Reachability Analysis of Probabilistic Systems by Successive Refinements

Pedro R. D'Argenio[1]*, Bertrand Jeannet[2], Henrik E. Jensen[2], and Kim G. Larsen[2,1]

[1] Faculty of Informatics, University of Twente
P.O. Box 217. NL-7500 AE - Enschede. The Netherlands
`dargenio@cs.utwente.nl`
[2] BRICS - Aalborg University
Frederik Bajers vej 7-E. DK-9220 Aalborg. Denmark
`{bjeannet,ejersbo,kgl}@cs.auc.dk`

Abstract. We report on a novel development to model check quantitative reachability properties on Markov decision processes together with its prototype implementation. The innovation of the technique is that the analysis is performed on an abstraction of the model under analysis. Such an abstraction is significantly smaller than the original model and may safely refute or accept the required property. Otherwise, the abstraction is refined and the process repeated. As the numerical analysis necessary to determine the validity of the property is more costly than the refinement process, the technique profits from applying such numerical analysis on smaller state spaces.

1 Introduction

The verification of systems has nowadays reached a clear maturity. Fully automatic tools, in particular model checkers, have been developed and successfully used in industrial cases. A model checker is a tool that can answer whether the system under study satisfies some required property. Many times, however, these type of properties are not expressive enough to assert adequately the correctness of a system. Nevertheless, it is desirable that the probability of reaching the unavoidable error is small enough. *Quantitative model checking*, that is, model checking of probabilistic models with respect to *probabilistic* properties, has already been studied during the last decade [13,2,5,20,4, etc.]. However, it was not until recently that attention was drawn to efficient tool implementations. In this paper we report on a novel development to model check quantitative properties.

We use *Markov decision processes* (see e.g. [27]) to describe the system under study. This model, also called probabilistic transition system (PTS), allows to combine probabilistic and non-deterministic steps and is a natural extension to traditional non-deterministic models (such as labelled transition systems). Our preference for a probabilistic model that allows non-determinism is based on two

* Supported by the STW-PROGRESS project TES-4999

facts. First, PTSs are closed under parallel compostition which facilitates the modelling process. Second, PTSs are also closed under abstraction. This reason is fundamental as the method introduced in this paper is based on abstraction techniques.

We focus on a restricted set of reachability properties. They allow to specify that the probability to reach a particular final condition f from any state satisfying a given initial condition i is smaller (or greater) than a probability p. This type of properties is not so restrictive as it seems since we can always use checking automata to add additional constraints to the property.

The method we present is based on automatic abstraction and refinement techniques. The basic idea is to use abstraction to reduce the high cost of probabilistic analysis. The difficulty lies in finding the right abstraction level, depending on the property to prove. To address it, the method starts with a coarse abstraction of the system which is obtained by *partitioning* the state space, according to the property under study. The property is then checked on the obtained abstract model. The verdict may be inconclusive, that is, p happens to be between the calculated upper bound of the minimum and the lower bound of the maximum actual probabilities. In this case the previous abstraction is refined and the question posed again. The process is successively repeated until a satisfactory answer is given, or no further refinement is possible. To efficiently store the state space, perform abstractions and process the refinement steps, we use BDDs and MTBDDs (more precisely ADDs) [10,3]. The soundness of the method is asserted by considering a suitable probabilistic simulation [22,28] (which preserves the kind of property we consider), and by showing that abstraction by partitioning respects this simulation relation.

The contributions of this paper are first the definition of the probabilistic simulation relation that allows to prove the soundness of our method, and secondly, the design of efficient algorithms to abstract PTSs, to analyse and to refine them. Finally, experimental results shows the effectiveness of the method.

Related work. The partition refinement method we use on PTSs resorts to principles already applied to finite-state systems [6] and timed automata [1]. However our aim is not to generate a minimal model w.r.t. a bisimulation relation, but to steer the refinement process in order to prove as early as possible an intended (probabilistic) property.

The efficiency provided by MTBDDs to store and logically manipulate the state space made them also the choice of recent quantitative model checkers [14, 9]. However, if it comes to model analysis via numerical recipes like simplex or (iterative) solutions of equations systems, experience has shown that MTBDDs do not outperform classical data structures (such as sparse matrices) [3,18,9]. The main reason appears to be that any of these algorithms tend to require the storage of a distinct real number per actual state [16]. In our case, the use of MTBDDs is focus on the manipulation of probabilistic transition relations and its use in the abstraction techniques. After abstraction, the size of the problem submitted to numerical analysis becomes a significantly smaller issue.

Other quantitative model checkers have also been developed. The tool Prob-Verus [14] allows to check the validity of a PCTL formula [13] on a (discrete time) Markov chain. Therefore, models do not contain non-determinism. Instead, Prism [9] is a quantitative model checker for PCTL formulas on (discrete time) Markov decision processes, i.e., non-determinism is inherent to the model. Like Prism, we also do model checking on Markov decision processes, but we restrict to a particular kind of PCTL formula. For completeness reason, we also mention the quantitative model checker $\mathsf{E} \vdash \mathsf{MC}^2$ [17], which model checks probabilistic timed properties on continuous-time Markov chains.

Organization of the paper. Section 2 and Section 3 introduce the theoretical foundations of the implemented tool. The algorithms, data structure, and methodological techniques are explained in Sections 4 and 5. An example is reported in Section 6. Finally we present our conclusions and discuss further work. Proofs and further details are reported in [8].

2 Probabilistic Transition Systems

Probabilistic transition systems (PTS for short) generalise the well-known transition systems with probabilistic information. In a PTS, a transition does not lead to a single state but to a probability space whose sample space is a set of states. The model we define is widely used (see, e.g. [28,5,23].) and is also known as Markov decision processes [27]. We consider in addition a function that labels each state with a property assumed to be valid in this state.

Let $\mathsf{Distr}(\Omega)$ be the set of all discrete probability distributions over the sample space Ω. Let PF be a set of propositional formulas closed under $\wedge$ and $\neg$.

Definition 1. *A* probabilistic transition system *(PTS for short) is a structure* $\mathsf{T} = (S, \longrightarrow, f)$ *where* S *is a set of states,* $\longrightarrow\ \subseteq S \times \mathsf{Distr}(S)$ *is the transition relation, and* $f : S \to \mathsf{PF}$ *is a proposition assignment. We write* $s \longrightarrow \pi$ *if* $(s, \pi) \in \longrightarrow$*, and* $s \longrightarrow$ *if there is a* π *such that* $s \longrightarrow \pi$*; otherwise, we write* $s \not\longrightarrow$ *and call* s *a* sink state*. A PTS is said to be a* fully probabilistic transition system *(FPTS for short) if whenever* $s \longrightarrow \pi$ *and* $s \longrightarrow \rho$ *then* $\pi = \rho$*. It will be convenient to distinguish an initial state* $s_0 \in S$*. In this case we call the structure* (T, s_0)*, a* rooted (fully) probabilistic transition system*. A proposition* $g \in \mathsf{PF}$ *is satisfied in state* s*, notation* $s \models g$*, whenever* $f(s) \Rightarrow g$ *holds is a tautology.*

Example 1. Consider a system that either increments a counter with probability 0.5 or it deadlocks with probability 0.5 while the counter is smaller than 20. Formally, it can be modelled by a PTS $\mathsf{Counter} = (S, \longrightarrow, f)$ where $S = \{\mathsf{a}, \mathsf{b}\} \times \{0 \,..\, 20\}$, $f(s, i) = (s \wedge x = i)$, and

$$(\mathsf{a}, 20) \longrightarrow \{(\mathsf{a}, 20) \mapsto 1\}$$
$$(\mathsf{a}, i) \longrightarrow \{(\mathsf{a}, i+1) \mapsto 0.5, (\mathsf{b}, i) \mapsto 0.5\} \quad \textbf{if } i < 20$$

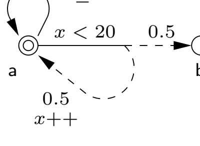


Fig. 1.

A symbolic representation of this PTS is depicted in Fig. 1. □

Let $\mathsf{T} = (S, \longrightarrow, f)$. A *simple path starting from* $s_0 \in S$ in T is a finite sequence of S-states, $\sigma = s_0 s_1 s_2 \ldots s_n$, where for each $0 \leq i < n$ there exists $\pi_i \in \mathsf{Distr}(S)$ such that $s_i \longrightarrow \pi_i$ and $\pi_i(s_{i+1}) > 0$. Let $\sigma(i)$ denote the state in the i-th position. Let $|\sigma|$ be the length of σ. Let $\mathit{first}(\sigma) = \sigma(1)$ and $\mathit{last}(\sigma) = \sigma(|\sigma|)$. We let $\mathit{s\text{-}paths}(\mathsf{T})$ denote the sets of simple paths in T starting from any $s \in S$. A state t is *reachable* from other state s in T if there is $\sigma \in \mathit{s\text{-}paths}(\mathsf{T})$ with $s = \mathit{first}(\sigma)$ and $t = \mathit{last}(\sigma)$. Let $\mathit{reach}(\mathsf{T}, s)$ denote the set of all states reachable from s in T.

For any rooted FPTS (F, s), the *probability measure* $\mathsf{P}_{\mathsf{F},s}$ on the σ-algebra induced by (F, s) is the unique probability measure defined such that $\mathsf{P}_{\mathsf{F},s}(\sigma) =$ **if** $(s = s_0)$ **then** $\pi_0(s_1) \cdot \pi_1(s_2) \cdot \ldots \cdot \pi_{n-1}(s_n)$ **else** 0. In particular, $\mathsf{P}_{\mathsf{F},s}(\sigma)$ is the probability of σ in F starting from s

Any given PTS T defines a set of *probabilistic executions*, each one obtained by iteratively scheduling one of the possible post-state distributions from each pre-state, starting from a given state $s_0 \in S$. Notice that the same state s of T may occur more than once during a probabilistic execution and each time a different distribution from s may be scheduled. In order to distinguish such occurrences every state s of a probabilistic execution is extended with the past history of s, that is, with the unique path leading from the start state to s.

Definition 2. *A* probabilistic path *of* T *is a FPTS* $\mathsf{F} = (\mathit{s\text{-}paths}(\mathsf{T}), \longrightarrow_{\mathsf{F}}, f \circ \mathit{last})$ *where* $q \longrightarrow_{\mathsf{F}} \rho$ *implies* $\mathit{last}(q) \longrightarrow_{\mathsf{T}} \pi$ *with* $\rho(qs) = \pi(s)$ *for all* $s \in S$. *If in addition, for all* $q \in \mathit{s\text{-}paths}(\mathsf{T})$ *such that* $|q| < i$, $\mathit{last}(q) \longrightarrow_{\mathsf{T}}$ *implies that* $q \longrightarrow_{\mathsf{F}}$, *then the rooted FPTS* (F, s_0) *is said to be a* probabilistic execution fragment of length i of T starting from $s_0 \in S$. *If* $i = \infty$ *then* (F, s_0) *is said to be a* probabilistic execution of T starting from $s_0 \in S$.

Denote by $\mathit{paths}(\mathsf{T})$ the set of all probabilistic paths of T, by $\mathit{execs}(\mathsf{T}, s_0, i)$ the set of all probabilistic execution fragments of length i starting from s_0, and by $\mathit{execs}(\mathsf{T}, s_0)$ the set of all probabilistic executions of T starting from s_0.

Given a simple path $\sigma \in \mathit{s\text{-}paths}(\mathsf{T})$ define $\sigma^{\uparrow} \in \mathit{s\text{-}paths}(\mathsf{F})$ (F being a probabilistic path of T) such that $|\sigma^{\uparrow}| = |\sigma|$ and for all $0 < i \leq |\sigma|$, $\sigma^{\uparrow}(i) = \sigma(1) \ldots \sigma(i)$. We extend $^{\uparrow}$ to sets of simple paths in the usual way. Let $\mathsf{f} \in \mathsf{PF}$ and define $\Sigma_{\mathsf{f}} \triangleq \{\sigma \in \mathit{s\text{-}paths}(\mathsf{T}) \mid \mathit{last}(\sigma) \models \mathsf{f}$ and $\forall 0 < i < |\sigma|.\ \sigma(i) \models \neg\mathsf{f}\}$, i.e., Σ_{f} is the set of all minimal paths in T that end in final condition f. The *minimum* and *maximum probabilities* of reaching a final condition $\mathsf{f} \in \mathsf{PF}$ from an initial condition $\mathsf{i} \in \mathsf{PF}$ in a rooted PTS (T, s_0) are defined respectively by

$$\mathsf{P}^{\inf}_{\mathsf{T},s_0}(\mathsf{i},\mathsf{f}) \triangleq \inf\left\{\mathsf{P}_{\mathsf{F},q}(\Sigma_{\mathsf{f}}^{\uparrow}) \mid s \in \mathit{reach}(\mathsf{T}, s_0),\ s \models \mathsf{i},\ \text{and } (\mathsf{F}, q) \in \mathit{execs}(\mathsf{T}, s)\right\}$$

$$\mathsf{P}^{\sup}_{\mathsf{T},s_0}(\mathsf{i},\mathsf{f}) \triangleq \sup\left\{\mathsf{P}_{\mathsf{F},q}(\Sigma_{\mathsf{f}}^{\uparrow}) \mid s \in \mathit{reach}(\mathsf{T}, s_0),\ s \models \mathsf{i},\ \text{and } (\mathsf{F}, q) \in \mathit{execs}(\mathsf{T}, s)\right\}$$

Example 2. Consider the $\mathsf{Counter}$ of Example 1. Take the initial condition $\mathsf{i} = (\mathsf{a} \wedge x = 0)$ and the final condition $\mathsf{f} = (\mathsf{b} \wedge x \geq 15)$. The reader is invited to check that $\Sigma_{\mathsf{f}} = \{(a,j)(a,j+1)\ldots(a,i-1)(a,i)(b,i) \mid 15 \leq i < 20 \wedge 0 \leq j \leq i\}$ and that $\mathsf{P}^{\inf}_{\mathsf{T},(a,0)}(\mathsf{i},\mathsf{f}) = \mathsf{P}^{\sup}_{\mathsf{T},(a,0)}(\mathsf{i},\mathsf{f}) = \frac{31}{2^{20}}$. □

3 Probabilistic Simulation

Probabilistic simulation [22,28] will be central to state the correctness of the technique proposed in this paper.

Definition 3. *Let $C \subseteq S \times S$ be a relation on states defining a* discrimination criterion. *R is a C-(probabilistic) simulation if, whenever sRt,*

1. *$(s,t) \in C$, and*
2. *if $s \longrightarrow \pi$, there exist ρ such that $t \longrightarrow \rho$ and $\pi \sqsubseteq_R \rho$.*

where $\pi \sqsubseteq_R \rho$ if there is $\delta \in \mathsf{Distr}(S \times S)$ such that for all $s,t \in S$, (i) $\pi(s) = \delta(s,S)$, (ii) $\rho(t) = \delta(S,t)$, and (iii) $\delta(s,t) > 0 \implies sRt$. s is C-simulated *by t, notation $s \leq_C t$, if there is a C-simulation R with sRt.*

Notice that whenever the discriminating criteria C is a preorder, so is $\leq_C$.

Our interest is to check when a PTS reaches a goal f starting from any state satisfying some initial condition i ($\mathsf{i}, \mathsf{f} \in \mathsf{PF}$). Consider the discriminating condition $(s,t) \in \mathsf{C}_{\mathsf{i},\mathsf{f}}$ defined by ($s \models \mathsf{f} \implies t \models \mathsf{f}$) and ($s \models \mathsf{i} \iff t \models \mathsf{i}$) We write only C whenever i and f are clear from the context. Simulations $\leq_{\mathsf{C}}$, $\leq_{\mathsf{C}^{-1}}$, and $\leq_{\mathsf{C} \cap \mathsf{C}^{-1}}$ are the relations needed to prove correctness of the technique.

We are interested in whether the probability of reaching a particular final condition f from any (reachable) state satisfying a given initial condition i is smaller or greater than a given value p. The next theorem states that if a PTS T_1 satisfies this property, and another T_2 ($\mathsf{C} \cap \mathsf{C}^{-1}$)-simulates T_1, then T_2 also satisfies the property.

Theorem 1. *Let (T_1, s_0^1) and (T_2, s_0^2) be two rooted PTSs such that none of them contains a sink state. Then*

1. $(\mathsf{T}_1, s_0^1) \leq_{\mathsf{C}} (\mathsf{T}_2, s_0^2)$ *implies* $\mathsf{P}^{\sup}_{\mathsf{T}_1, s_0^1}(\mathsf{i}, \mathsf{f}) \leq \mathsf{P}^{\sup}_{\mathsf{T}_2, s_0^2}(\mathsf{i}, \mathsf{f})$.
2. $(\mathsf{T}_1, s_0^1) \leq_{\mathsf{C}^{-1}} (\mathsf{T}_2, s_0^2)$ *implies* $\mathsf{P}^{\inf}_{\mathsf{T}_1, s_0^1}(\mathsf{i}, \mathsf{f}) \geq \mathsf{P}^{\inf}_{\mathsf{T}_2, s_0^2}(\mathsf{i}, \mathsf{f})$.
3. $(\mathsf{T}_1, s_0^1) \leq_{\mathsf{C} \cap \mathsf{C}^{-1}} (\mathsf{T}_2, s_0^2)$ *implies* $\mathsf{P}^{\sup}_{\mathsf{T}_1, s_0^1}(\mathsf{i}, \mathsf{f}) \leq \mathsf{P}^{\sup}_{\mathsf{T}_2, s_0^2}(\mathsf{i}, \mathsf{f})$ *and* $\mathsf{P}^{\inf}_{\mathsf{T}_1, s_0^1}(\mathsf{i}, \mathsf{f}) \geq \mathsf{P}^{\inf}_{\mathsf{T}_2, s_0^2}(\mathsf{i}, \mathsf{f})$.

The requirement that every state has a transition is not really harmful as each sink state can always be completed with a self-looping transition without affecting the properties of the original PTS.

The proposed technique is based on successive refinements of a coarse abstraction of the original PTS T. Each refinement is an abstraction of the next (finer) refinement in which T is the finest one. In the following, we state that the refinement operation preserves simulation. A consequence of Theorem 1 is that if a given abstraction satisfy the desired reachability property, so does T.

Definition 4. *Let $\mathcal{A} = (A_i)_I$ be a partition of S, i.e., for all $i,j \in I$, $A_i \cap A_j \neq \emptyset \iff i = j$, and $\bigcup_I A_i = S$. Let $\mathsf{T} = (S, \longrightarrow, f)$. The* quotient PTS according *to $\mathcal{A}$ is defined by $\mathsf{T}/\mathcal{A} = (\mathcal{A}, \longrightarrow_{\mathcal{A}}, f_{\mathcal{A}})$, where*

1. *$A \longrightarrow_{\mathcal{A}} (\pi/\mathcal{A})$ if $\exists s \in A.\ s \longrightarrow \pi$ and $\forall A' \in \mathcal{A}.\ (\pi/\mathcal{A})(A') \triangleq \sum_{s' \in A'} \pi(s')$,*
2. *$f_{\mathcal{A}}(A) \triangleq \bigwedge_{s \in A} f(s)$.*

For a rooted PTS (T, s_0), its quotient is given by $(\mathsf{T}, s_0)/\mathcal{A} \triangleq (\mathsf{T}/\mathcal{A}, A)$ provided $s_0 \in A \in \mathcal{A}$.

We say that $\mathsf{T}/\mathcal{A}$ is a C-*abstraction* of T if for all $A \in \mathcal{A}$, and $s, t \in A$, $(s \models \mathsf{i} \iff t \models \mathsf{i})$. We say that $\mathsf{T}/\mathcal{A}$ is a $(\mathsf{C} \cap \mathsf{C}^{-1})$-*abstraction* of T if for all $A \in \mathcal{A}$, and $s, t \in A$, $(s \models \mathsf{f} \iff t \models \mathsf{f})$, and $(s \models \mathsf{i} \iff t \models \mathsf{i})$.

Theorem 2. *Let (T, s_0) be a rooted PTS. Let $\mathcal{B}$ be a refinement of $\mathcal{A}$ (i.e., $\mathcal{B}$ is also a partition of S such that $\forall B \in \mathcal{B}.\ \exists A \in \mathcal{A}.\ B \subseteq A$).*

1. *if $\mathsf{T}/\mathcal{A}$ is a C-abstraction of T then (a) $(\mathsf{T}, s_0)/\mathcal{B} \leq_{\mathsf{C}} (\mathsf{T}, s_0)/\mathcal{A}$, and (b) $\mathsf{T}/\mathcal{B}$ is also a C-abstraction of T.*
2. *if $\mathsf{T}/\mathcal{A}$ is a $(\mathsf{C} \cap \mathsf{C}^{-1})$-abstraction of T then (a) $(\mathsf{T}, s_0)/\mathcal{B} \leq_{\mathsf{C} \cap \mathsf{C}^{-1}} (\mathsf{T}, s_0)/\mathcal{A}$, and (b) $\mathsf{T}/\mathcal{B}$ is also a $(\mathsf{C} \cap \mathsf{C}^{-1})$-abstraction of T.*

Example 3. Let

$$\mathcal{A} = \big\{\{(\mathsf{a}, 0)\}, \{(\mathsf{a}, i) \mid 1 \leq i < 15\}, \{(\mathsf{a}, i) \mid 15 \leq i < 20\}, \{(\mathsf{a}, 20)\},$$
$$\{(\mathsf{b}, i) \mid 0 \leq i < 15\}, \{(\mathsf{b}, i) \mid 15 \leq i < 20\}, \{(\mathsf{b}, 20)\}\big\}$$

Then $\mathsf{Counter}/\mathcal{A} = (\mathcal{A}, \longrightarrow_{\mathcal{A}}, f_{\mathcal{A}})$, with $\longrightarrow_{\mathcal{A}}$ defined by (see also Fig. 2)

$$[(\mathsf{a},0)] \longrightarrow_{\mathcal{A}} \{[(\mathsf{a},1)] \mapsto 0.5, [(\mathsf{b},0)] \mapsto 0.5\}$$
$$[(\mathsf{a},1)] \longrightarrow_{\mathcal{A}} \{[(\mathsf{a},1)] \mapsto 0.5, [(\mathsf{b},0)] \mapsto 0.5\}$$
$$[(\mathsf{a},1)] \longrightarrow_{\mathcal{A}} \{[(\mathsf{a},15)] \mapsto 0.5, [(\mathsf{b},1)] \mapsto 0.5\}$$
$$[(\mathsf{a},15)] \longrightarrow_{\mathcal{A}} \{[(\mathsf{a},15)] \mapsto 0.5, [(\mathsf{b},15)] \mapsto 0.5\}$$
$$[(\mathsf{a},15)] \longrightarrow_{\mathcal{A}} \{[(\mathsf{a},20)] \mapsto 0.5, [(\mathsf{b},15)] \mapsto 0.5\}$$
$$[(\mathsf{a},20)] \longrightarrow_{\mathcal{A}} \{[(\mathsf{a},20)] \mapsto 1\}$$

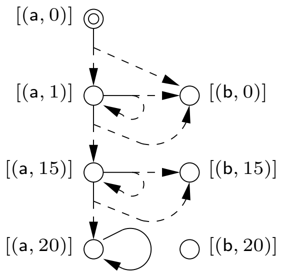


Fig. 2.

where $[s]$ denotes the class of s, i.e., the set in $\mathcal{A}$ such that $s \in [s]$. Notice that $\mathsf{Counter}/\mathcal{A}$ is indeed a $(\mathsf{C} \cap \mathsf{C}^{-1})$-abstraction of $\mathsf{Counter}$ with i and f as before. In addition, $\mathsf{P}^{\inf}_{\mathsf{T},[(a,0)]}(\mathsf{i},\mathsf{f}) = 0$ and $\mathsf{P}^{\sup}_{\mathsf{T},[(a,0)]}(\mathsf{i},\mathsf{f}) = \frac{1}{4}$, which is a sound approximation of the actual solution (see Example 2). $\square$

4 Model-Checking and Partitioning

To perform model checking, the require the PTS under study to be finite. In order to describe the encoding of PTS we need some particular notation. If $s \longrightarrow \pi$, we call the pair (s, π) a *nail*. Nails have the same functionality as *probabilistic states* in alternating models; they are depicted with black boxes in

Fig. 3. Let $\mathsf{T} = (S, \longrightarrow, f)$ be a PTS. From now on we assume the initial and final conditions i and f are atomic propositions and for all $s \in S$, $f(s)$ is either *true*, i, f or $\mathsf{i} \wedge \mathsf{f}$. T will also be described in an equivalent manner by a tuple $(S, N, \mathit{Org}, \tau, f)$ where N is the set of nails, $\mathit{Org} : N \to S$ associates to each nail (s, π) its *origin state* s, and $\tau : N \to \mathsf{Distr}(S)$ associates its distribution π. Let $\mathcal{N}(s) = \{n \in N \mid \mathit{Org}(n) = s\}$ be the set of *outgoing nails* of state s. Notice that nails having different origin states can share the same distribution.

Computing $\mathsf{P}^{\inf}(\mathsf{i})$ and $\mathsf{P}^{\sup}(\mathsf{i})$. For the rest of this section we wil use the shorthand $\mathsf{P}^{\inf}(s)$ and $\mathsf{P}^{\sup}(s)$ for $\mathsf{P}^{\inf}_{\mathsf{T},s}(s, \mathsf{f})$ and $\mathsf{P}^{\sup}_{\mathsf{T},s}(s, \mathsf{f})$, respectively.

According to [5], the sets of states for which $\mathsf{P}^{\inf}(s) = 0$ or $\mathsf{P}^{\sup}(s) = 0$ can be computed by resorting to simple fixpoint computations on graphs, whereas the infimum and supremum probabilities of the other states satisfies the following equations:

$$\mathsf{P}^{\inf}(s) = \min_{n \in \mathcal{N}(s)} \textstyle\sum_{s' \in S} \tau(n)(s')\mathsf{P}^{\inf}(s') \quad (1)$$

$$\mathsf{P}^{\sup}(s) = \max_{n \in \mathcal{N}(s)} \textstyle\sum_{s' \in S} \tau(n)(s')\mathsf{P}^{\sup}(s')1 \quad (2)$$

To solve such a system, two methods have been explored: one can either transform such a system into a linear programming problem, and use classical techniques of linear programming, or consider the system as a fixpoint equation, and compute its least fixpoint by iterative methods. The solving method is however not the aim of this paper. We choosed linear programming with exact arithmetic, in order to avoid numerical problems and to get exact results.

Partitioning and complexity of the analysis. Basically, the two sources of complexity in these systems of equations are first the number of states of the PTS, which is of the same order as the number of variables, and then the number of nails, which gives the number of linear expressions in min or max expressions.

Partitioning the state space allows to address the first source of complexity. The question is how to proceed with the nails. Consider the PTS depicted in Fig. 3(a). The first effect of the abstraction is that several edges outgoing from a nail will lead to the same class; we have to merge these edges and add their probabilities, as shown on Fig. 3(b), where s_0, s_1 and s_2, s_3 are merged into equivalence classes k_0 and k_1. A second effect is that this operation makes some nails become equivalent, as (s_0, a_0) and (s_0, a_1) on Fig. 3(b). This effect is our main point: we expect that partitioning the state space will equate many nails and therefore address the second source of complexity. Such a situation is very likely to happen in systems that are specified in a symbolic way using data variables (see example 1).

5 Algorithms and Data Structures

Representation of states, transition relation, and partitions. As stated in the introduction, we use BDDs to represent sets of states, and ADDs to represent the transition relation.

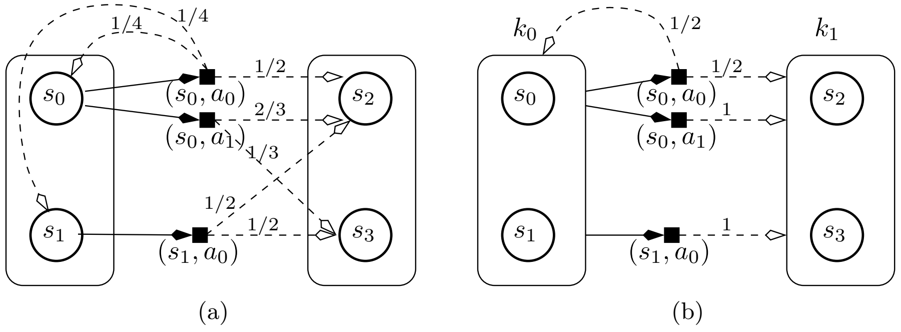


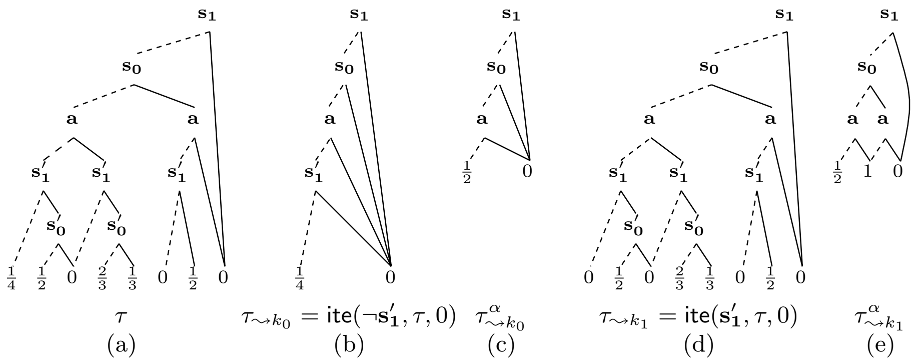
Fig. 3. A concrete PTS and its abstraction

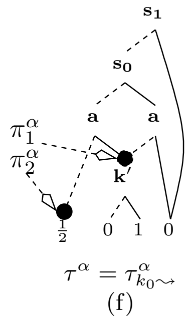
As S is finite, we can encode each state $s \in S$ by a Boolean vector $\boldsymbol{s}$ of length $n = \lceil \log_2 |S| \rceil$. We then use BDDs to represent (sets of) states. Similarly, we use ADDs to represent the function τ, which belongs to the space $N \to (S \to [0,1])$, isomorphic to the space $N \times S \to [0,1]$. In order to encode nails, we use an auxiliary set A that solves in each state the nondeterministic choice on its outgoing nails. Let $p = \log(\max_{s \in S} |\mathcal{N}(s)|)$ and $\mathbb{B} = \{0,1\}$. We consider that $S \subseteq \mathbb{B}^n$, $A \subseteq \mathbb{B}^p$, $N \subseteq S \times A$, and $\tau : S \times A \times S \to [0,1]$. τ is then represented by an ADD using unprimed variables, auxiliary and primed variables, noted $\overrightarrow{\mathbf{s}}$, $\overrightarrow{\mathbf{a}}$, $\overrightarrow{\mathbf{s'}}$. The Boolean vectors $\boldsymbol{s}$ and $\boldsymbol{s'}$ represent states s and s', using respectively unprimed and primed variables. Each nail $n \in \mathcal{N}(s)$ outgoing from a state s is thus encoded by a pair $(\boldsymbol{s}, \boldsymbol{a}) \in S \times A$. Fig. 4(a) shows the ADD representing the system of Fig. 3(a).

A partition of a finite set S is defined by a set K of *classes*, and a function $def : K \to 2^S$ such that: $\bigcup_{k \in K} def(k) = S$ and $\forall k \neq k' : def(k) \cap def(k') = \emptyset$ (and $\forall k : def(k) \neq \emptyset$). As usual, sets are represented by BDDs. In order to use classes k in BDDs, we use Boolean vectors $\boldsymbol{k}, \boldsymbol{k'} \in \mathbb{B}^n$, represented with variables $\overrightarrow{\mathbf{k}}$ and $\overrightarrow{\mathbf{k'}}$.

Simplification of a PTS and Boolean analysis. Before performing any abstraction, we first try to simplify the PTS, using conditions i and f. Obviously, states that are not reachable from states satisfying i cannot influence the value of $\mathsf{P}^{\inf}(\mathsf{i})$ and $\mathsf{P}^{\sup}(\mathsf{i})$, so we can discard them from S and simplify τ. In a different spirit, the states s satisfying $\mathsf{P}^{\inf}(s) = \mathsf{P}^{\sup}(s) = 0$ or $\mathsf{P}^{\inf}(s) = \mathsf{P}^{\sup}(s) = 1$ can be respectively gathered in a *safe* partition or included in the final partition. These partitions are then transformed in a sink state by appropriately changing τ, since their outgoing transitions are irrelevant for the computation. Afterwards, a new reachability analysis allows to further simplify the state space and the transition function. The computation of such states can be done by fixpoint computations with BDDs, by considering a suitable Boolean abstraction of a PTS.

Notice that we do not necessarily reduce the number of nodes of BDDs and ADDs by the above simplifications. However, futile computations are avoided by restricting partitioning only to the relevant states. Boolean fixpoint computa-


τ
(a)
$\tau_{\rightsquigarrow k_0} = \mathsf{ite}(\neg \mathbf{s_1'}, \tau, 0)$
(b)
$\tau^{\alpha}_{\rightsquigarrow k_0}$
(c)
$\tau_{\rightsquigarrow k_1} = \mathsf{ite}(\mathbf{s_1'}, \tau, 0)$
(d)
$\tau^{\alpha}_{\rightsquigarrow k_1}$
(e)



The states s_0, s_1, s_2, s_3 are encoded by Boolean vectors on variables $(\mathbf{s_0}, \mathbf{s_1})$, $\mathbf{s_0}$ being the least significant bit; a_0 and a_1 are encoded by variable $\mathbf{a}$, and class k_0 and k_1 by variable $\mathbf{k}$. The left branch of a node corresponds to a false value for its variable. For readability, leaves are sometimes duplicated. The nodes π_1^{α} and π_2^{α} represent respectively the distributions $\{k_0 \mapsto \frac{1}{2}; k_1 \mapsto \frac{1}{2}\}$ and $\{k_0 \mapsto 0; k_1 \mapsto 1\}$. We have $g(k_0, \pi_1^{\alpha}) = \neg \mathbf{s_0} \wedge \neg \mathbf{s_1}$ and $g(k_0, \pi_2^{\alpha}) = \neg \mathbf{s_1}$.

Fig. 4. Representation and abstraction of the PTS of Fig. 3 with ADDs

tions are also used on PTS abstracted by partitioning in order to compute the set of classes k for which $\mathsf{P}^{\inf}(k) = 0$ or $\mathsf{P}^{\sup}(k) = 0$, which is required to solve equations (1) and (2) in Section 4.

Abstraction of a PTS by partitioning. The problem is the following: given a system $(S, N \subseteq S \times A, Org, \tau)$ and a partition (K, def) of S, compute an *abstract* system $(S^{\alpha}, N^{\alpha}, Org^{\alpha}, \tau^{\alpha})$ with

$$S^{\alpha} = K \ , \ N^{\alpha} = N \ , \ \tau^{\alpha} : \begin{array}{rl} N & \to \mathsf{Distr}(K) \\ (s,a) & \mapsto \lambda k. (\sum_{s' \in def(k)} \tau(s,a,s')) \end{array}$$

We do not specify the function Org^{α} since it is not necessary to compute it.

To compute τ^{α} with ADDs, we first transform the summation indexed by $\boldsymbol{s'} \in def(k)$ into an unconstrained summation. For $k' \in K$, define the ADD

$$\tau_{\rightsquigarrow k'}(\boldsymbol{s}, \boldsymbol{a}, \boldsymbol{s'}) = \mathsf{ite}(\boldsymbol{s'} \in def(k'), \tau(\boldsymbol{s}, \boldsymbol{a}, \boldsymbol{s'}), 0)$$

where ite is the if-then-else operator on ADDs (see Figs. 4(b) and (d)). Then, for every $(\boldsymbol{s}, \boldsymbol{a}, \boldsymbol{k'})$, $\tau^{\alpha}(\boldsymbol{s}, \boldsymbol{a}, \boldsymbol{k'}) = \sum_{\boldsymbol{s'}} \tau_{\rightsquigarrow k'}(\boldsymbol{s}, \boldsymbol{a}, \boldsymbol{s'})$, which we note $\tau^{\alpha}_{\rightsquigarrow k'}(\boldsymbol{s}, \boldsymbol{a})$. This unconstrained summation on all valuations taken by primed variables correspond exactly to the *existential quantification* of primed variables in the ADD $\tau_{\rightsquigarrow k'}$, as defined for instance in the library CUDD [30]. This operation benefits from the usual caching techniques of BDDs, and can be implemented in a time quadratic in the number of nodes of the input graph. So we have

$$\begin{aligned} \tau^{\alpha}_{\rightsquigarrow k'}(\boldsymbol{s},\boldsymbol{a}) &= \textstyle\sum_{\overrightarrow{\mathbf{s}}} \tau_{\rightsquigarrow k'}(\boldsymbol{s},\boldsymbol{a},\boldsymbol{s'}) \\ \text{and}\ \ \tau^{\alpha}(\boldsymbol{s},\boldsymbol{a},\boldsymbol{k'}) &= \textstyle\sum_{k'\in K} \mathsf{ite}(\boldsymbol{k'}, \tau^{\alpha}_{\rightsquigarrow k'}(\boldsymbol{s},\boldsymbol{a}), 0) \end{aligned}$$

where $\sum_{k'\in K}$ is a disjoint summation on ADDs implemented by a cascade of ite operators (see Figs. 4(c), (e), and (f)). Computing τ^{α} in that way requires $|K|$ intersection operations (to obtain $\tau_{\rightsquigarrow k}$), $|K|$ existential quantifications on ADDs, and $|K|$ applications of the ite operator.

From ADDs to equations on abstract system. Let $(S^{\alpha}, N^{\alpha}, Org^{\alpha}, \tau^{\alpha})$ be an abstract system defined by a partition with initial class k_{i} and final class k_{f}. We want to compute $\mathsf{P}^{\inf}(k_{\mathsf{i}})$ and $\mathsf{P}^{\sup}(k_{\mathsf{i}})$. Therefore we need to generate the systems of inequalities (1) and (2) from the ADD τ^{α}. That is, to each class k, we have to associate its set of outgoing nails $\{(s,a) \mid s \in \mathit{def}(k)\}$, extracting the corresponding distributions and detecting efficiently identical distributions to avoid redundancy in equations.

We select the nails outgoing from a class k by computing

$$\tau^{\alpha}_{k\rightsquigarrow}(\boldsymbol{s},\boldsymbol{a},\boldsymbol{k'}) = \mathsf{ite}(\boldsymbol{s} \in \mathit{def}(k), \tau^{\alpha}(\boldsymbol{s},\boldsymbol{a},\boldsymbol{k'}), 0)$$

The important point now for the extraction of the distributions is that we require the variables $\overrightarrow{\mathbf{k'}}$ to be ordered below the variables $\overrightarrow{\mathbf{s}}$ and $\overrightarrow{\mathbf{a}}$ in ADDs. This allows to extract the distributions by performing a depth-first search of the graph rooted at $\tau^{\alpha}_{k\rightsquigarrow}$, stopping as soon as a node indexed by a variable belonging to $\overrightarrow{\mathbf{k'}}$ is encountered. Such a node corresponds to a distribution (Fig. 4(f)). Because of this variable ordering and the sharing of nodes the ADD $\tau^{\alpha}_{k\rightsquigarrow}$, the set of its *different* distributions can be obtained for free by a simple graph algorithm.

The third step, generating a linear expression from an ADD representing a distribution, is done by enumerating the valuations on variables $\overrightarrow{\mathbf{k'}}$ that leads to a non-zero leaf, c.f. Fig. 4(f) and the explanations. We resort then to section 4 to solve the system of equations.

Automatic partition refinement. The choice of a suitable abstraction is a difficult problem, because only the results of the analysis can decide whether the abstraction offers enough precision to check the intended property. This is why we have chosen an incremental partitioning method.

The verification starts with a rough partition of the system. If the analysis of this abstract PTS allows to conclude that the property is satisfied by the concrete PTS, the verification process is finished. Otherwise, a partition refinement step is performed in order to obtain more precise information. This process is iterated up to success or until all classes of the partition are stable. If this last situation occurs, we can conclude that the property is false and extract a counter-example path.

The initial partition contains three distinguished classes: the safe, initial, and final classes, denoted k_{s}, k_{i}, and k_{f}, with $\mathit{def}(k_{\mathsf{s}}) = \{s \mid \mathsf{P}^{\sup}(s) = 0\}$, $\mathit{def}(k_{\mathsf{i}}) = \mathsf{i}$, and $\mathit{def}(k_{\mathsf{f}}) = \mathsf{f}$. The safe and final classes are never split. As our tool allows to specify processes that combines an explicit control structure and operations on data variables, we use this control structure to partition the remaining state space.

Our refinement method tries to stabilize classes, by separating concrete states in a class that have different future, as do all partition refinement methods based on a bisimulation criteria [6,1,31], and most of those dealing with infinite state systems [29,21]. We implement this idea by considering the set of states $g(k, \pi^\alpha) \subseteq \mathit{def}(k)$ in class k that can lead to the abstract distribution π^α. $g(k, \pi^\alpha)$ can be seen as the *guard* of the distribution π^α in class k. For instance, in Fig. 3(b), if π_1^α denotes the distribution attached to the nail (s_0, a_0), then $g(k_0, \pi_1^\alpha) = \{s_0\}$, and if π_2^α denotes the distribution associated to nails (s_0, a_1) and (s_1, a_0), $g(k_0, \pi_2^\alpha) = \{s_0, s_1\}$. If such a guard is neither empty, nor equal to the definition of the class k, then the class k can be safely split into two classes k' and k'' according to this *discriminating* guard, with $\mathit{def}(k') = g(k, \pi^\alpha)$ and $\mathit{def}(k'') = \mathit{def}(k) \setminus g(k, \pi^\alpha)$, because states in class k' are certainly not bisimilar to states in class k''. A guard $g(k, \pi^\alpha)$ is obtained by computing the union of paths in the Add $\tau_{k\rightsquigarrow}^\alpha$ that leads from the root node to the node representing the distribution π^α (Fig. 4(f)). Such an operation can again be implemented with a complexity linear in the number of nodes of the Add $\tau_{k\rightsquigarrow}^\alpha$.

Our global strategy for refinement tries, between each analysis step, to split once every class for which there exist a guard. After a partition refinement, a new abstract transition function τ^α has to be computed. When a class k has not been split, the Adds $\tau_{\rightsquigarrow k}$ and $\tau_{\rightsquigarrow k}^\alpha$ are reused; otherwise, we need to recompute them, as well as the Adds τ^α and $\tau_{k\rightsquigarrow}^\alpha$. So the refinement process require $\mathcal{O}(|K|)$ Bdds operations.

Conclusion. The algorithms presented in this section allows to partition and to refine an abstract PTS with $\mathcal{O}(|K|)$ Bdds operations; the complexity of these operations is in turn linear or quadratic in the number of *nodes* of the input diagrams.

6 Example

The Bounded Retransmission Protocol (BRP) [15,12,7] has become a nice benchmark example as it is simple to understand, yet its overall behaviour is not trivial. The BRP is based on the alternating bit protocol but allows for a bounded number of retransmissions of a *chunk*, i.e., part of a file, only. So, eventual delivery is not guaranteed and the protocol may abort the file transfer. By using our technique, we are able to quantify the probability of such abortion.

The protocol consists of a Sender and a Receiver exchanging data via two unreliable (lossy) channels, K and L. The Sender reads a file to be transmitted (which is assumed to be divided in N chunks) and sets the retry counter to 0. Then it sends the elements of the file one by one over K to the Receiver. A *frame* sent through channel K consists of three bits and a chunk. The first bit indicates

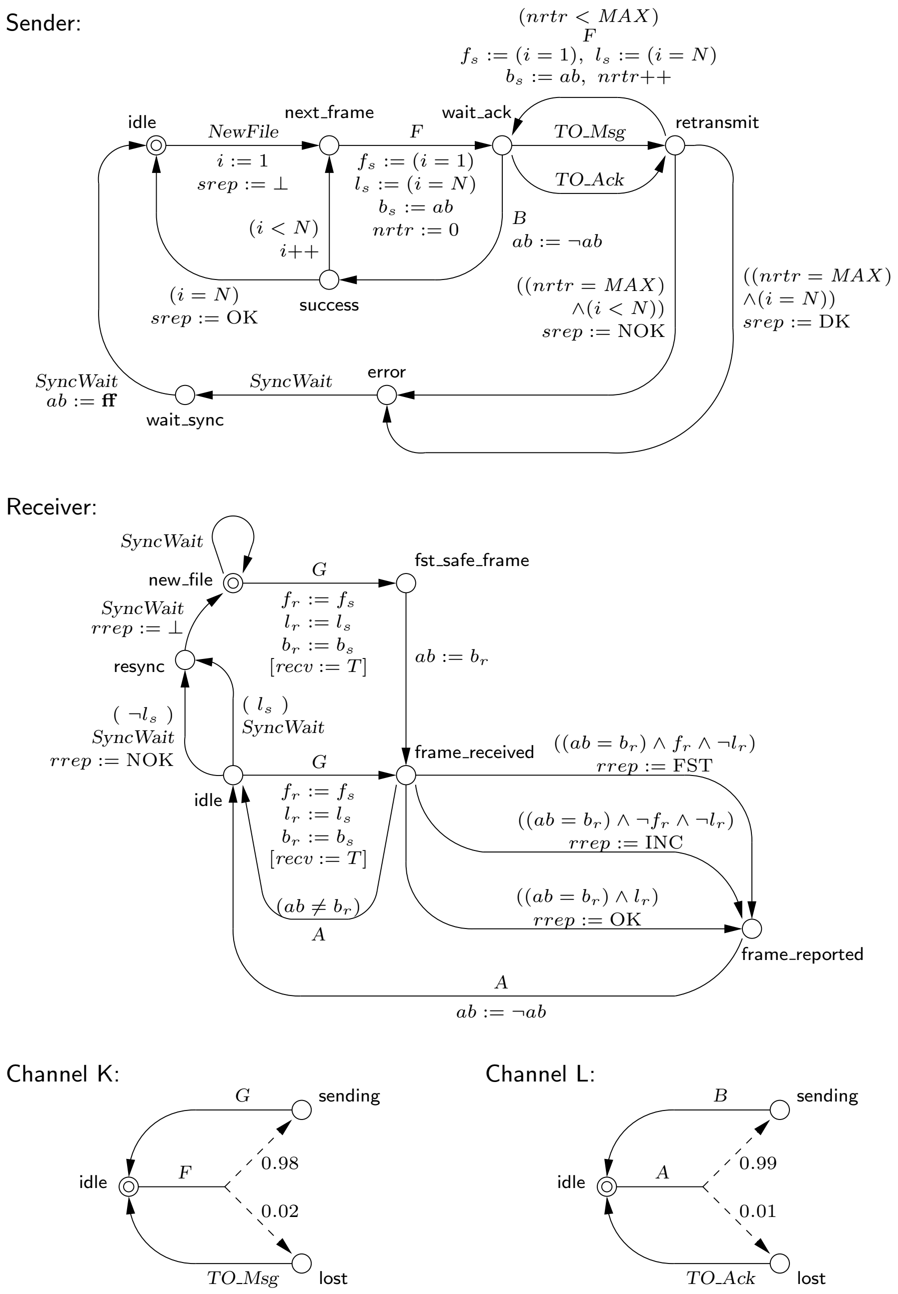


Fig. 5. PTS model of the bounded retransmission protocol

whether the chunk is the first element of the file, the second one indicates if it is the last one, and the third bit, the so-called alternating bit, is used to guarantee that data is not duplicated. After sending the frame, the Sender waits for an acknowledgement or for a timeout. In case of acknowledgement, if it corresponds to the last chunk, the sending client is informed of correct transmission (signal OK); otherwise the next element of the file is sent. If a timeout occurs, the frame is resent (after the counter for the number of retries is incremented), or the transmission of the file is broken off. The latter occurs if the retry counter exceeds its maximum value MAX. In this case the sender client is informed whether the Sender did not complete the transmission (NOK), or whether it sent the last chunk but it was never acknowledge (DK) in which case the success of the transmission is unknown. Afterwards, and before sending a new file, the Sender waits enough time to ensure that the Receiver has properly reacted to the communication break.

The Receiver waits for a frame to arrive. This frame is delivered at the receiving client informing whether it is the first (FST), an intermediate (INC), or the last one (OK). Afterwards, an acknowledgement is sent over L to the Sender. Then the Receiver simply waits for more frames to arrive. The receiver remembers whether the previous frame was the last element of the file and the expected value of the alternating bit. Each frame is acknowledged, but it is handed over to the receiving client only if the alternating bit indicates that it is new. Note that (only) if the previous frame was last of the file, then a fresh frame will be the first of the subsequent file and a repeated frame will still be the last of the old file. If a long enough time had passed since the last frame was received, the Receiver assumes that the normal communication flow broke down. If this happen, the receiving client is informed, provided the last element of the file has not yet been delivered. Since our model does not consider time, we assume that premature timeouts are not possible and that the Sender and Receiver always re-synchronise properly after normal communication is broken.

The description of the components of the protocol in terms of PTS is given in Fig. 5. It abstracts from the data that is being transmitted. The components synchronise through common alphabet (a la CSP [19]). Notice that the only probabilistic features are those occurring in the medium. In this model we assume that a frame is lost with probability 0.02, and acknowledgement is lost with probability 0.01.

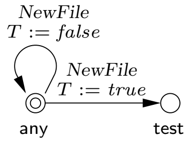


Fig. 6.

A checking automaton (Fig. 6) ensures that the transmitted file is invariant for the property under study, i.e, the property is only interesting for exactly one file transmission. Notice that the checking automata selects an arbitrary file to test. We study several properties. The considered initial condition is test@Check $\wedge$ next_frame@Sender $\wedge$ (i@Sender $= 1$). The different final conditions are listed in Table 1. Notice the flag *recv* at the Receiver side; it is used to register that the last sent file has actually started to be received. Properties A and B define the minimal correctness requirement of the protocol. They should *not* be valid.

Table 1. Reachability Conditions

	Final condition	Meaning of the property
A	$(\mathit{srep}@\mathsf{Sender} = \mathrm{NOK}) \wedge (\mathit{rrep}@\mathsf{Receiver} = \mathrm{OK}) \wedge (\mathit{recv}@\mathsf{Receiver})$	The Sender reports a certain unsuccessful transmission but the Receiver got the complete file. (The probability should be 0!)
B	$(\mathit{srep}@\mathsf{Sender} = \mathrm{OK}) \wedge \neg(\mathit{rrep}@\mathsf{Receiver} = \mathrm{OK}) \wedge (\mathit{recv}@\mathsf{Receiver})$	The Sender reports a certain successful transmission but the Receiver did not get the complete file. (The probability should be 0!)
1	$\mathsf{error}@\mathsf{Sender}$	The Sender does not report a successful transmission
2	$\mathsf{error}@\mathsf{Sender} \wedge (\mathit{srep}@\mathsf{Sender} = \mathrm{DK})$	The Sender reports an uncertainty on the success of the transmission
3	$\mathsf{error}@\mathsf{Sender} \wedge (\mathit{srep}@\mathsf{Sender} = \mathrm{NOK}) \wedge (i@\mathsf{Sender} > 8)$	The Sender reports a certain unsuccessful transmission after transmitting half a file
4	$\neg(\mathit{srep}@\mathsf{Sender} = \bot) \wedge \neg(\mathit{recv}@\mathsf{Receiver})$	The Receiver does not receive any chunk of a file, i.e., the first message never arrives. "$\neg(\mathit{srep}@\mathsf{Sender} = \bot)$" ensures that the Sender did try to send a chunk

Properties 1 to 3 are concerned with transmissions that the Sender does not consider successful while property 4 considers an attempt for transmission with no reaction at the Receiver side.

The exercise we perform is to try to find the minimum number of retransmissions (MAX) that satisfies our probabilistic requirements for these properties when the transmitted file has length $N = 16$. Table 2 reports these results. Some remarks are in order. Each row in Table 2 reports a different instance according to the maximum number of retransmission MAX which is specified in the first column. The second column reports the number of reachable states in the respective instance (#reach.), and the third one the number of relevant states, i.e., reachable states that may lead to a state satisfying the final condition (#relev).

For each property we tried two different values of desired probability. Thus, for instance, property 1 is require to hold with probability less than 0.05 in the first experiment and than 0.01 in the second one. The least three columns report the last possible refinement together with its respective convergence value. For each experiment we report the number of refinements (#refin.) necessary to conclude the required property and for this last refinement, the number of abstract states (#abst.) and the upper bound for the actual minimum and maximum probability ($\mathsf{P}^{\mathrm{inf}}$ and $\mathsf{P}^{\mathrm{sup}}$, respectively). We also report whether the property holds ($\surd$) or not ($\times$) on the verdict columns (Verd.)

Notice that, in this example, the proposed method do the actual verdict on an abstract state space which is, on average, around 20 times smaller than the concrete reachable state space. In particular it has performed quite well for Properties 2 and 3 in the larger systems ($MAX \geq 3$). We have experience two

Table 2. Results in a BRP with file length = 16

MAX	#reach.	#relev.	#refin.	#abst.	$p^{\inf}$	$p^{\sup}$	Verd.	#refin.	#abst.	$p^{\inf}$	$p^{\sup}$	Verd.	#refin.	#abst.	$p^{\inf} = p^{\sup}$
Property 1:			prob. $\leq 10^{-3}$					prob. $\leq 10^{-5}$					Convergence		
0	1174	269	1	12	0.0298	1	×	1	12	0.0298	1	×	88	99	0.383717
1	2068	697	6	35	1.5679e-03	1	×	1	17	6.01899e-04	1	×	92	247	0.0141144
2	2962	1125	92	404	4.22546e-04	4.24145e-04	√	3	27	1.20404e-05	1	×	94	411	4.23333e-04
3	3856	1553	94	564	1.25943e-05	1.2642e-05	√	76	459	1.02284e-05	1	×	96	575	1.26178e-05
4	4750	1981	95	724	3.75311e-07	3.76733e-07	√	95	724	3.75311e-07	3.76733e-07	√	97	739	3.76012e-07
5	5644	2409	95	884	1.11843e-08	1.12267e-08	√	95	884	1.11843e-08	1.12267e-08	√	97	903	1.12051e-08
Property 2:			prob. $\leq 10^{-4}$					prob. $\leq 10^{-6}$					Convergence		
0	1174	1134	85	99	0.0189293	0.0189293	×	85	99	0.0189293	0.0189293	×	85	99	0.0189293
1	2068	2028	87	244	8.76261e-04	8.76307e-04	×	87	244	8.76261e-04	8.76307e-04	×	89	247	8.76284e-04
2	2962	2922	8	54	0	2.64633e-05	√	89	404	2.64531e-05	2.64531e-05	×	91	411	2.64531e-05
3	3856	3816	9	72	0	8.88033e-06	√	10	76	0	7.88615e-07	√	93	575	7.88606e-07
4	4750	4710	9	81	0	2.64636e-05	√	11	93	0	2.64636e-07	√	96	739	2.35007e-08
5	5644	5604	9	82	0	2.64636e-05	√	11	97	0	7.88615e-07	√	99	903	7.00322e-10
Property 3:			prob. $\leq 10^{-3}$					prob. $\leq 10^{-5}$					Convergence		
0	1174	950	43	93	0.149825	0.149825	×	43	93	0.149825	0.149825	×	43	93	0.149825
1	2068	1844	45	229	6.15567e-03	6.15599e-03	×	45	229	6.15567e-03	6.15599e-03	×	47	232	6.15584e-03
2	2962	2738	42	360	0	1.85196e-04	√	47	379	1.85191e-04	1.85191e-04	×	49	386	1.85191e-04
3	3856	3632	59	519	0	5.52026e-06	√	59	519	0	5.52026e-06	√	68	540	5.52026e-06
4	4750	4526	43	371	0	3.04092e-04	√	46	379	0	1.64505e-07	√	102	693	1.64505e-07
5	5644	5420	52	455	0	8.8846e-06	√	52	455	0	8.8846e-06	√	128	848	4.90225e-09
Property 4:			prob. $\leq 10^{-3}$					prob. $\leq 10^{-5}$					Convergence		
0	1174	256	1	4	0.02	0.02	×	1	4	0.02	0.02	×	1	4	0.02
1	2068	465	1	6	4e-04	4e-04	√	1	6	4e-04	4e-04	×	1	6	4e-04
2	2962	674	1	6	0	4e-04	√	3	8	8e-06	8e-06	√	3	8	8e-06
3	3856	883	1	6	0	4e-04	√	3	8	0	8e-06	√	5	10	1.6e-07
4	4750	1092	1	6	0	4e-04	√	3	8	0	8e-06	√	7	12	3.2e-09
5	5644	1301	1	6	0	4e-04	√	3	8	0	8e-06	√	9	14	6.4e-11

Table 3. Performance (with time format “h:mm:ss.d”)

MAX	Property 1		Property 2		Property 3		Property 4	
	($\leq 10^{-5}$)	Converg.	($\leq 10^{-6}$)	Converg.	($\leq 10^{-5}$)	Converg.	($\leq 10^{-5}$)	Converg.
0	0.5	11.6	13.3	13.5	8.7	8.8	0.4	0.4
1	0.6	2:25.6	3:54.1	4:11.9	1:49.4	2:02.2	0.5	0.5
2	1.2	8:27.8	11:02.8	11:24.8	5:16.9	5:55.2	0.6	0.6
3	8:50.5	17:05.8	6.3	22:44.9	11:29.2	15:30.7	0.7	0.7
4	27:19.9	28:52.1	8.8	35:50.7	5:58.0	40:52.9	0.7	0.8
5	41:09.3	45:05.0	10.2	52:21.0	11:03.0	1:31:14.7	0.8	0.9

different situations: either there is a gradual convergence to the infimum, but almost none to the supremum until an abrupt convergence in the last refinements (e.g. Property 1), or vice-versa (e.g Props. 2 and 3). The first case case may allow for an early rejection of the required property but would require many refinements if the property does hold (compare the number of refinements and abstract states of the × cases against the √ cases in Property 1). Instead, the second case will give an early report if the property holds (compare now the different results in Property 2).

The exercise of convergence is more costly as no criterion to stop the execution is provided and it proceeds until no more refinement is possible or the probability has definitely converged. At this maximum point notice that the state compression ratio is 10 times on average.

7 Concluding Remarks

In this article we introduced an efficient technique for quantitative model checking. The method relies on automatic abstraction of the original system. This allows to significantly reduce the size of the problem to which numerical analysis is applied in order to compute the quantitative factor of the property under study. Since the numerical analysis is the most costly part of the whole process this reduction is of high importance. This reduction is achieved first because bisimilar states are never distinguished, and secondly because using incremental abstraction refinement and confronting the analysis against a desired (or undesired) probability allows prompt answers on very compact spaces.

The execution time is currently not the best as the tool should be optimised. Table 3 reports the tool performance for a set of properties[1]. The current implementation performs numerical analysis using linear programming techniques under exact rational arithmetics. This method is very fast (compared to the painfully slow iterative methods) and it does not suffer of numerical unstability since numbers are represented in its exact form. Two remarks are in order. First, numerical analysis is applied in each refinement step, which is inefficient since a refinement step may add only few partitions with low chances of sensibly affecting the result of the previous iteration. Second, the already mentioned asymmetric convergence in which only the minimum or the maximum gradually converges to the actual value while the other does not until the last refinements.

It is in our near future plans to develop efficiency improvements. One of these improvements concerns the refinement strategy and the suitable alternation of refinement and analysis that should be used. Another improvement would be to take advantage of the fact that probabilities usually appears only in some part of the modelled system: failures do not appear everywhere!

On a long term agenda, we plan to use this incremental refinement technique to check probabilistic timed automata. Model checking of PTCTL properties on such model was proven decidable by resorting to their region graphs [25]. However, region graphs are known to be impractical. Our technique would allow to generate progressively a minimal *probabilistic* model, in the spirit of [1].

Acknowledgements. We thank Holger Hermanns and Joost-Pieter Katoen for fruitful discussions.

References

1. R. Alur, C. Courcoubetis, N. Halbwachs, D. Dill, and H. Wong-Toi. Minimization of timed transition systems. In R. Cleaveland, ed., *Procs. of CONCUR 92,* Stony Brook, NY, LNCS 630, pp. 340–354. Springer, 1992.
2. A. Aziz, V. Singhal, F. Balarin, R.K. Bryton, and A.L. Sangiovanni-Vincentelli. It usually works:the temporal logics of stochastic systems. In P. Wolper, ed., *Procs. of the 7th CAV,* Liège, LNCS 939, pp. 155–165. Springer, 1995.

[1] Time measured on a Sun Enterprise E3500 under Solaris 5.6.

3. R.I. Bahar, E.A. Frohm, C.M. Gaona, G.D. Hachtel, E. Macii, A. Pardo, and F. Somenzi. Algebraic decision diagrams and their applications. *Formal Methods in System Design*, 10(2/3):171–206, 1997.
4. C. Baier, J.-P. Katoen, and H. Hermanns. Approximate symbolic model checking of continuous-time Markov chains. In J.C.M. Baeten and S. Mauw, eds., *Procs. of CONCUR 99,* Eindhoven, LNCS 1664, pp. 146–161. Springer, 1999.
5. A. Bianco and L. de Alfaro. Model checking of probabilistic and nondeterministic systems. In *Procs.* 15th *FSTTCS ,* Pune, LNCS 1026, pp. 499–513. Springer, 1995.
6. A. Bouajjani, J. C. Fernandez, N. Halbwachs, P. Raymond, and C. Ratel. Minimal state graph generation. *Science of Computer Programming*, 18:247–269, 1992.
7. P.R. D'Argenio, J.-P. Katoen, T.C. Ruys, and J. Tretmans. The bounded retransmission protocol must be on time! In E. Brinksma, ed., *Procs. of the 3rd TACAS,* Enschede, LNCS 1217, pp. 416–431. Springer, 1997.
8. P.R. D'Argenio, B. Jeannet, H.E. Jensen, and K.G. Larsen. Reachability Analysis of Probabilistic Systems by Successive Refinements. CTIT Technical Report, 2001. To appear.
9. L. de Alfaro, M. Kwiatkowska, G. Norman, D. Parker, and R. Segala. Symbolic model checking of concurrent probabilistic processes using MTBDDs and the Kronecker representation. In Graf and Schwartzbach [11].
10. M. Fujita, P.C. McGeer, and J.C.-Y. Yang. Multi-terminal binary decision diagrams: An efficient data structure for matrix representation. *Formal Methods in System Design*, 10(2/3):149–169, April 1997.
11. S. Graf and M. Schwartzbach, eds. *Procs. of the 6th Workshop TACAS,* Berlin, LNCS 1785. Springer, 2000.
12. J.F. Groote and J. van de Pol. A bounded retransmission protocol for large data packets –A case study in computer checked algebraic verification–. In M. Wirsing and M. Nivat, eds., *Procs. of the* 5th *AMAST Conference,* Munich, LNCS 1101. Springer, 1996.
13. H.A. Hansson and B. Jonsson. A logic for reasoning about time and reliability. *Formal Aspects of Computing*, 6:512–535, 1994.
14. V. Hartonas-Garmhausen and S. Campos. ProbVerus: Probabilistic symbolic model mhecking. In In Katoen [24], pp. 96–110.
15. L. Helmink, M.P.A. Sellink, and F.W. Vaandrager. Proof-checking a data link protocol. In H. Barendregt and T. Nipkow, eds., *Procs. International Workshop TYPES'93,* Nijmegen, LNCS 806, pp. 127–165. Springer, 1994.
16. H. Hermanns. Personal communication, 2001.
17. H. Hermanns, J.-P. Katoen, J. Meyer-Kayser, and M. Siegle. A Markov chain model checker. In Graf and Schwartzbach [11], pp. 347–362.
18. H. Hermanns, J. Meyer-Kayser, and M. Siegle. Multi terminal binary decision diagrams to represent and analyse continuous time Markov chains. In B. Plateau, W.J. Stewart, and M. Silva, eds., *3rd Int. Workshop on the Numerical Solution of Markov Chains*, pp. 188–207. Prensas Universitarias de Zaragoza, 1999.
19. C.A.R. Hoare. *Communicating Sequential Processes.* Prentice-Hall International, Englewood Cliffs, 1985.
20. M. Huth and M. Kwiatkowska. Quantitative analysis and model checking. In *Procs.* 12th *Annual Symposium on Logic in Computer Science,* Warsaw. IEEE Press, 1997.
21. B. Jeannet. Dynamic partitioning in linear relation analysis. Application to the verification of reactive systems. *Formal Methods in System Design*, 2001. To appear.

22. B. Jonsson and K.G. Larsen. Specification and refinement of probailistic processes. In *Procs.* 6^{th} *Annual Symposium on Logic in Computer Science,* Amsterdam, pp. 266–277. IEEE Press, 1991.
23. B. Jonsson, K.G. Larsen, and W. Yi. Probabilistic extensions in process algebras. In J.A. Bergstra, A. Ponse, and S. Smolka, eds., *Handbook of Process Algebras*, pp. 685–710. Elsevier, 2001.
24. J.-P. Katoen, ed. *Procs of the 5th ARTS,* Bamberg, LNCS 1601. Springer, 1999.
25. M. Kwiatkowska, G. Norman, R. Segala, and J. Sproston. Automatic verification of real-time systems with probability distributions. In Katoen [24], pp. 75–95.
26. K.G. Larsen and A. Skou. Bisimulation through probabilistic testing. *Information and Computation*, 94:1–28, 1991.
27. M.L. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming.* John Wiley & Sons, 1994.
28. R. Segala. *Modeling and Verification of Randomized Distributed Real-Time Systems.* PhD thesis, Massachusetts Institute of Technology, 1995.
29. H. Sipma, T.E. Uribe, and Z. Manna. Deductive model checking. In R. Alur and T.A. Henzinger, eds. *Procs. of the 8th CAV,* New Brunswick, New Jersey, LNCS 1102. Springer, 1996.
30. F. Somenzi. CUDD: Colorado University Decision Diagram Package. `ftp://vlsi.colorado.edu/pub`.
31. R. F. Lutje Spelberg, W. J. Toetenel, and M. Ammerlaan. Partition refinement in real-time model checking. In A.P. Ravn and H. Rischel, eds., *Procs. of the 5th FTRTFT,* Lyngby, LNCS 1486, pp. 143–157. Springer, 1998.