. Therefore, we can apply the previous extraction/resolution method in order to compute the bounds of the stationary distribution.

6 Experiments

The proposed approach has been implemented in a prototype analyzer called Marchal (*MARkov CHains AnaLyzer*) composed of two parts. The first one is an abstract interpreter implemented in the OCaml language and based on the CIL frontend [39] and the Apron library [27]; it operates by structural induction on an input C-like probabilistic program in order to infer its abstract Markov chain. The second part is an implementation in Mathematica [26] of the parametric Fourier-Motzkin elimination algorithm that finds symbolic bounds of the stationary distribution of the abstract Markov chain of interest.

We have considered a wireless transmission scenario over lossy links characterized by a drop probability p. Five well-known backoff mechanisms have been considered, that will be denoted by $\mathbf{C}_1$, $\mathbf{C}_\infty$, $\mathbf{C}_n$, $\mathbf{L}_n$ and $\mathbf{L}_{t\infty}$. The first mechanism $\mathbf{C}_1$ is our motivating example of Fig. 1a that uses a single constant backoff window of length B and does not retransmit the packet in case of loss. The mechanism $\mathbf{C}_\infty$ uses also a constant backoff window to avoid collisions but improves reliability by trying to send the packet until an acknowledgement is received. The mechanism $\mathbf{C}_n$ denotes a constant backoff window with a limited number of retransmissions fixed by a parameter n. Additionally, we have tested two backoff mechanisms with a dynamic window that increases linearly at each failed transmission. The mechanism $\mathbf{L}_n$ bounds the number of transmissions by a parameter n, while the mechanism $\mathbf{L}_{t\infty}$ uses a truncated policy in which the number of attempts is unbounded but the maximal window length is limited by a parameter t. For all mechanisms, a sleep period of duration S follows every transmission phase. The programs of these mechanisms are presented in Appendix B.

Our prototype Marchal has been compared to the state-of-the-art probabilistic model checker Prism [30]. More specifically, we used its parametric engine based

on PARAM [25] that can produce closed-form stationary distributions when the transition probabilities are symbolic. For MARCHAL, we have varied the partitioning parameter U of uniform distributions and we have performed a non-relational analysis using the interval domain BOX and a relational analysis using the polyhedra domain POLY. In order to highlight the differences between MARCHAL and PRISM, we have selected four deterministic scenarios for each backoff mechanism with various ranges of parameters: fixed small values, fixed large values, large ranges and open ranges. In addition, one non-determinisitic case with large ranges has been analyzed using the same non-perfect clock model as presented earlier in Fig. 4. Note that the cases of open ranges and non-determinism are not supported by PRISM.

6.1 Efficiency

The efficiency of both tools is measured in terms of analysis time. The obtained results are summarized in Table 1 and are reported in seconds. For MARCHAL, we have also divided the overall analysis time into two parts: the abstract Markov chain (*AMC*) extraction phase and the parametric Fourier-Motzkin (*PFM*) resolution phase. A timeout of 30 mins is used as a limit for the overall analysis time.

PRISM. It is important to note that PRISM performs a precise analysis in all cases. On the one hand, this allows fast analysis times of simple cases such as configurations with fixed small values. However for the other cases, the models increase in size and complexity, and the scalability of PRISM is affected due to the absence of approximation mechanisms. More specifically, PRISM has to perform a separate analysis for every possible value of the parameters, resulting in systematic timeouts for the range cases. Also, as an obvious consequence, open range cases are not supported. Another limitation of PRISM is its partial support for non-determinism that was not useful for the analysis of our benchmark protocols. More precisely, despite the fact that PRISM does support non-determinism modeled as Markov decision processes, the extraction of the stationary distribution is available only for discrete and continuous time Markov chains, not for Markov decision processes.

MARCHAL. For fixed values scenarios, PRISM performs better than MARCHAL in most cases. The strength of our approach becomes more clear for the cases of ranges, where MARCHAL was able to return an answer before the timeout in all cases, at least using the BOX domain. Some timeouts occurred, however, with the relational domain POLY. As we can notice from the convergent cases, the *PFM* proportion is always predominant in the overall analysis time. Most of these computations are performed when checking the signs of parametric coefficients during the resolution process (lines 5, 6 and 7 in Fig. 16), for which we use the Mathematica API.

However, since it is always sound to remove constraints from a system of inequalities, a simple optimization consists in ignoring inequalities for which the sign check procedure consumes an excessive amount of time. This is done by constraining the duration of the evaluation of constraints $a_{i,j} > 0$, $a_{i,j} < 0$ and $a_{i,j} = 0$ in lines 5, 6 and 7 respectively in Fig. 16. In our experiments, this duration guard

Table 1: Analysis time, reported in seconds.
Legend: *AMC*: abstract Markov chain extraction, *PFM*: parametric Fourier-Motzik resolution, ✱: presence of non-determinism, ∞: timeout of 30 mins, ✘: unsupported analysis case.

Protocol	Prism	U	Marchal					
			Box			Poly		
			AMC	*PFM*	*Total*	*AMC*	*PFM*	*Total*
$\mathbf{C}_1$: *Constant single backoff*								
$B = 3, S = 10^2$	1.72	2	0.71	2.59	3.3	1.46	2.87	4.3
		3	0.99	2.62	3.6	2.38	2.70	5.1
$B = 20, S = 10^3$	10.80	2	0.68	2.60	3.3	1.47	2.71	4.2
		3	0.82	2.94	3.8	2.39	2.82	5.2
$B \in [3, 20], S \in [10^2, 10^3]$	∞	2	0.77	2.53	3.3	2.04	2.50	4.5
		3	1.06	2.68	3.7	3.29	2.74	6.0
$B \in [3, 20], S \in [10^2, 10^3]$ ✱	✘	2	0.85	2.61	3.5	2.79	2.80	5.6
		3	1.05	2.60	3.7	4.87	3.00	7.9
$B \geq 3, S \geq 10^2$	✘	2	0.68	2.41	3.1	1.66	2.63	4.3
		3	0.87	2.56	3.4	3.12	2.78	5.9
$\mathbf{C}_\infty$: *Constant unbounded backoffs*								
$B = 3, S = 10^2$	1.59	2	0.95	3.16	4.1	2.54	3.38	5.9
		3	1.79	4.82	6.6	4.85	3.31	8.2
$B = 20, S = 10^3$	10.32	2	1.07	3.33	4.4	2.70	3.36	6.1
		3	1.84	5.09	6.9	4.92	4.47	9.4
$B \in [3, 20], S \in [10^2, 10^3]$	∞	2	1.16	3.20	4.4	3.49	5.02	8.5
		3	1.92	3.39	5.3	6.61	5.50	12.1
$B \in [3, 20], S \in [10^2, 10^3]$ ✱	✘	2	1.07	4.79	5.9	5.08	8.97	14.1
		3	2.03	5.26	7.3	10.28	21.13	31.4
$B \geq 3, S \geq 10^2$	✘	2	1.06	2.50	3.6	2.81	5.21	8.0
		3	1.81	2.52	4.3	5.98	5.68	11.7
$\mathbf{C}_n$: *Constant bounded backoffs*								
$B = 3, S = 10^2, n = 2$	1.74	2	2.49	10.77	13.3	8.10	10.30	18.4
		3	4.21	14.63	18.8	14.75	15.12	29.9
$B = 20, S = 10^3, n = 6$	38.07	2	5.96	13.72	19.7	23.08	13.55	36.6
		3	11.22	25.22	36.4	48.46	24.75	73.2
$B \in [3, 20], S \in [10^2, 10^3], n \in [2, 6]$	∞	2	5.39	10.37	15.8	39.90	180.84	220.7
		3	10.40	14.36	24.8	74.55	1136.72	1211.3
$B \in [3, 20], S \in [10^2, 10^3], n = 2$	∞	2	2.60	10.47	13.1	11.14	168.97	180.1
		3	4.42	15.67	20.1	22.72	809.66	832.4
$B \in [3, 20], S \in [10^2, 10^3], n = 2$ ✱	✘	2	2.96	10.42	13.4	16.44	95.43	111.9
		3	4.99	9.55	14.5	29.03	397.72	426.8
$B \geq 3, S \geq 10^2, n \geq 2$	✘	2	5.08	2.73	7.8	26.58	167.51	194.1
		3	9.46	2.86	12.3	50.18	991.01	1041.2
$\mathbf{L}_n$: *Linear bounded backoffs*								
$B = 3, S = 10^2, n = 2$	1.74	2	2.33	10.69	13.0	8.15	10.60	18.8
		3	4.20	15.51	19.7	15.01	15.77	30.8
$B = 20, S = 10^3, n = 6$	13.36	2	5.99	5.09	11.1	22.84	28.88	51.7
		3	11.11	7.25	18.4	45.63	51.17	96.8
$B \in [3, 20], S \in [10^2, 10^3], n \in [2, 6]$	∞	2	5.60	3.26	8.9		∞	
		3	10.19	7.56	17.8		∞	
$B \in [3, 20], S \in [10^2, 10^3], n = 2$	∞	2	2.65	11.06	13.7	11.34	135.54	146.9
		3	4.24	15.51	19.8	20.92	669.55	690.5
$B \in [3, 20], S \in [10^2, 10^3], n = 2$ ✱	✘	2	2.95	8.86	11.8	16.48	58.10	74.6
		3	4.93	10.32	15.2	30.94	395.34	426.3
$B \geq 3, S \geq 10^2, n \geq 2$	✘	2	5.22	2.80	8.0		∞	
		3	9.75	3.01	12.8		∞	
$\mathbf{L}_{t\infty}$: *Linear truncated backoffs*								
$B = 3, S = 10^2, t = 2$	1.74	2	2.01	13.60	15.6	5.85	13.34	19.2
		3	3.64	19.96	23.6	11.73	19.14	30.9
$B = 20, S = 10^3, t = 6$	9.74	2	7.06	5.48	12.5	29.57	5.57	35.1
		3	13.27	6.31	19.6	61.76	7.05	68.8
$B \in [3, 20], S \in [10^2, 10^3], t \in [2, 6]$	∞	2	3.54	3.39	6.9		∞	
		3	6.40	5.44	11.8		∞	
$B \in [3, 20], S \in [10^2, 10^3], t = 2$	∞	2	2.12	11.96	14.1	8.28	160.09	168.4
		3	3.76	12.81	16.6	16.23	268.40	284.6
$B \in [3, 20], S \in [10^2, 10^3], t = 2$ ✱	✘	2	2.27	10.19	12.5	11.85	122.16	134.0
		3	4.12	11.65	15.8	23.96	1682.07	1706.0
$B \geq 3, S \geq 10^2, t \geq 2$	✘	2	3.28	2.90	6.2		∞	
		3	6.08	3.04	9.1	29.52	1054.99	1084.5

was fixed to 5 seconds; when this guard is reached, the constraint is not added to the system P, and the algorithm continues the elimination process with the other constraints. In most cases, this ensured that the overall analysis does not reach the global 30 mins time limit; however, a few timeouts remained.

Another issue of our approach is its inefficiency in the presence of a bounding parameter that limits the number of transmissions, such as parameter n for cases $\mathbf{C}_n$ and $\mathbf{L}_n$, and parameter t for $\mathbf{L}_{t\infty}$. In most experiments, a timeout occurred using the relational domain Poly. The main reason behind this problem is that our abstract domain is not able to reflect the impact of such parameters on the stationary distribution invariants. When analyzing a loop `while` (e) s where statement s contains a sequence of observable states, we are not able to reflect the impact of the exit condition e on the aggregate sojourn time of those states, because sojourn times are attached to observable states individually. Consequently, the bounded loop is approximated with an unbounded one and the information of the bounding parameter is ignored. This results in complex stationary distribution invariants that are not well adapted for the resolution algorithm. For this reason, we have added an additional scenario for $\mathbf{C}_n$, $\mathbf{L}_n$ and $\mathbf{L}_{t\infty}$ in which the bounding parameter has a small fixed value of 2. This allows us to perform an unrolling of the transmission loop. As a consequence, the analysis time of Marchal was improved and all timeouts disappeared, while Prism was not able to analyze these scenarios.

6.2 Precision

In this section, we quantify the loss of precision induced by the approximations of our analysis. Note that the outcome of Marchal is a symbolic expression of the stationary distribution that is always in the range $[0, 1]$. Measuring the imprecision of such expressions is not obvious because it varies depending on the value of the parameters. Therefore, we compute numerically the average gap between the obtained upper and lower bounds over the entire ranges of the parameters for each configuration. The smaller is the gap, the better is the precision: 0 means finding the exact solution, while 1 means that no interesting one was found. Obviously, by doing so, the precision of open range scenarios cannot be computed, so we omit them in this study.

The obtained results are shown in Table 2. Note that we do not present the results of Prism because it produces always precise results with no errors. Instead, we show the relative performance speedup of Marchal over Prism to illustrate the precision/efficiency tradeoff of the analysis (positive numbers indicating that Marchal was faster).

In 12 of the 23 cases, Marchal produced small error gaps (< 0.2): Prism was faster in 5 of these cases; while in 6 of them, Prism was not able to find a solution. In the remaining 11 of the 23 cases, the results obtained by Marchal were too coarse. This is due to (*i*) the naive partitioning abstraction of the uniform distribution, and (*ii*) the presence of bounded loops that are not handled properly by our semantics as discussed earlier in the efficiency study. Nevertheless, in 7 of these 11 cases, Prism was not able to return an answer within the fixed timeout. In summary, we can conclude that the approaches of Marchal and Prism are complementary:

Table 2: Analysis error, computed as the average gap between the upper and lower bounds of the stationary distribution. The speedup column gives the improvement in time when comparing with PRISM. The column BOX ⊕ POLY reports the result of combining both domains.
Legend: **bold**: minimal error per case, ✱: presence of non-determinism, ∞: MARCHAL timeout, ++: PRISM timeout.

Protocol	U	BOX		POLY		BOX ⊕ POLY	
		Error	Speedup	*Error*	Speedup	*Error*	Speedup
$\mathbf{C}_1$: *Constant single backoff*							
$B = 3, S = 10^2$	2	0.003	$-1.6s$	0.003	$-2.6s$	0.003	$-5.9s$
	3	**0.000**	$-1.9s$	0.000	$-3.4s$	0.000	$-7.0s$
$B = 20, S = 10^3$	2	0.008	$+7.5s$	0.008	$+6.6s$	0.008	$+3.3s$
	3	**0.005**	$+7.0s$	0.005	$+5.6s$	0.005	$+1.8s$
$B \in [3, 20], S \in [10^2, 10^3]$	2	0.218	++	0.010	++	0.010	++
	3	0.154	++	**0.006**	++	0.006	++
$B \in [3, 20], S \in [10^2, 10^3]$ ✱	2	0.222	++	0.012	++	0.012	++
	3	0.158	++	**0.007**	++	0.007	++
$\mathbf{C}_\infty$: *Constant unbounded backoffs*							
$B = 3, S = 10^2$	2	0.236	$-2.5s$	0.237	$-4.3s$	0.237	$-8.4s$
	3	**0.000**	$-5.0s$	0.000	$-6.6s$	0.000	$-13.2s$
$B = 20, S = 10^3$	2	0.894	$+5.9s$	0.894	$+4.3s$	0.894	$-0.1s$
	3	**0.834**	$+3.4s$	0.834	$+0.9s$	0.834	$-6.0s$
$B \in [3, 20], S \in [10^2, 10^3]$	2	0.978	++	0.739	++	0.740	++
	3	0.963	++	**0.600**	++	0.600	++
$B \in [3, 20], S \in [10^2, 10^3]$ ✱	2	0.978	++	0.792	++	0.792	++
	3	0.967	++	**0.680**	++	0.680	++
$\mathbf{C}_n$: *Constant bounded backoffs*							
$B = 3, S = 10^2, n = 2$	2	0.007	$-11.5s$	0.007	$-16.7s$	0.007	$-29.9s$
	3	**0.000**	$-17.1s$	0.000	$-28.1s$	0.000	$-47.0s$
$B = 20, S = 10^3, n = 6$	2	0.898	$+18.4s$	0.898	$+1.4s$	0.898	$-18.2s$
	3	**0.838**	$+1.6s$	0.838	$-35.1s$	0.838	$-71.6s$
$B \in [3, 20], S \in [10^2, 10^3], n \in [2, 6]$	2	0.982	++	0.749	++	0.749	++
	3	0.961	++	**0.667**	++	0.667	++
$B \in [3, 20], S \in [10^2, 10^3], n = 2$	2	0.321	++	**0.039**	++	0.039	++
	3	0.237	++	0.950	++	0.223	++
$B \in [3, 20], S \in [10^2, 10^3], n = 2$ ✱	2	0.323	++	0.044	++	0.044	++
	3	0.240	++	**0.022**	++	0.022	++
$\mathbf{L}_n$: *Linear bounded backoffs*							
$B = 3, S = 10^2, n = 2$	2	0.009	$-11.3s$	0.010	$-17.0s$	0.010	$-30.0s$
	3	**0.001**	$-18.0s$	0.001	$-29.0s$	0.001	$-48.8s$
$B = 20, S = 10^3, n = 6$	2	1.000	$+2.3s$	0.935	$-38.4s$	0.935	$-49.4s$
	3	1.000	$-5.0s$	**0.893**	$-83.4s$	0.893	$-101.8s$
$B \in [3, 20], S \in [10^2, 10^3], n \in [2, 6]$	2	**1.000**	++		∞		∞
	3	1.000	++		∞		∞
$B \in [3, 20], S \in [10^2, 10^3], n = 2$	2	0.322	++	**0.041**	++	0.041	++
	3	0.238	++	0.237	++	0.068	++
$B \in [3, 20], S \in [10^2, 10^3], n = 2$ ✱	2	0.323	++	0.046	++	0.046	++
	3	0.240	++	**0.023**	++	0.023	++
$\mathbf{L}_{t\infty}$: *Linear truncated backoffs*							
$B = 3, S = 10^2, t = 2$	2	0.414	$-13.9s$	0.414	$-17.4s$	0.414	$-33.1s$
	3	**0.124**	$-21.9s$	0.124	$-29.1s$	0.124	$-52.7s$
$B = 20, S = 10^3, t = 6$	2	**1.000**	$-2.8s$	1.000	$-25.4s$	1.000	$-37.9s$
	3	1.000	$-9.8s$	1.000	$-59.1s$	1.000	$-78.7s$
$B \in [3, 20], S \in [10^2, 10^3], t \in [2, 6]$	2	**1.000**	++		∞		∞
	3	1.000	++		∞		∞
$B \in [3, 20], S \in [10^2, 10^3], t = 2$	2	0.970	++	0.782	++	0.782	++
	3	0.942	++	**0.653**	++	0.653	++
$B \in [3, 20], S \in [10^2, 10^3], t = 2$ ✱	2	0.962	++	0.815	++	0.814	++
	3	0.944	++	**0.712**	++		∞

MARCHAL is not always the most optimal, but can compute interesting results that are out-of-the scope of PRISM's approach.

These preliminary results show the importance of tunable approximations that allow MARCHAL to tradeoff precision and efficiency, contrary to PRISM that returns always precise results. Three factors influenced the tradeoff of MARCHAL:

- Even though the partitioning approach of the uniform distribution is a naive abstraction, many cases were improved by adjusting properly its parameter U.
- The relational invariants provided by the POLY impacted considerably the precision, but at the cost of efficiency.
- The number of unrollings of `while` loops was also an important tuning parameter of the analysis.

Some few particular cases violate the monotony of this tradeoff. For instance, consider the $\mathbf{C}_n$ protocol and the case $B \in [3, 20], S \in [10^2, 10^3], n = 2$. The precision decreased from 0.321 using BOX/$U = 2$ to 0.950 using POLY/$U = 3$, which is counter-intuitive. Essentially, by increasing the precision of the domain, the distribution invariants can become more complex and difficult to handle by Mathematica. Therefore, some constraints are ignored to ensure convergence, which leads to partial solutions not covering all the range of the parameters. For the uncovered region, the trivial bound $[0, 1]$ is assumed so that we are nevertheless sound, albeit we decrease the precision of the overall result. Nevertheless, the invariants of both cases can be merged by considering the most precise bounds. The result of this combination is reported in the column BOX $\oplus$ POLY in Table 2 and shows an improvement of the precision in these particular cases.

7 Related Work

The analysis of probabilistic programs has gained great interest over the last years. Many techniques have been proposed with varying precision/scalability tradeoffs. Overall, two kinds of quantitative properties have been considered:

Distribution inference. Most existing tools aim at inferring the probability of reaching particular program states. This kind of analysis extends the classic (qualitative) notion of state reachability to provide more refined (quantitative) answers about the program safety, e.g. the likelihood of violating an assertion. In the literature, this is designated as *distribution inference*, *bayesian inference* or *probabilistic reachability*.

Expectation invariants. Other works focus on finding invariants about the expectation of some program variable or expression. An expectation gives the mean value of an expression by considering all scenarios weighted with their probabilities. Note that a distribution inference analysis can be used to obtain expectation invariants. However, a tailored expectation analysis can be more efficient.

It is worth noting that the kind of properties investigated in our work is different from those two notions. We are interested in computing rates at which performance indicators change during time (*e.g.* rates of packet transmission, energy consumption, *etc.*). Computing such rates is based on finding the stationary

distribution of the process, which is different than computing reachability probabilities or expectations. For this reason, existing verification techniques are not adequate to infer the kind of properties we are interested in, except Prism as we will discuss later.

In the following, we give an overview of the most representative solutions in the literature:

7.1 Model Checking

Daws [14] presented a theoretic-language approach to find exact symbolic probabilities of events expressed in PCTL. The analysis is limited to parametric discrete time Markov chains with finite state spaces unlike our approach. By considering a chain as an automaton over the alphabet of probability events, symbolic reachability probabilities can be encoded as regular expressions using a state elimination algorithm. The obtained regular expressions are evaluated symbolically by structural induction to extract rational functions giving the desired parametric reachability probability. Several enhancements of this approach were proposed to support non-determinism [25] and conditional probabilities [16].

Prism [30] is a famous model checker that has been successfully applied for analyzing many probabilistic systems. It supports several stochastic models, such as discrete and continous Markov chains, Markov decision processes and probabilistic timed automata. In addition to a numeric resolution engine, Prism integrated the parametric reachability analysis of Param [25] which allows computing symbolic stationary distribution of discrete time Markov chains, but it is limited to finite state spaces.

7.2 Symbolic Execution

Geldenhuys et al. [22] extended the Symbolic PathFinder engine [2] to compute exact reachability probabilities. The analysis targets (non-probabilistic) functions with symbolic input parameters drawn from finite uniform distributions. Symbolic execution traces are enriched with path probabilities computed by dividing the number of reaching paths by the total space of values of the inputs. To do so, volume counting techniques [15] are required, which limits the scalability of the approach. Several other techniques extend this approach to support multi-threaded programs [20], to handle non-determinism [33], or to use Monte Carlo sampling for better efficiency [21].

Sankaranarayanan et al. [44] proposed another symbolic approach that can infer formally guaranteed bounds of reachability probabilities. It targets infinite state probabilistic programs with various discrete and continous distributions. Also, the authors propose a branch-and-bound technique to perform sound and approximate volume counting.

More recently, Barthe et al. [3] described a symbolic execution based on martingales in order to derive post-loop expectations of program variables. Informally, a martingale is an expression having an expectation that does not change. The proposed technique uses Doob's decomposition in order to infer martingale expres-

sions automatically. After that, post-loop expectations are computed by applying the optional stopping theorem.

7.3 Static Analysis

Abstract interpretation of probabilistic programs was introduced by Monniaux [36] to compute upper-bounds of reachability probabilities. The analysis lifts standard concrete non-probabilistic semantics to probabilistic semantics by extending the measure-based formalization of Kozen [29] in order to handle non-determinism. Later, Monniaux extended this work to support backward reachabilty analysis [37] and LTL properties on Markov decision processes [38].

In the same line, Bouissou et al. [4] developed a probabilistic abstract interpretation of numeric programs. The aim of the analysis is to quantify rounding errors during numeric computations by propagating noise-related uncertainties as probabilities. The analysis is based on p-boxes [19] and Dempster–Shafer structures [45], but lacks a widening operator. An enhancement of this approach was proposed in [5] that employs concentration of measure inequalities [17].

In [11], Cousot et al. proposed a systematic framework for formalizing probabilistic abstract interpretations by introducing the concept of *law abstraction* as a means to approximate probability distributions on program states. This formalism provides general theoretic guidelines to build sound probabilistic abstract interpretations, but does not provide practical solutions for widening loop iterations.

Chakarov et al. [7] presented a static analysis that extends the weakest pre-expectation calculus of McIver and Morgan [34] to compute reachability probabilities and to prove almost sure termination. In [8], Chakarov et al. proposed another pre-expectation based analysis using abstract interpretation for discovering expectation invariants through the abstract domain of polyhedra with an appropriate widening operator.

Wang et al. [51] proposed another systematic framework for backward data flow analysis of probabilistic programs. Domains are formalized as measurable spaces over program states, and transfer functions correspond to kernels giving the probability that execution of a statement will hit some target environment. The analysis is intra-procedural and modular by computing function summaries that maintain sound input-output relations. The framework makes distinction between three kinds of widenings, depending on the exit condition of the loop: (classic) conditional, non-deterministic or probabilistic. Two instances of the framework are presented: a bayesian inference computing lower-bounds of probability reachability distributions, and a linear expectation invariant analysis over polyhedra.

7.4 Minimization of Markov Chains

In addition to verification approaches, conservative minimization techniques for Markov chains have been extensively investigated. Chain lumping [28] consists in downsizing a Markov chain by constructing a quotient Markov chain over some equivalence relation. By imposing particular constraints on this relation, one can relate the stationary distribution of the lumped chain to the original concrete one. In our approach, no constraint is imposed on the equivalence relation, albeit the

obtained stationary distribution bounds may be too coarse if the relation is not carefully designed. In addition, the construction of the lumped chain is performed dynamically by structural induction on the program syntax, which is not the case for classic lumping techniques that require a prior knowledge of the entire concrete chain.

Abate et al. [1] proposed another conservative minimization technique for discrete time Markov process with general (uncountable) state spaces. The basic idea is to derive a finite state Markov chain that enjoys particular approximation guarantees. Using classic probabilistic model checkers, such as Prism, the approximate chain is analyzed to derive the desired reachability probability. The result is combined with the approximation guarantees to provide a guaranteed error bound w.r.t. the reachability probability of the same property on the original Markov process. Soudjani et al. [46] extended this approach to approximate the state probability as a function of time. However, it not clear how these results can be adapted to bound the stationary distribution of the original process.

8 Conclusion

We have presented a novel approach for obtaining guaranteed bounds of performance metrics of communication protocols. The method is based on the framework of abstract interpretation and proposes an abstract Markov chain domain for approximating the semantics of programs with probabilistic and non-deterministic choices. We have also explained how to exploit the information encapsulated within this domain in order to infer a sound approximation of the stationary distribution of the protocol, which is the key ingredient for computing a large range of performance metrics such as the throughput and the energy consumption. A prototype of the analysis has been presented along with some preliminary results.

Many problems about enhancing the proposed approach are still open. As reported by our benchmarks results, the efficiency and the precision of the prototype analyzer are significantly affected in cases where the program uses a bounding parameter to limit transmission attempts. To overcome this limitation, we can enrich our abstraction by inferring invariants about *macro sojourn time* that reflect the relations between the overall sojourn time in particular sequences of states (e.g. within a `while` loop). Another problem is related to the parametric resolution step of our analysis. In general, the required time to perform the projection is predominant in the overall analysis time, and we believe that the efficiency of the resolution algorithm can still be improved by introducing approximations. Also, the presented analysis targeted a simple C-like language and we would like to extend it to support real-world programs in full-fledged C and more complex probability distributions. Finally, our work supports a single process model and we are interested in extending it to communicating concurrent programs.

References

1. A. Abate, J.-P. Katoen, J. Lygeros, and M. Prandini. Approximate model checking of stochastic hybrid systems. *European Journal of Control*, 16(6):624 – 641, 2010.
2. S. Anand, C. S. Păsăreanu, and W. Visser. JPF-SE: A symbolic execution extension to Java PathFinder. In *TACAS'07*, volume 4424 of *LNCS*, pages 134–138. Springer, 2007.

3. G. Barthe, T. Espitau, L. Ferrer Fioriti, and J. Hsu. Synthesizing probabilistic invariants via Doob's decomposition. In *CAV '16*, volume 9779 of *LNCS*, pages 43–61. Springer, 2016.
4. O. Bouissou, E. Goubault, J. Goubault-Larrecq, and S. Putot. A generalization of p-boxes to affine arithmetic. *Computing*, 94(2):189–201, 2012.
5. O. Bouissou, E. Goubault, S. Putot, A. Chakarov, and S. Sankaranarayanan. Uncertainty propagation using probabilistic affine forms and concentration of measure inequalities. In *TACAS '16*, volume 9636 of *LNCS*, pages 225–243. Springer, 2016.
6. P. Buchholz. Exact and ordinary lumpability in finite markov chains. *Journal of Applied Probability*, 31(1):59–75, 1994.
7. A. Chakarov and S. Sankaranarayanan. Probabilistic program analysis with martingales. In *CAV '13*, volume 8044 of *LNCS*, pages 511–526. Springer, 2013.
8. A. Chakarov and S. Sankaranarayanan. Expectation invariants for probabilistic program loops as fixed points. In *SAS '14*, volume 8723 of *LNCS*, pages 85–100. Springer, 2014.
9. P. Cousot and R. Cousot. Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *POPL '77*, pages 238–252. ACM, 1977.
10. P. Cousot and N. Halbwachs. Automatic discovery of linear restraints among variables of a program. In *POPL '78*, pages 84–97. ACM, 1978.
11. P. Cousot and M. Monerau. Probabilistic abstract interpretation. In *ESOP '12*, volume 7211 of *LNCS*, pages 169–193. Springer, 2012.
12. R. Cousot. *Fondements des méthodes de preuve d'invariance et de fatalité de programmes parallèles*. Thèse d'État ès sciences mathématiques, Institut National Polytechnique de Lorraine, Nancy, France, 1985.
13. G. R. Dattatreya. *Performance Analysis of Queuing and Computer Networks*. Chapman and Hall/CRC, 2008.
14. C. Daws. Symbolic and parametric model checking of discrete-time Markov chains. In *ICTAC '04*, volume 3407 of *LNCS*, pages 280–294. Springer, 2004.
15. J. A. De Loera, R. Hemmecke, J. Tauzer, and R. Yoshida. Effective lattice point counting in rational convex polytopes. *Journal of Symbolic Computation*, 38(4):1273 – 1302, 2004.
16. C. Dehnert, S. Junges, N. Jansen, F. Corzilius, M. Volk, H. Bruintjes, J.-P. Katoen, and E. Ábrahám. PROPhESY: A PRObabilistic ParamEter SYnthesis tool. In *CAV '15*, volume 9206 of *LNCS*, pages 214–231. Springer, 2015.
17. D. P. Dubhashi and A. Panconesi. *Concentration of Measure for the Analysis of Randomized Algorithms*. Cambridge University Press, 2009.
18. J. Feret. Abstract interpretation-based static analysis of mobile ambients. In *SAS '01*, volume 2126 of *LNCS*, pages 412–430. Springer, 2001.
19. S. Ferson, V. Kreinovick, L. Ginzburg, and F. Sentz. Constructing probability boxes and Dempster-Shafer structures. Technical report, SAND2002-4015, 2003.
20. A. Filieri, C. S. Păsăreanu, and W. Visser. Reliability analysis in symbolic pathfinder. In *ICSE '13*, pages 622–631. IEEE Press, 2013.
21. A. Filieri, C. S. Păsăreanu, W. Visser, and J. Geldenhuys. Statistical symbolic execution with informed sampling. In *FSE '14*, pages 437–448. ACM, 2014.
22. J. Geldenhuys, M. B. Dwyer, and W. Visser. Probabilistic symbolic execution. In *ISSTA '12*, pages 166–176. ACM, 2012.
23. G. Grimmett and D. Stirzaker. *Probability and Random Processes*. Oxford University Press, 2001.
24. A. Grïßlinger. Extending the polyhedron model to inequality systems with non-linear parameters using quantifier elimination. Master thesis, University of Passau, 2003.
25. E. Hahn, H. Hermanns, and L. Zhang. Probabilistic reachability for parametric markov models. *International Journal on Software Tools for Technology Transfer*, 13(1):3–19, 2011.
26. Wolfram Research, Inc. Mathematica, Version 11.2, 2017. Champaign, IL.
27. B. Jeannet and A. Miné. Apron: A library of numerical abstract domains for static analysis. In *CAV '09*, volume 5643 of *LNCS*, pages 661–667. Springer, 2009.
28. J. G. Kemeney and J. L. Snell. *Finite Markov Chains*. Springer-Verlag, 1976.
29. D. Kozen. A probabilistic PDL. *Journal of Computer and System Sciences*, 30(2):162 – 178, 1985.
30. M. Kwiatkowska, G. Norman, and D. Parker. PRISM 4.0: Verification of probabilistic real-time systems. In *CAV '11*, volume 6806 of *LNCS*, pages 585–591. Springer, 2011.

31. T. Le Gall and B. Jeannet. Lattice automata: A representation for languages on infinite alphabets, and some applications to verification. In *SAS '07*, volume 4634 of *LNCS*, pages 52–68. Springer, 2007.
32. D. Lesens, N. Halbwachs, and P. Raymond. Automatic verification of parameterized networks of processes. *Theoretical Computer Science*, 256(1-2):113–144, 2001.
33. K. Luckow, C. S. Păsăreanu, M. B. Dwyer, A. Filieri, and W. Visser. Exact and approximate probabilistic symbolic execution for nondeterministic programs. In *ASE '14*, pages 575–586. ACM, 2014.
34. A. McIver and C. Morgan. *Abstraction, Refinement And Proof For Probabilistic Systems*. Monographs in Computer Science. Springer, 2004.
35. A. Miné. The octagon abstract domain. *Higher-Order and Symbolic Computation (HOSC)*, 19(1):31–100, 2006.
36. D. Monniaux. Abstract interpretation of probabilistic semantics. In *SAS '00*, volume 1824 of *LNCS*, pages 322–339. Springer, 2000.
37. D. Monniaux. Backwards abstract interpretation of probabilistic programs. In *ESOP '01*, volume 2028 of *LNCS*, pages 367–382. Springer, 2001.
38. D. Monniaux. Abstract interpretation of programs as Markov decision processes. *Science of Computer Programming*, 58:179–205, 2005.
39. G. Necula, S. McPeak, S. Rahul, and W. Weimer. CIL: Intermediate language and tools for analysis and transformation of C programs. In *CC '02*, pages 213–228, 2002.
40. A. Ouadjaout and A. Miné. Quantitative static analysis of communication protocols using abstract Markov chains. In *SAS '17*, volume 10422 of *LNCS*, pages 277–29. Springer, 2017.
41. R. Parikh. On context-free languages. *Journal of ACM*, 13(4):570–581, 1966.
42. M. L. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. Wiley Series in Probability and Statistics. John Wiley & Sons, 1994.
43. S. Ross. *Stochastic Processes*. John Wiley & Sons, 1996.
44. S. Sankaranarayanan, A. Chakarov, and S. Gulwani. Static analysis for probabilistic programs: Inferring whole program properties from finitely many paths. In *PLDI '13*, pages 447–458. ACM, 2013.
45. G. Shafer. *A Mathematical Theory of Evidence*. Princeton University Press, 1976.
46. S. E. Z. Soudjani and A. Abate. Precise approximations of the probability distribution of a Markov process in time: An application to probabilistic invariance. In *TACAS '14*, volume 8413 of *LNCS*, pages 547–561. Springer, 2014.
47. P. Suriana. Fourier-Motzkin with non-linear symbolic constant coefficients. Master thesis, Massachusetts Institute of Technology, 2016.
48. P. Van Hentenryck, A. Cortesi, and B. Le Charlier. Type analysis of Prolog using type graphs. *The Journal of Logic Programming*, 22(3):179–209, 1995.
49. A. Venet. Automatic analysis of pointer aliasing for untyped programs. *Science of Computer Programming*, 35(2):223–248, 1999.
50. S. Villemot. Automates finis et intérpretation abstraite: Application à l'analyse statique de protocoles de communication. Rapport de DEA, École normale supérieure, 2002.
51. D. Wang, J. Hoffmann, and T. Reps. PMAF: An algebraic framework for static analysis of probabilistic programs. In *PLDI '18*, pages 513–528. ACM, 2018.

A Proof of Theorem 2

Let $\rho \in \{\, \rho \mid (-, \rho, -) \in \gamma(I^\sharp) \,\}$ be a concrete initial environment. We divide the proof into two parts.

A.1 Proof of the Weak Markov Property

First, we want to prove that:

$$\forall s_i^\sharp \in \Sigma^\sharp : \mathbb{E}_{\mathcal{E}} \left[\!\!\left[\boldsymbol{\pi}^\sharp [\![\mathsf{P}]\!](I^\sharp)(s_i^\sharp) \right]\!\!\right] \rho \leq \sum_{s_j^\sharp \in \Sigma^\sharp} \mathbb{E}_{\mathcal{E}} \left[\!\!\left[\boldsymbol{\pi}^\sharp [\![\mathsf{P}]\!](I^\sharp)(s_j^\sharp) \times \mathbb{P}^\sharp [\![\mathsf{P}]\!](I^\sharp)(s_j^\sharp, s_i^\sharp) \right]\!\!\right] \rho$$

Let $s_i^\sharp \in \Sigma^\sharp$ be an abstract state of the final abstract Markov chain $\mathbb{M}^\sharp [\![\mathsf{P}]\!] I^\sharp$. We start by linking the evaluation of the abstract distribution $\boldsymbol{\pi}^\sharp [\![\mathsf{P}]\!](I^\sharp)(s_i^\sharp)$ to the value of the concrete one $\boldsymbol{\pi} [\![\mathsf{P}]\!] \rho(s_i)$:

$$\begin{aligned}
\mathbb{E}_{\mathcal{E}} \left[\!\!\left[\boldsymbol{\pi}^\sharp [\![\mathsf{P}]\!](I^\sharp)(s_i^\sharp) \right]\!\!\right] \rho &= \sum_{s_i \in \gamma_\Sigma(s_i^\sharp)} \boldsymbol{\pi} [\![\mathsf{P}]\!](\rho)(s_i) && \text{(Defintion 14)} \\
&= \sum_{\substack{s_j^\sharp \in \Sigma^\sharp \\ s_i \in \gamma_\Sigma(s_i^\sharp) \\ s_j \in \gamma_\Sigma(s_j^\sharp)}} \boldsymbol{\pi} [\![\mathsf{P}]\!](\rho)(s_j) \times \mathbb{P} [\![\mathsf{P}]\!](\rho)(s_j, s_i) && \text{(Using (2))} \\
&= \sum_{\substack{s_j^\sharp \in \Sigma^\sharp \\ s_i \in \gamma_\Sigma(s_i^\sharp) \\ s_j \in \gamma_\Sigma(s_j^\sharp)}} \frac{\nu_i}{\nu_j} \boldsymbol{\pi} [\![\mathsf{P}]\!](\rho)(s_j) \times \sum_{s_j \xrightarrow{\omega} s_i \in \mathbb{M}[\![\mathsf{P}]\!] \rho} \Pr [\![\omega]\!] \rho && \text{(Definition 6)}
\end{aligned}$$

where ν_i and ν_j are the sojourn times at states s_i and s_j respectively.

By exploiting the over-approximation of these sojourn times provided by the abstract numeric domain, we can infer the following:

$$\begin{aligned}
&\mathbb{E}_{\mathcal{E}} \left[\!\!\left[\boldsymbol{\pi}^\sharp [\![\mathsf{P}]\!](I^\sharp)(s_i^\sharp) \right]\!\!\right] \rho \leq \\
&\quad \sum_{s_j^\sharp \in \Sigma^\sharp} \frac{\max^\sharp [\![\nu]\!](s_i^\sharp)}{\min^\sharp [\![\nu]\!](s_j^\sharp)} \sum_{s_j \in \gamma_\Sigma(s_j^\sharp)} \boldsymbol{\pi} [\![\mathsf{P}]\!](\rho)(s_j) \times \sum_{\substack{s_i \in \gamma_\Sigma(s_i^\sharp) \\ s_j \xrightarrow{\omega} s_i \in \mathbb{M}[\![\mathsf{P}]\!] \rho}} \Pr [\![\omega]\!] \rho
\end{aligned}$$

Using the soundness condition guaranteed by Theorem 1 we can derive:

$$\begin{aligned}
&\mathbb{E}_{\mathcal{E}} \left[\!\!\left[\boldsymbol{\pi}^\sharp [\![\mathsf{P}]\!](I^\sharp)(s_i^\sharp) \right]\!\!\right] \rho \leq \\
&\quad \sum_{s_j^\sharp \in \Sigma^\sharp} \frac{\max^\sharp [\![\nu]\!](s_i^\sharp)}{\min^\sharp [\![\nu]\!](s_j^\sharp)} \sum_{\substack{s_j^\sharp \xrightarrow{\omega^\sharp} s_j^\sharp \in \mathbb{M}^\sharp [\![\mathsf{P}]\!] I^\sharp \\ s_j \in \gamma_\Sigma(s_j^\sharp)}} \boldsymbol{\pi} [\![\mathsf{P}]\!](\rho)(s_j) \times \sum_{\substack{s_i \in \gamma_\Sigma(s_i^\sharp) \\ \omega \in \gamma^{\mathbf{L}}_{\mathscr{A}}(\omega^\sharp) \\ s_j \xrightarrow{\omega} s_i \in \mathbb{M}[\![\mathsf{P}]\!] \rho}} \Pr [\![\omega]\!] \rho \qquad (4)
\end{aligned}$$

For the next step, we need to over-approximate the concrete transition probabilities $\Pr [\![\omega]\!] \rho$, which is done using the following lemma:

Lemma 1 *Let $s_j \in \Sigma$ a concrete state and $\omega^\sharp \in \Omega^\sharp$ an abstract scenario. We have:*

$$\sum_{\substack{s_i \in \gamma_\Sigma(s_i^\sharp) \\ \omega \in \gamma^{\mathbf{L}}_{\mathscr{A}}(\omega^\sharp) \\ s_j \xrightarrow{\omega} s_i \in \mathbb{M}[\![\mathsf{P}]\!] \rho}} \Pr [\![\omega]\!] \rho \leq \mathbb{E}_{\mathcal{E}} \left[\!\!\left[\Pr^\sharp \left[\!\!\left[\omega^\sharp \right]\!\!\right] \right]\!\!\right] \rho$$

Proof We proceed by induction on the structure of $\omega^\sharp$ using Definition 12 of abstract scenarios probabilities. An important notice that allows building this proof is that, starting from a concrete state s_j, outgoing transitions could be labeled only by outcomes of the *same distribution*. Indeed, because we are dealing with transitions from only one discrete time Markov chain $\mathbb{M}[\![\,\mathsf{P}\,]\!]\rho$, we can have only a pure probabilistic behavior with no non-determinism and with a single distribution at each state. So, we have the following cases:

– *Case* $\omega^\sharp = \varepsilon^\sharp$:

$$\begin{aligned}
\sum_{\substack{s_i \in \gamma_\Sigma(s_i^\sharp) \\ \omega \in \gamma^{\mathbf{L}}_{\mathscr{A}}(\varepsilon^\sharp) \\ s_j \xrightarrow{\omega} s_i \in \mathbb{M}[\![\,\mathsf{P}\,]\!]\rho}} \Pr[\![\,\omega\,]\!]\rho &= \sum_{\substack{s_i \in \gamma_\Sigma(s_i^\sharp) \\ s_j \xrightarrow{\varepsilon} s_i \in \mathbb{M}[\![\,\mathsf{P}\,]\!]\rho}} \Pr[\![\,\varepsilon\,]\!]\rho \\
&\leq \Pr[\![\,\varepsilon\,]\!]\rho && \text{(Since } \varepsilon \text{ is the only possible outcome)} \\
&= 1 && \text{(Definition 5)} \\
&= \mathbb{E}_{\mathcal{E}}[\![\,1\,]\!]\rho \\
&= \mathbb{E}_{\mathcal{E}}\left[\!\!\left[\,\Pr^\sharp\left[\!\!\left[\,\varepsilon^\sharp\,\right]\!\!\right]\,\right]\!\!\right]\rho && \text{(Defintion 12)}
\end{aligned}$$

– *Case* $\omega^\sharp = \mathbf{b}_l$:

$$\begin{aligned}
\sum_{\substack{s_i \in \gamma_\Sigma(s_i^\sharp) \\ \omega \in \gamma^{\mathbf{L}}_{\mathscr{A}}(\mathbf{b}_l) \\ s_j \xrightarrow{\omega} s_i \in \mathbb{M}[\![\,\mathsf{P}\,]\!]\rho}} \Pr[\![\,\omega\,]\!]\rho &= \sum_{\substack{s_i \in \gamma_\Sigma(s_i^\sharp) \\ s_j \xrightarrow{b_l} s_i \in \mathbb{M}[\![\,\mathsf{P}\,]\!]\rho}} \Pr[\![\,b_l\,]\!] \\
&\leq \Pr[\![\,b_l\,]\!]\rho \\
&= \rho[p_l] \\
&= \mathbb{E}_{\mathcal{E}}[\![\,p_l\,]\!]\rho \\
&= \mathbb{E}_{\mathcal{E}}\left[\!\!\left[\,\Pr^\sharp[\![\,\mathbf{b}_l\,]\!]\,\right]\!\!\right]\rho
\end{aligned}$$

– *Case* $\omega^\sharp = \overline{\mathbf{b}}_l$:

$$\begin{aligned}
\sum_{\substack{s_i \in \gamma_\Sigma(s_i^\sharp) \\ \omega \in \gamma^{\mathbf{L}}_{\mathscr{A}}(\overline{\mathbf{b}}_l) \\ s_j \xrightarrow{\omega} s_i \in \mathbb{M}[\![\,\mathsf{P}\,]\!]\rho}} \Pr[\![\,\omega\,]\!]\rho &= \sum_{\substack{s_i \in \gamma_\Sigma(s_i^\sharp) \\ s_j \xrightarrow{\overline{b}_l} s_i \in \mathbb{M}[\![\,\mathsf{P}\,]\!]\rho}} \Pr\left[\!\!\left[\,\overline{b}_l\,\right]\!\!\right]\rho \\
&\leq \Pr\left[\!\!\left[\,\overline{b}_l\,\right]\!\!\right]\rho \\
&= 1 - \rho[p_l] \\
&= \mathbb{E}_{\mathcal{E}}[\![\,1 - p_l\,]\!]\rho \\
&= \mathbb{E}_{\mathcal{E}}\left[\!\!\left[\,\Pr^\sharp\left[\!\!\left[\,\overline{\mathbf{b}}_l\,\right]\!\!\right]\,\right]\!\!\right]\rho
\end{aligned}$$

– *Case* $\omega^\sharp = \mathbf{u}_l^i$:

$$\begin{aligned}
\sum_{\substack{s_i \in \gamma_\Sigma(s_i^\sharp) \\ \omega \in \gamma^{\mathbf{L}}_{\mathscr{A}}(\mathbf{u}_l^i) \\ s_j \xrightarrow{\omega} s_i \in \mathbb{M}[\![\,\mathsf{P}\,]\!]\rho}} \Pr[\![\,\omega\,]\!]\rho &= \sum_{\substack{s_i \in \gamma_\Sigma(s_i^\sharp) \\ s_j \xrightarrow{u_l^{i,a,b}} s_i \in \mathbb{M}[\![\,\mathsf{P}\,]\!]\rho}} \Pr\left[\!\!\left[\, u_l^{i,a,b} \,\right]\!\!\right]\rho \\
&\leq \Pr\left[\!\!\left[\, u_l^{i,a,b} \,\right]\!\!\right]\rho \\
&= \frac{1}{b-a+1} \\
&\leq \mathbb{E}_{\mathcal{E}}\left[\!\!\left[\frac{1}{\min_\star^\sharp\left[\!\!\left[\, \overrightarrow{\mathtt{u}}_l \,\right]\!\!\right] - \max_\star^\sharp\left[\!\!\left[\, \overleftarrow{\mathtt{u}} \,\right]\!\!\right] + 1} \right]\!\!\right]\rho \\
&= \mathbb{E}_{\mathcal{E}}\left[\!\!\left[\Pr^\sharp[\![\,\mathbf{u}_l^i\,]\!] \right]\!\!\right]\rho
\end{aligned}$$

– *Case* $\omega^\sharp = \mathbf{u}_l^\star$:

$$\begin{aligned}
\sum_{\substack{s_i \in \gamma_\Sigma(s_i^\sharp) \\ \omega \in \gamma^{\mathbf{L}}_{\mathscr{A}}(\mathbf{u}_l^\star) \\ s_j \xrightarrow{\omega} s_i \in \mathbb{M}[\![\,\mathsf{P}\,]\!]\rho}} \Pr[\![\,\psi\,]\!](\omega) &= \sum_{\substack{s_i \in \gamma_\Sigma(s_i^\sharp) \\ i \in [U, b-a+1] \\ s_j \xrightarrow{u_l^{i,a,b}} s_i \in \mathbb{M}[\![\,\mathsf{P}\,]\!]\rho}} \Pr\left[\!\!\left[\, u_l^{i,a,b} \,\right]\!\!\right]\rho \\
&\leq \sum_{i \in [U, b-a+1]} \frac{1}{b-a+1} \\
&= \max\left(0, \frac{b-a+1-U+1}{b-a+1}\right) \\
&= \max\left(0, \frac{b-a+2-U}{b-a+1}\right) \\
&\leq \mathbb{E}_{\mathcal{E}}\left[\!\!\left[\max\left(0, \frac{\max_\star^\sharp\left[\!\!\left[\, \overrightarrow{\mathtt{u}}_l \,\right]\!\!\right] - \min_\star^\sharp\left[\!\!\left[\, \overleftarrow{\mathtt{u}}_l \,\right]\!\!\right] + 2 - U}{\min_\star^\sharp\left[\!\!\left[\, \overrightarrow{\mathtt{u}}_l \,\right]\!\!\right] - \max_\star^\sharp\left[\!\!\left[\, \overleftarrow{\mathtt{u}}_l \,\right]\!\!\right] + 1}\right) \right]\!\!\right]\rho \\
&= \mathbb{E}_{\mathcal{E}}\left[\!\!\left[\Pr^\sharp\left[\!\!\left[\, \mathbf{u}_l^\star \,\right]\!\!\right] \right]\!\!\right]\rho
\end{aligned}$$

– *Case* $\omega^\sharp = \omega_0^\sharp \xi^\sharp$:

$$\begin{aligned}
\sum_{\substack{s_i \in \gamma_\Sigma(s_i^\sharp) \\ \omega \in \gamma^{\mathbf{L}}_{\mathscr{A}}(\omega_0^\sharp \xi^\sharp) \\ s_j \xrightarrow{\omega} s_i \in \mathbb{M}[\![\,\mathsf{P}\,]\!]\rho}} \Pr[\![\,\omega\,]\!]\rho &= \sum_{\substack{s_i \in \gamma_\Sigma(s_i^\sharp) \\ \omega_0 \in \gamma^{\mathbf{L}}_{\mathscr{A}}(\omega_0^\sharp) \\ \xi \in \gamma^{\mathbf{L}}_{\mathscr{A}}(\xi^\sharp) \\ s_j \xrightarrow{\omega_0 \xi} s_i \in \mathbb{M}[\![\,\mathsf{P}\,]\!]\rho}} \Pr[\![\,\omega_0\,]\!]\rho \Pr[\![\,\xi\,]\!]\rho \\
&\leq \sum_{\substack{s_k \in \gamma_\Sigma(s_k^\sharp) \\ \omega_0 \in \gamma^{\mathbf{L}}_{\mathscr{A}}(\omega_0^\sharp) \\ s_j \xrightarrow{\omega_0} s_k \in \mathbb{M}[\![\,\mathsf{P}\,]\!]\rho}} \Pr[\![\,\omega_0\,]\!]\rho \sum_{\substack{s_i \in \gamma_\Sigma(s_i^\sharp) \\ \xi \in \gamma^{\mathbf{L}}_{\mathscr{A}}(\xi^\sharp) \\ s_k \xrightarrow{\xi} s_i \in \mathbb{M}[\![\,\mathsf{P}\,]\!]\rho}} \Pr[\![\,\xi\,]\!]\rho \\
&\leq \mathbb{E}_{\mathcal{E}}\left[\!\!\left[\Pr^\sharp\left[\!\!\left[\, \omega_0^\sharp \,\right]\!\!\right] \right]\!\!\right]\rho \times \mathbb{E}_{\mathcal{E}}\left[\!\!\left[\Pr^\sharp\left[\!\!\left[\, \xi^\sharp \,\right]\!\!\right] \right]\!\!\right]\rho \qquad \text{⟅\textit{Induction hypothesis}⟆} \\
&= \mathbb{E}_{\mathcal{E}}\left[\!\!\left[\Pr^\sharp\left[\!\!\left[\, \omega_0^\sharp \,\right]\!\!\right] \times \Pr^\sharp\left[\!\!\left[\, \xi^\sharp \,\right]\!\!\right] \right]\!\!\right]\rho \\
&= \mathbb{E}_{\mathcal{E}}\left[\!\!\left[\Pr^\sharp\left[\!\!\left[\, \omega_0^\sharp \xi^\sharp \,\right]\!\!\right] \right]\!\!\right]\rho
\end{aligned}$$

– *Case* $\omega^\sharp = {\omega^\sharp}_1 + {\omega^\sharp}_2$:

$$\sum_{\substack{s_i \in \gamma_\Sigma(s_i^\sharp) \\ \omega \in \gamma^{\mathbf{L}}_{\mathscr{A}}(\omega_1^\sharp + \omega_2^\sharp) \\ s_j \xrightarrow{\omega} s_i \in \mathbb{M}[\![\,\mathsf{P}\,]\!]\rho}} \Pr[\![\,\omega\,]\!]\rho = \sum_{\substack{s_i \in \gamma_\Sigma(s_i^\sharp) \\ \omega \in \gamma^{\mathbf{L}}_{\mathscr{A}}(\omega_1^\sharp) \cup \gamma^{\mathbf{L}}_{\mathscr{A}}(\omega_2^\sharp) \\ s_j \xrightarrow{\omega} s_i \in \mathbb{M}[\![\,\mathsf{P}\,]\!]\rho}} \Pr[\![\,\omega\,]\!]\rho$$

$$\leq \sum_{\substack{s_i \in \gamma_\Sigma(s_i^\sharp) \\ \omega_1 \in \gamma^{\mathbf{L}}_{\mathscr{A}}(\omega_1^\sharp) \\ s_j \xrightarrow{\omega_1} s_i \in \mathbb{M}[\![\,\mathsf{P}\,]\!]\rho}} \Pr[\![\,\omega_1\,]\!]\rho + \sum_{\substack{s_i \in \gamma_\Sigma(s_i^\sharp) \\ \omega_2 \in \gamma^{\mathbf{L}}_{\mathscr{A}}(\omega_2^\sharp) \\ s_j \xrightarrow{\omega_2} s_i \in \mathbb{M}[\![\,\mathsf{P}\,]\!]\rho}} \Pr[\![\,\omega_2\,]\!]\rho$$

$$\leq \mathbb{E}_{\mathcal{E}}\left[\!\left[\Pr^\sharp\left[\!\left[\omega_1^\sharp \right]\!\right] \right]\!\right]\rho + \mathbb{E}_{\mathcal{E}}\left[\!\left[\Pr^\sharp\left[\!\left[\omega_2^\sharp \right]\!\right] \right]\!\right]\rho$$

$$= \mathbb{E}_{\mathcal{E}}\left[\!\left[\Pr^\sharp\left[\!\left[\omega_1^\sharp \right]\!\right] + \Pr^\sharp\left[\!\left[\omega_2^\sharp \right]\!\right] \right]\!\right]\rho$$

$$= \mathbb{E}_{\mathcal{E}}\left[\!\left[\Pr^\sharp\left[\!\left[\omega_1^\sharp + \omega_2^\sharp \right]\!\right] \right]\!\right]\rho$$

□

Finally, using this lemma as an over-approximation of concrete scenario probabilities, we can derive from (4) the following:

$$\mathbb{E}_{\mathcal{E}}\left[\!\left[\boldsymbol{\pi}^\sharp[\![\,\mathsf{P}\,]\!](I^\sharp)(s_i^\sharp) \right]\!\right]\rho \leq \sum_{s_j^\sharp \in \Sigma^\sharp} \frac{\max^\sharp[\![\,\nu\,]\!](s_i^\sharp)}{\min^\sharp[\![\,\nu\,]\!](s_j^\sharp)} \sum_{\substack{s_j^\sharp \xrightarrow{\omega^\sharp} s_j^\sharp \in \mathbb{M}^\sharp[\![\,\mathsf{P}\,]\!]I^\sharp \\ s_j \in \gamma_\Sigma(s_j^\sharp)}} \boldsymbol{\pi}[\![\,\mathsf{P}\,]\!](\rho)(s_j) \times \sum_{\substack{s_i \in \gamma_\Sigma(s_i^\sharp) \\ \omega \in \gamma^{\mathbf{L}}_{\mathscr{A}}(\omega^\sharp) \\ s_j \xrightarrow{\omega} s_i \in \mathbb{M}[\![\,\mathsf{P}\,]\!]\rho}} \Pr[\![\,\omega\,]\!]\rho$$

$$\leq \sum_{s_j^\sharp \in \Sigma^\sharp} \frac{\max^\sharp[\![\,\nu\,]\!](s_i^\sharp)}{\min^\sharp[\![\,\nu\,]\!](s_j^\sharp)} \sum_{s_j^\sharp \xrightarrow{\omega^\sharp} s_j^\sharp \in \mathbb{M}^\sharp[\![\,\mathsf{P}\,]\!]I^\sharp} \mathbb{E}_{\mathcal{E}}\left[\!\left[\Pr^\sharp\left[\!\left[\omega^\sharp \right]\!\right] \right]\!\right]\rho \times \sum_{s_j \in \gamma_\Sigma(s_j^\sharp)} \boldsymbol{\pi}[\![\,\mathsf{P}\,]\!](\rho)(s_j)$$

⟅ *Definition 14* ⟆

$$= \sum_{s_j^\sharp \in \Sigma^\sharp} \frac{\max^\sharp[\![\,\nu\,]\!](s_i^\sharp)}{\min^\sharp[\![\,\nu\,]\!](s_j^\sharp)} \sum_{s_j^\sharp \xrightarrow{\omega^\sharp} s_j^\sharp \in \mathbb{M}^\sharp[\![\,\mathsf{P}\,]\!]I^\sharp} \mathbb{E}_{\mathcal{E}}\left[\!\left[\Pr^\sharp\left[\!\left[\omega^\sharp \right]\!\right] \right]\!\right]\rho \times \mathbb{E}_{\mathcal{E}}\left[\!\left[\boldsymbol{\pi}^\sharp[\![\,\mathsf{P}\,]\!](I^\sharp)(s_j^\sharp) \right]\!\right]\rho$$

$$= \sum_{s_j^\sharp \in \Sigma^\sharp} \mathbb{E}_{\mathcal{E}}\left[\!\left[\boldsymbol{\pi}^\sharp[\![\,\mathsf{P}\,]\!](I^\sharp)(s_j^\sharp) \right]\!\right]\rho \times \mathbb{E}_{\mathcal{E}}\left[\!\left[\frac{\max^\sharp[\![\,\nu\,]\!](s_i^\sharp)}{\min^\sharp[\![\,\nu\,]\!](s_j^\sharp)} \sum_{s_j^\sharp \xrightarrow{\omega^\sharp} s_j^\sharp \in \mathbb{M}^\sharp[\![\,\mathsf{P}\,]\!]I^\sharp} \Pr^\sharp\left[\!\left[\omega^\sharp \right]\!\right] \right]\!\right]\rho$$

⟅ *Defintion 13* ⟆

$$= \sum_{s_j^\sharp \in \Sigma^\sharp} \mathbb{E}_{\mathcal{E}}\left[\!\left[\boldsymbol{\pi}^\sharp[\![\,\mathsf{P}\,]\!](I^\sharp)(s_j^\sharp) \right]\!\right]\rho \times \mathbb{E}_{\mathcal{E}}\left[\!\left[\mathbb{P}^\sharp[\![\,\mathsf{P}\,]\!](I^\sharp)(s_j^\sharp, s_i^\sharp) \right]\!\right]\rho$$

$$= \sum_{s_j^\sharp \in \Sigma^\sharp} \mathbb{E}_{\mathcal{E}}\left[\!\left[\boldsymbol{\pi}^\sharp[\![\,\mathsf{P}\,]\!](I^\sharp)(s_j^\sharp) \times \mathbb{P}^\sharp[\![\,\mathsf{P}\,]\!](I^\sharp)(s_j^\sharp, s_i^\sharp) \right]\!\right]\rho$$

□

A.2 Normalization Constraint

The second part of the theorem is the normalization constraint:

$$\sum_{s^\sharp \in \Sigma^s} \mathbb{E}_{\mathcal{E}}\left[\!\left[\boldsymbol{\pi}^\sharp[\![\,\mathsf{P}\,]\!](I^\sharp)(s^\sharp) \right]\!\right]\rho = 1$$

We have:

$$\begin{aligned}\sum_{s^\sharp \in \Sigma^s} \mathbb{E}_{\mathcal{E}} \left[\!\!\left[\boldsymbol{\pi}^\sharp [\![\mathsf{P}]\!](I^\sharp)(s^\sharp) \right]\!\!\right] \rho &= \sum_{\substack{s^\sharp \in \Sigma^\sharp \\ s \in \gamma_\Sigma(s^\sharp)}} \boldsymbol{\pi} [\![\mathsf{P}]\!](\rho)(s) && \text{⟅Defintion (14)⟆} \\ &= \sum_{s \in \Sigma} \boldsymbol{\pi} [\![\mathsf{P}]\!](\rho)(s) && \text{⟅Spurious states have null probability⟆} \\ &= 1\end{aligned}$$

□

B Benchmarks Programs

(a) $\mathbf{C}_\infty$: Constant Unbounded Backoffs

```
while (1) {
  data = sense();
  while(1) {
    t = uniform(1, B);
    wait t;
    if(unicast(data)) {
      break;
    }
    wait S;
  }
}
```

(b) $\mathbf{C}_n$: Constant Bounded Backoffs

```
while (1) {
  data = sense();
  i = 0;
  while(i < n)
    {
      t = uniform(1, B);
      wait t;
      if(unicast(data)) {
        break;
      } else {
        i = i + 1;
      }
    }
  wait S;
}
```

(c) $\mathbf{L}_n$: Linear Bounded Backoffs

```
while (1) {
  data = sense();
  i = 0;
  while(i < n) {
    t = uniform(1, B + i);
    wait t;
    if(unicast(data)) {
      break;
    } else {
      i = i + 1;
    }
  }
  wait S;
}
```

(d) $\mathbf{L}_{t\infty}$: Linear Truncated Backoffs

```
while (1) {
  data = sense();
  i = 0;
  while(1) {
    b = uniform(1, B + i);
    wait b;
    if(unicast(data)) {
      break;
    }
    if (i < t - 1) {
      i = i + 1;
    }
  }
  wait S;
}
```

Fig. 17: Programs of the analyzed backoff mechanisms.