Rapture: A tool for verifying Markov Decision Processes

Bertrand Jeannet[1], Pedro R. D'Argenio[2], and Kim G. Larsen[3]

[1] IRISA – INRIA, Campus de Beaulieu. F-35042 Rennes Cedex. France
`Bertrand.Jeannet@irisa.fr`

[2] FaMAF - UNC, Ciudad Universitaria. 5000 - Córdoba. Argentina
`dargenio@mate.uncor.edu`

[3] BRICS - Aalborg University, Frederik Bajers vej 7-E. DK-9220 Aalborg. Denmark
`kgl@cs.auc.dk`

Abstract. We present a tool that performs verification of quantified reachability properties over Markov decision processes (or probabilistic transition system). The originality of the tool is to provide two reduction techniques that limit the state space explosion problem: automatic abstraction and refinement algorithms, and a so-called essential states reduction. We present several case-studies to illustrate the usefulness of these techniques.

1 Introduction

Fully automatic verification of a specified transition system with respect to a given temporal logic property is known as *model checking* [22, 5]. For such systems model checkers allow to verify properties such as "the system will never reach an erroneous situation", and the property can be stated true or false. In many cases however, the absolute validity of a formula cannot be determined as wished, because of the nature of the system. For instance, consider a protocol that attempts to access to a lossy medium a bounded number of times after which it aborts. A property like "access will be granted" is obviously false. Nevertheless, to assess quality of service one would like the protocol grants access to the medium "often enough". Therefore, we would like, instead, to verify a *quantified* property like "access will be granted with probability at least 99%".

In this paper, we present Rapture[1], a tool that performs verification of quantified reachability properties over Markov decision processes (or probabilistic transition system). The system to be analysed is described as a parallel composition of finite probabilistic automata extended with finite-state variables. The automata communicate à la CSP [17] via synchronisation on a set of channels.

Other tools that verify quantified properties on (discrete time) Markov decision processes have been developed. For instance, ProbVerus [13] and

[1] Rapture is a loose acronym of *"Reachability Analysis of Probabilistic Transition systems based on REduction strategies"*. Rapture can be freely downloaded from `http://www.irisa.fr/prive/bjeannet/prob/prob.html`.

Prism [20] can check the validity of properties specified in the logic PCTL [12] (ProbVerus is however restricted to Markov chains). The originality of Rapture is to provide two reduction techniques that limit the state space explosion problem: automatic abstraction and refinement algorithms, and the so-called essential states reduction [6, 7]. The use of these techniques considerably reduces the high cost of the numerical analysis involved in the computation of the minimum and maximum reachability probabilities for PTSs. The price to pay is that Rapture cannot do verification of the full PCTL. Like ProbVerus and Prism, Rapture uses Bdds and Mtbdds [10, 1] to efficiently store the state space and the transition relation, but unlike them, Rapture uses these data structures to perform abstractions and process the refinement steps rather than to perform numerical analysis. Numerical analysis in Rapture is indeed performed by two different linear programming solvers: the first one uses sparse matrix on floating point numbers, the second uses dense matrix on exact rational numbers, which enables *exact* computations.

The paper is organized as follows: Section 2 shortly describes Rapture modelling language and Section 3, the properties it checks. Section 4 explains the machinery inside Rapture. Section 5 reports the performance of the tool on several case studies.

2 The model of systems: probabilistic transition systems

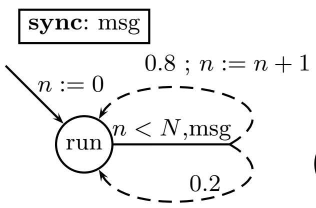
Probabilistic transition systems (PTS for short) generalize the well-known transition systems with probabilistic information. In a PTS, a transition does not lead to a single state but to a distribution over a set of states. The model we define is widely used (see, e.g. [23, 3, 18]) and is also known as Markov decision processes [21].

Fig. 1 depicts three PTSs and the result of their parallel composition. The **Sender** process sends a number N of messages. There is a probability 0.2 that it sends again the same message (here modelled by the absense of an increment of n). The **Line** process represents the transmission process. There is a probability 0.1 that the message is not transmitted to the **Receiver** process. The **Receiver** process just counts the number of received messages up to $M \geq N$. If M messages have been counted, the counter can be undermistically be reseted or maintained at this maximum value. In a "normal" execution, we should have $n = m$ at the end of the execution. The parallel composition operator uses synchronization over channels with a semantic *à la CSP* [17].

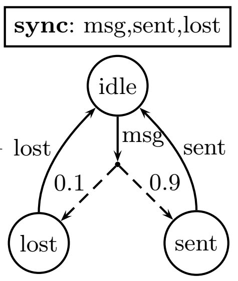
Such processes can be defined in Rapture with a textual language describing automata extended with finite-state variables, as shown on Fig. 1, where we have $N = M = 7$.

3 Rapture verification through reachability properties

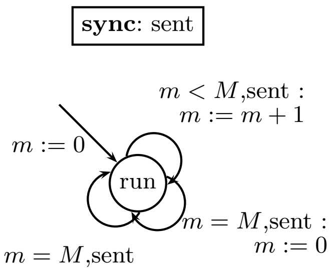
Informally, the properties Rapture verifies are of the type: *"the probability to reach a set of* final *states from a given* initial *state is lower (or greater) than a*



(a) Sender



(b) Line

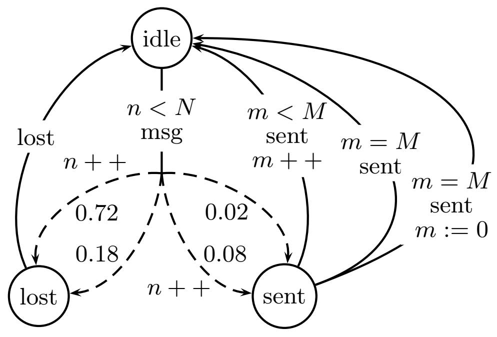


(c) Receiver

```
channel msg,sent,lost;

process Sender {
 sync msg;
 var  n : uint(3);
 init #run and n=0;
 loc run:
   when n<7 sync msg goto
     { run 0.8 assign { n:=n+1 };
       run 0.2 }
}
process Receiver {
 sync sent;
 var  m : uint(3);
 init #run and m=0;
 loc run:
   when m<7 sync sent goto run assign { m:=m+1 };
   when m=7 sync sent goto run;
   when m=7 sync sent goto run assign { m:=0 };
}
process Line {
 sync msg,lost,sent;
 init #idle;
 loc idle:
   when true sync msg goto { lost 0.1 ; sent 0.9 };
 loc lost:
   when true sync lost goto idle;
 loc sent:
   when true sync sent goto idle;
}

system Sender,Receiver,Line;
```



(d) Parallel composition

Fig. 1. Example PTS

given bound, for any execution of the system". For instance, in the PTS described in Fig. 1, provided that the system has reached a state with $n = 5$ and $m = 7$, is the probability to reach a state with $n = 7, m = 1$ for any execution greater (or lower) than 0.5? The important point is that this probability depends on the considered execution, as soon as the system exhibits non-determinism. For instance, if the PTS is in the state **sent** with $m = 7$, it can nondeterministically reset m or leave it unchanged.

We specify the set of initial and final states of our reachability property by adding to the textual description of fig 1 the lines

```
initial Sender.n=5 and Receiver.m=7;
final   Sender.n=7 and Receiver.m=1;
```

The probability of the property is specified on the control line of our tool. The property in quote can be checked with the command '`rapture -ratio 200 -goal i0.5 ex.pts`'. Flag '`-goal i0.5`' indicates that we would like to check that the minimum probability of the reachability property is above 0.5. Rapture returns the report given in Fig. 2: the minimum probability to reach the final set is 0, and it was proven by discrete fixpoint computation only. Indeed, if m is never reset, it is not possible to ever reach a final state. Therefore the property is false. Alternatively, we can check whether the probability of reaching a final state is always lower than 0.9 using, for instance, the command '`rapture -goal s0.9 ex.pts`'. Rapture returns the report given in Fig. 3, which states that the property is true.

```
** computing processes and expressions
** Boolean composition
** Boolean analysis
    Initially: 10 state variables, 1024 states
    relation: 101 nodes
    Reordering...
    Reachability analysis from init: 192 states, 5 nodes
    Reachability analysis from initial: 72 states, 8 nodes
    psup0: 20 states, 12 nodes
    Extending final states
    Second reachability analysis from initial:
                                       64 states, 10 nodes
    Non-sink state space: 48 states, 9 nodes
    Pinf=0: 51 states
    All initial states are in pinf0; pinf = 0.0

Example: (Sender.loc=run)(Receiver.loc=run)
         (Line.loc=sent,lost)
         (Sender.n=5)(Receiver.m=7)
```

Fig. 2. Infimum prob. ≥ 0.5

```
idem
...
** computing global probabilistic transition function
** size of the transition function: 119 nodes, 619 paths, 6 leaves
System build and analysed (Boolean analysis) in 0.06 seconds

Building initial partition in 0 seconds

Step 1, automaton has 6 locations and 10 nails, 5684 bytes
essential automaton has 5 locations and 9 nails
...
Step 3, automaton has 28 locations and 33 nails, 19552 bytes
essential automaton has 15 locations and 20 nails
Computing psup: 13 variables, 20 constraints, 6 equalities;
pinf=-inf  psup=0.828196810136  diff=inf
analysis in 0 seconds
** Success **
After 3 steps and 5 divisions, final automaton has 28 locations
pinf=-inf psup=0.828196810136 diff=inf
Times in seconds:
  building: 0.06
  analysis: 0.03 (numerical computations: 0; refinement: 0.03)
```

Fig. 3. Supremum prob. ≤ 0.9

4 Verification method

The standard verification method for verifying reachability properties is to compute the *minimum* and/or the *maximum* probability of reaching a final state from an initial state, and to use it to deduce the truth of the property. The computation of those *extremum probabilities* is done by solving a system of fixpoint equations involving min and max operators over sets of linear expressions.

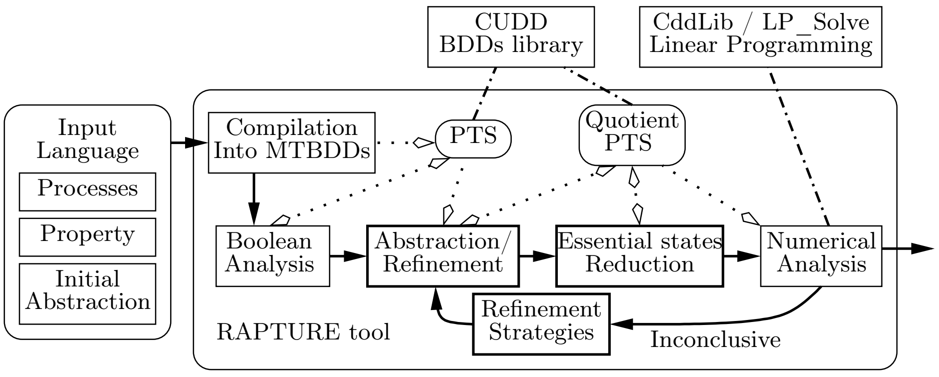


Fig. 4. Architecture of the RAPTURE tool

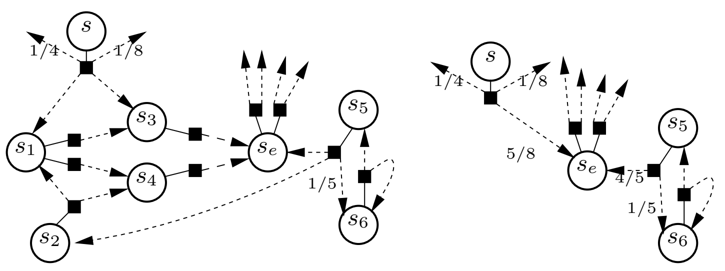
The two solving methods are the *value iteration method*, used together with a symbolic representation in [20], or the *linear programming method*. In both case, the number of unknowns is the number of the states of the analysed PTS. The aim of our tool is to reduce as much as possible the number of unknowns to be considered to compute extremum probabilities as efficiently as possible, possibly in an approximate way. The architecture of RAPTURE is depicted in Fig. 4.

Representation of Probabilistic Transition Systems. — Following [16, 8], we use BDDs (Binary Decision Diagrams [4]) to represent sets of states. Transition relations are represented with MTBDDs (Multi-Terminal Decision Diagrams), with real numbers as terminal nodes. We refer the reader to [6] for more details about the encoding and the chosen variable ordering in diagrams.

The reduction techniques implemented in the tool. — The purpose of the reduction techniques which have been implemented is to overcome the strong limitation in the size of the systems that can be verified. Three reduction techniques are implemented.

Discrete precomputations. — We use a standard precomputation of certain sets of system states in order to simplify the system before applying linear programming techniques. These sets are: the set of all reachable states *Reach*, and for each $p \in \{0, 1\}$ the set of states with infimum (resp. supremum) probability p of reaching F. These latter sets of states are denoted $\mathsf{P}^{\text{inf}}_{=0}$, $\mathsf{P}^{\text{inf}}_{=1}$, $\mathsf{P}^{\text{sup}}_{=0}$, and $\mathsf{P}^{\text{sup}}_{=1}$, respectively. All of the above sets can be computed using discrete fixpoint analysis [9] on a Boolean abstraction of the system. In our case these analysis are implemented using BDDs. As we restrict our attention to simple reachability properties, we can use these analysis to reduce the state space under consideration, unlike tools that handle more complex probabilistic properties involving nested fixpoints [13, 8, 15].

The Abstraction and successive refinement method. — The main principle of our method is founded on abstraction and successive refinements of the initial abstract model. The idea is to try the verification of the property on a (rough) abstraction of the model, induced by a partition of the state space, and in case of



(a) Initial situation (b) After reduction

Fig. 5. Example of essential reduction

failure of the verification to refine this abstract model into a finer abstraction, on which better approximations of extremum probabilities can be computed. This process is stopped as soon as a verdict (true or false) to the property can be deduced from the computations.

The initial partition (or abstraction) should at least separate initial states, final states, and others, for correctness reasons [6]. The user can also specify a set of PTSs and of variables that should not be abstracted in the initial partition. This choice usually depends on the property to check, but due to the refinement process a bad choice of these parameters will only slow down the verification of the property. By using a suitable variable ordering in BDDs, the computation of an abstract PTS from a concrete PTS and a partition of the state space of size k can be performed in $\mathcal{O}(k)$ BDD operations, which are in turn linear or quadratic in the number of nodes of the involved BDDs/MTBDDs [6].

If a partition is not detailed enough to deliver a good approximation of the extremum probabilities involved in the property, heuristic strategies are used to refine it into a more detailed partition. They are all based on the stabilization of the partition w.r.t. a transition relation ([7]). Several heuristics are offered to the user, who chooses them when he launches the verification. These strategies do not necessarily stabilize each class w.r.t. the current partition in one step, but to proceed in a more incremental and guided way. They differ on the abstract transition that is used as a basis for the split of a class. The user can also tune the ratio between the refinement and the analysis steps.

Essential states reduction. — We developed an additional reduction technique, the *essential state reduction* [7]. This abstraction, which preserves extremum probabilities, is based on the observation that most transitions in a PTS are non-probabilistic, i.e. they have a unique successor state. To give the intuition of this reduction, suppose that a fragment of a PTS looks like the one depicted in Fig. 5(a). All executions starting from s_1, s_2, s_3, s_4 are leading to the state s_e with probability 1. If we are only interested in probabilities, we can reduce the system by representing all of the above states via the single state s_e, and in addition merge (add) the probabilities for any distribution to enter the states represented by s_e, as shown on Fig. 5(b).

This essential state reduction is applied on the abstract model, just before the generation of the LP problem.

Numerical methods available in the tool. — The previously described reduction techniques are independent from the chosen method for solving the numerical equations. The tool is currently connected to two different LP solver. The first one, LP_SOLVE [2], uses sparse matrix over floating point numbers as its internal representation. Sparse matrices are very useful for our purpose, as the size of the support of the distributions (i.e., the number of successor states of the distributions) is generally very small in our models. However we encountered numerical precision problems with our first case studies, the Bounded Retransmission Protocol [14, 6]. This was not due to proper numerical instability, but to the fact that with IEEE floating point numbers, $1 - 1e20 = 1$!! Very small probabilities indeed appear in this case study, if a great number of retransmissions is allowed. The second one, which is part of the polyhedra library CDDLIB [11], uses dense matrices of rational numbers. It is useful when the above-mentioned problem arises, or in cases where numerical instability is observed and *exact* results are wanted. However, in other case LPSOLVE performs much better: floating point arithmetic is cheap compared to multi-precision rational (integer) arithmetic.

5 Experiments

We have conducted several experiments in order to evaluate our reduction and refinement strategies as well as our implementation.

Bounded Retransmission Protocol. — This protocol, originally studied in [14], is based on the well-known 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. We use the version presented in [6], where probabilities model the possible failures of the two channels used for sending chunks and acknowledgments, respectively. In Table 1 we check the maximum probabilities that the sender does not report a successful transmission. We consider a file composed of either 16 or 64 chunks, and N is the number of allowed retransmissions. We use here the dense matrix based solver with exact arithmetic, because probabilities of very different magnitude order appear in LP problems, which makes the usual floating point arithmetic unstable. The initial partitioning is here performed w.r.t. the explicit control structure of the specification: only variables are abstracted.

The meaning of row labels in the table is the following: `#reach` is the number of states reachable from root states, `#rel` is the number of relevant states after Boolean preprocessing, and `time` is the time needed to build and preprocess the PTS. The three next sets of rows details the refinement process for different upper bounds for $\mathsf{P}^{\mathrm{sup}}$. `#refin` is the number of refinement steps, `#abst` the number of states of the most refined abstract PTS, `#ess` the number of its essential states, `psup` the computed probability, `verd` is the verdict (true or false), and `time(a+r)` gives the time spent in numerical analysis and in refinement process. When the verdict is false, the refinement has gone to the stable partitioning of the PTS and gives the actual $\mathsf{P}^{\mathrm{sup}}$ of the concrete PTS.

Table 1. Results in BRP

		file length 16				file length 64
	MAX	2	4	8	15	15
	#reach.	3908	6060	10364	17896	58024
	#relev.	1014	1790	3342	6058	26362
	time	0.94	1.10	1.59	1.75	5.20
$\leq 10^{-3}$	#refin.	4	5	6	6	6
	#abst.	52	89	161	161	161
	#ess.	24	42	85	85	85
	psup	3.09e-04	4.27e-06	7.89e-06	7.89e-06	7.89e-06
	verd.	T	T	T	T	T
	time(a+r)	0.07+0.88	0.72+0.83	2.96+1.68	2.95+1.72	2.97+2.62
$\leq 10^{-10}$	#refin.	9	10	7	7	7
	#abst.	375	675	242	247	247
	#ess.	152	272	108	115	115
	psup	2.65e-05	2.35e-08	7.01e-12	7.01e-12	7.01e-12
	verd.	F	F	T	T	T
	time(a+r)	0.58+2.56	3.30+5.42	3.15+2.07	3.54+2.33	3.51+3.55
$\leq 10^{-90}$	#refin.	9	10	11	12	16
	#abst.	375	675	1275	2325	9765
	#ess.	152	272	512	932	3908
	psup	2.65e-05	2.35e-08	1.85e-14	3.87e-25	3.87e-25
	verd.	F	F	F	F	F
	time(a+r)	0.58+2.56	3.30+5.42	15.87+11.00	186.58+22.06	1209.72+165.3

Observe the efficiency of Boolean preprocessing and essential states reduction, which gives both a reduction of one third in average. Notice also that it is nearly as easy to prove $\mathsf{P}^{\text{sup}} \leq 10^{-3}$ for big instances of BRP than for small ones: that means that the refinement strategy works well and will not perform too many useless splits. It can also be observed that checking smaller upper bounds can still be performed on very small abstract PTSs, compared to the concrete one, even reduced by preprocessing, and also compared to the stable partitioning (row $\mathsf{P}^{\text{sup}} \leq 10^{-90}$).

The probabilistic dinning philosophers. — In this example, which originates from [19] and has been analysed using PRISM [20], N philosophers are trying to eat. We want to prove a lower bound on the probability for some process to eat after a number of time units specified by value of deadline, with the aditional requirement that a philosopher cannot stay idle for more than K steps. Table 2 shows results for $N = 3$ and different values of K. The chosen deadline corresponds to the smallest one for which the property holds with a probability more than 0.

Here we give in the table not only the number of abstract and essential states, but also in each case the number of abstract distributions. We use the sparse matrix based solver with ordinary floating point arithmetic. The initial partition is chosen to be obtained by abstracting everything but the counter used for the deadline, as it is clear the value of the deadline is of fundamental importance for the studied property. Most of the encouraging observations made for the BRP are still true. The only exception is that essential state reduction does not perform as good as in the BRP. Execution times are much higher, because MTBDDs are much bigger, and the abstract PTSs are much more complex, which results in very big LP problems. Still, refinement remains much cheaper than analysis, and state space reduction between the concrete PTS and the abstract one allowing to prove the property is impressive.

Table 3 compares various refinement options and initial control structures on a particular instance of the system. The first column corresponds to the options that work best and that were used in the previous table: the initial partition detail only the counter for the deadline, and we use n-ary division, giving priority

Table 2. Results in Dining Philosophers with $N = 3$

	K	4	5	6
	deadline	23	27	31
	#reach.	1.00e06	1.97e06	3.40e06
	#relev.	121041	271287	488859
	time	14.4	23.6	34
$\leq \frac{1}{16}$	#refin.	5	7	8
	#abst.	3064/11536	16903/52435	35780/111084
	#ess.	2778/11250	14442/49974	30361/105665
	pinf	0.0625	0.0625	0.0625
	verd.	T	T	T
	time(a+r)	49.6+79.5	2120+590	10353+1462
$\leq \frac{1}{8}$	#refin.	7	8	9
	#abst.	8512/22757	21011/59866	37542/114703
	#ess.	6668/20913	16996/55851	31656/108817
	pinf	0.125	0.125	0.125
	verd.	T	T	T
	time(a+r)	290+220	3683+712	20335+1575

Table 3. Results in Dining Philosophers with $N = K = 3$ and deadline $= 19$

	control	deadline	deadline	deadline	deadline	ctrl. struct.
	option	nary+osl	bin+osl	nary+lso	nary+a	nary+osl
	#reach.	408397				
	#relev.	30018				
	time	6.14				
$\leq \frac{1}{4}$	#refin.	2	2	4	2	4
	#abst.	51/87	35/122	882/2880	140/680	5861/12972
	#ess.	51/87	35/122	827/2825	140/680	4109/11196
	pinf	0.25	0.25	0.25	0.25	0.25
	verd.	T	T	T	T	T
	time(a+r)	0.03+3.85	0.02+3.48	5.16+16.79	0.19+5.09	53.6+44.8

to different types of probabilistic transitions. Using binary divisions gives similar results (second column). Column 3 shows that inverting the priority of the different types of split in column 1 gives very bad results: a much more refined system is needed to prove the property. Last column illustrates the importance of a good initial partition. Here, we generated it according to the explicit control structure of the philosopher, and it produces very bad result.

Binary Exponential Backoff Algorithm in the IEEE 802.3. — This protocol is part of the CSMA/CD protocol and is used to state the policy in which machines retry to access the medium after a collision was detected. The protocol works as follows. After a collision the time is divided into slots. Each of the colliding hosts waits 0 or 1 slots (each with probability 1/2) before retrying to access the medium. If collision happens again each host will wait 0, 1, 2, or 3 slots with probability 1/4. Collision may repeat several times. In its ith collision, a host must choose a waiting time between 0 and $2^i - 1$ with probability $1/2^i$ each. After the Kth collision, it will only choose between 0 and $2^K - 1$, and after the Nth unsuccessful attempt ($N \geq K$), the host will gave up. When no collision happens, the only transmitting host seize the line and its message is transmitted. The appendix shows details on the modelling of the binary exponential backoff method and the property under study using the Rapture modelling language.

The property we study is whether one given host gives up with probability less than or equal to p. Results are reported in Table 4 where 3 hosts are considered, the number of attempts to access the channel before giving up is $N = 5$, and the exponential variable grows until $K = 2$. The check probability p varies and is specified in the table. We have done three types of run: selecting two types of initial partition (see the appendix) and using the default initial partition which ammounts to distinguishing states if they belong to different

Table 4. Results in the Binary Exponential Backoff $N = 5$, $K = 2$

#reach. 752170 #relev. 752170 time 33.14						
Technique		nary+osl	nary+lso+init$_1$	nary+lso+init$_2$	nary+osl+init$_2$	nary+a+init$_2$
$\leq 5 \cdot 10^{-2}$	verd.	T	T	T	T	T
	#refin.	8	7	8	8	7
	#abst.	28237	6678	10099	7720	9994
	#ess.	23326	5616	8409	6535	8391
	psup	0.0299973	0.0161254	0.021849	0.0491828	0.0204969
	time(a+r)	2418.47+1265.01	96.3+555.57	235.67+942.24	124+788.51	212.4+804.83
$\leq 1 \cdot 10^{-2}$	verd.	T	T	T	T	T
	#refin.	9	8	9	10	8
	#abst.	56840	13637	20819	30701	20111
	#ess.	47548	11395	17705	25580	17077
	psup	0.00388816	0.00745022	0.00471054	0.000386098	0.00470079
	time(a+r)	10169.7+2023.94	477.85+980.62	1293.17+1599.61	2423.8+2111.9	1137.99+1432.39
$\leq 5 \cdot 10^{-3}$	verd.	T	T	T	T	T
	#refin.	9	9	9	10	8
	#abst.	56840	28143	20819	30701	20111
	#ess.	47548	24147	17705	25580	17077
	psup	0.00388816	0.00143866	0.00471054	0.000386098	0.00470079
	time(a+r)	10455.7+2140.99	2892.53+1748.86	1296.38+1622	2228.21+2017.41	1244.52+1435.45
$\leq 1 \cdot 10^{-3}$	verd.	T	T	T	T	T
	#refin.	10	10	10	10	9
	#abst.	97642	39209	30701	30701	30701
	#ess.	79655	33449	25580	25580	25580
	psup	0.000386098	0.000386098	0.000386098	0.000386098	0.000386098
	time(a+r)	29182.8+4096.43	5745.09+2464.28	2933.94+2349.45	2288.85+2084.56	2767.8+2177.3
$\leq 5 \cdot 10^{-4}$	verd.	T	T	T	T	T
	#refin.	10	10	10	10	9
	#abst.	97642	39209	30701	30701	30701
	#ess.	79655	33449	25580	25580	25580
	psup	0.000386098	0.000386098	0.000386098	0.000386098	0.000386098
	time(a+r)	28484.9+3922.24	5755.49+2469.5	3005.25+2330.12	2036.3+1865.65	2987.38+2348.35
$\leq 1 \cdot 10^{-4}$	verd.	F	F	F	F	F
	#refin.	10	10	10	10	9
	#abst.	97642	39209	30701	30701	30701
	#ess.	79655	33449	25580	25580	25580
	psup	—	—	—	—	—
	time(a+r)	28297.5+3931.48	5762.32+2470.58	2753.41+2345.52	2241.72+2049.34	2835.37+2240.27

control structure. Again, the importance of selecting a good initial partition is evident. We mention that combinations "lso" and "osl+init$_1$" (not shown on the table) showed instability problems[2]. Table 5 shows the same exercise but with $K = 3$. Both in Table 4 and Table 5 we have highlighted the techniques with best performance. Clearly the refinement priority order "lso" has done worst, but the other two techniques have shown rather incomparable results. The case "a", in which refinement is done w.r.t. all transitions, is faster on refining and spends more time per iteration than the case "osl". On average, this last technique seems to do better.

We carried out the experiments using the sparse matrix based solver with ordinary floating point arithmetic. We tried larger settings, but unfortunately many of them suffered the numerical instability problem. When running the dense matrix based solver with exact arithmetic it required a much larger use of memory which could not be allocated on the machine it was running. It would be very useful for such cases to have a solver using both sparse matrices and exact arithmetic. However, such a solver is still to be implemented!

[2] Refinement can be guided by defining a priority order on the type of transition that should be selected first to partition an equivalence class. The types are: "l" for *looping*, i.e. non-probabilistic transitions that loops on the same abstract state; "s" for *single*, i.e. non-probabilistic transitions leading to a different abstract state; "o" for *others*, namely, the probabilistic transitions.

Table 5. Results in the Binary Exponential Backoff $N = 5$, $K = 3$

#reach. $3.40796 \cdot 10^6$ #relev. $3.40796 \cdot 10^6$ time 111.04				
Technique		nary+lso+init$_2$	nary+osl+init$_2$	nary+a+init$_2$
$\leq 5 \cdot 10^{-2}$	verd.	T	T	T
	#refin.	8	9	6
	#abst.	14877	30657	13509
	#ess.	14088	27191	12939
	psup	0.0168896	0.00637896	0.0190634
	time(a+r)	816.03+1957.2	2893.04+4459.17	563.85+1680.64
$\leq 1 \cdot 10^{-2}$	verd.	T	T	T
	#refin.	9	9	7
	#abst.	32313	30657	27740
	#ess.	27727	27191	23769
	psup	0.00499646	0.00637896	0.00601642
	time(a+r)	3693.23+4046.84	3051.12+4397.42	2687.23+3279.83
$\leq 5 \cdot 10^{-3}$	verd.	T	T	T
	#refin.	9	10	8
	#abst.	32313	61970	55703
	#ess.	27727	53858	48628
	psup	0.00499646	0.00067507	0.00104156
	time(a+r)	3069.24+3585.62	10013.6+6426.98	12976.6+6077.6
$\leq 1 \cdot 10^{-3}$	verd.	T	T	T
	#refin.	10	10	9
	#abst.	66635	61970	97831
	#ess.	58173	53858	85099
	psup	0.000636677	0.00067507	9.60093e-05
	time(a+r)	27220.8+6555.4	10111.1+6754.95	34694.1+10976.3
$\leq 5 \cdot 10^{-4}$	verd.	T	T	T
	#refin.	11	11	9
	#abst.	97831	97831	97831
	#ess.	85099	85099	85099
	psup	9.60093e-05	9.60093e-05	9.60093e-05
	time(a+r)	46629.2+10074	27261.4+9426.13	34293.2+10615.6
$\leq 1 \cdot 10^{-4}$	verd.	T	T	T
	#refin.	11	11	9
	#abst.	97831	97831	97831
	#ess.	85099	85099	85099
	psup	9.60093e-05	9.60093e-05	9.60093e-05
	time(a+r)	45332+9899.48	27216+9408.45	34021.5+10672.9

6 Conclusion

We presented in this paper the probabilistic verification tool Rapture. We described its functionalities and the principles of its verification method. We gave also a set of experimental results that illustrates the behaviour of the implemented techniques and their efficiency.

Acknowledgments We thank the cooperation of Henrik E. Jensen in previous works, helping to develop the foundations of the current tool Rapture.

References

1. R. Bahar, E. Frohm, C. Gaona, G. Hachtel, E. Macii, A. Pardo, and F. Somenzi. Algebraic decision diagrams and their applications. *Formal Methods in System Design*, 10(2/3):171–206, April 1997.
2. M. Berkelaar. LP_Solve: Mixed integer linear program solver. `ftp://ftp.ics.ele.tue.nl/pub/lp_solve`.
3. A. Bianco and L. de Alfaro. Model checking of probabilistic and nondeterministic systems. In *Proceedings of FSTTCS'95*, *LNCS* 1026, 1995.
4. R. E. Bryant. Graph-based algorithms for boolean function manipulation. *IEEE Transactions on Computers*, C-35(8):677–692, 1986.
5. E. M. Clarke, E. A. Emerson, and A. P. Sistla. Automatic verification of finite-state concurrent systems using temporal logic specifications. *ACM TOPLAS*, 8(2), 1986.

6. P.R. D'Argenio, B. Jeannet, H.E. Jensen, and K.G. Larsen. Reachability analysis of probabilistic systems by successive refinements. In *Proceedings of PAPM-PROBMIV 2001*, *LNCS* 2165, Aachen (Germany), September 2001.
7. P.R. D'Argenio, B. Jeannet, H.E. Jensen, and K.G. Larsen. Reduction and refinement strategies for probabilistic analysis. In *Proceedings of PAPM-PROBMIV 2002*, *LNCS*, Copenhagen (Denmark), July 2002.
8. 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 *Proceedings of TACAS'00*, *LNCS* 1785, 2000.
9. Luca de Alfaro. *Formal Verification of Probabilistic Systems.* PhD thesis, Stanford University, 1997.
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. K. Fukuda. Cddlib. `ftp://ftp.ifor.math.ethz.ch/pub/fukuda/cdd`.
12. H.A. Hansson and B. Jonsson. A logic for reasoning about time and reliability. *Formal Aspects of Computing*, 6:512–535, 1994.
13. V. Hartonas-Garmhausen and S. Campos. ProbVerus: Probabilistic symbolic model checking. In *Proceedings of ARTS'99*, *LNCS* 1601, 1999.
14. L. Helmink, M. Sellink, and F. Vaandrager. Proof-checking a data link protocol. In *Proc. International Workshop TYPES'93*, *LNCS* 806, 1994.
15. H. Hermanns, J.-P. Katoen, J. Meyer-Kayser, and M. Siegle. A Markov chain model checker. In *Proceedings of TACAS'00*, *LNCS* 1785, 2000.
16. H. Hermanns, J. Meyer-Kayser, and M. Siegle. Multi terminal binary decision diagrams to represent and analyse continuous time Markov chains. In *Procs. of Int. Workshop on the Numerical Solution of Markov Chains.* Prensas Universitarias de Zaragoza, 1999.
17. C.A.R. Hoare. *Communicating Sequential Processes.* Prentice Hall, 1985.
18. B. Jonsson, K.G. Larsen, and W. Yi. Probabilistic extensions in process algebras. In J.A. Bergstra, A. Ponse, and S. Smolka, editors, *Handbook of Process Algebras.* Elsevier, 2001.
19. D. Lehmann and M. Rabin. On the advantages of free choice: A symmetric fully distributed solution to the dining philosophers problem. In *Proc. 8th Symposium on Principles of Programming Languages*, 1981.
20. PRISM Web Page. `http://www.cs.bham.ac.uk/~dxp/prism/`.
21. M.L. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming.* John Wiley & Sons, 1994.
22. J. P. Queille and J. Sifakis. Specification and verification of concurrent systems in caesar. In *International Symposium on Programming.* LNCS 137, Springer Verlag, April 1982.
23. R. Segala. *Modeling and Verification of Randomized Distributed Real-Time Systems.* PhD thesis, Massachusetts Institute of Technology, 1995.
24. M. Stoelinga and F. Vaandrager. Root contention in IEEE 1394. In *Proceedings of ARTS'99*, *LNCS* 1601, 1999.

Appendix: Details on the Binary Exponential Backoff Method

In the following, we report the Rapture model of the binary exponential backoff algorithm. This algorithm assumes the time is divided in slots. Process `Clock` controls such division. In order to proceed in an orderly fashion, each slot is divided in three sections marked by actions `Tick`, `Tack`, and `Tock`. In the first section (previous to `Tick`), hosts attempt to seize the line. Each host (modelled by process `Host_i`) indicates its will to access the line by incrementing the global variable `chan_req`. The second section (between `Tick` and `Tack`) is the middle of the slot and is used for each attempting host to check if there was a collision (i.e. if `chan_req`>1) and proceed according to the algorithm explained in Section 5. The last section (between `Tack` and `Tock`) is only used to reset variable `chan_req` in order to restart the process.

```
channel
   Tick, Tack, Tock ;

var
   chan_req         : uint ( DIM_HOST ) ;
   line_seized_flag : bool ;             // These two flags are used to
   gave_up_flag     : bool ;             // analyse some properties
                                         // not reported in this article.

process Clock {

  sync
      Tick, Tack, Tock ;

  init ( #start  and  ( chan_req = 0 )
         and  not line_seized_flag  and  not gave_up_flag ) ;

  loc start:
    when true goto begin_slot ;
  loc begin_slot :
    when true sync Tick goto mid_slot ;
  loc mid_slot :
    when true sync Tack goto reset ;
  loc reset :
    when true goto end_slot assign { chan_req := 0 } ;
  loc end_slot :
    when true sync Tock goto begin_slot ;
}
```

Process `Host_i` uses two constants: `MAX_NR_ATTEMPTS`, that represents the maximum number of attempts to access the line (N in Section 5), and `MAX_EXP` that is the maximum exponential value allowed (`MAX_EXP` $= 2^K$ where K is as in Section 5). Variables `nr_attempts` and `exp_val` are used to save the current number of attempts and the current exponential value (from which the random value will be chosen) respectively. Variable `slots_to_wait` contains the slots to wait before attempting to seize the line. Originally, `slots_to_wait` takes a random value uniformly distributed between 0 and `exp_val` $- 1$. This random choice is not straightforward since the RAPTURE modelling language does not allow probabilities to depend on variables. Therefore it is coded in a loop with the help of an auxiliary variable aux_exp. The random setting of slots_to_wait takes place in location process_collision, where the host will iterate ($\log_2$ exp_val) times. In each iteration i, variable slots_to_wait will be incremented by 2^i with probability 0.5. Variable aux_exp carries the appropriate 2^i value.

```
process Host_i {

  sync
       Tick, Tack, Tock ;

  var
       // Be aware that dimensions depend on constants MAX_NR_ATTEMPTS
       // and MAX_EXP
       nr_attempts   : uint ( DIM_TRI ) ;
       exp_val       : uint ( DIM_EXP ) ;
       aux_exp       : uint ( DIM_EXP ) ;
       slots_to_wait : uint ( DIM_EXP ) ;

  init ( #wait_tick  and  (nr_attempts = 0)  and  (exp_val = 1)
          and  (aux_exp = 1)  and  (slots_to_wait = 0) ) ;

  // Begining of the slot: try to seize the line or just wait.
  loc wait_tick :
    when ( slots_to_wait > 0 ) sync Tick
         // Wait these slot
         goto wait_tack
         assign { slots_to_wait := slots_to_wait - 1 } ;
    when ( slots_to_wait = 0 )
         // Do not wait any longer and try to seize the line
         goto init_cycle assign { chan_req := chan_req + 1 };
  loc init_cycle :
    when true sync Tick goto check_collision ;
  loc check_collision :
    when ( chan_req = 1 )
         // The attempt was succesful. Line seized
         goto line_seized
         assign { line_seized_flag := true } ;
    when ( (chan_req > 1)  and  (nr_attempts >= MAX_NR_ATTEMPTS) )
         // There was collision and maximum number of attepmts exceeded.
         // Give up.
         goto gave_up
         assign { gave_up_flag := true } ;
    when ( (chan_req > 1)  and  (nr_attempts < MAX_NR_ATTEMPTS) )
         // The attempt was unsuccesful. A collision occurred.
         // Set values for next attempt.
         goto process_collision
         assign { nr_attempts := nr_attempts + 1 ;
                  aux_exp := 1 ;
                  slots_to_wait := 0 } ;
  // Choose a random value to wait until next attempt.
  loc process_collision :
    when ( (aux_exp > exp_val)  and  (exp_val >= MAX_EXP) )
         goto wait_tack ;
    when ( (aux_exp > exp_val)  and  (exp_val < MAX_EXP) )
         goto wait_tack
         assign { exp_val := 2 * exp_val } ;
    when ( aux_exp <= exp_val )
         goto {
           process_collision .5
             assign { aux_exp := 2 * aux_exp } ;
           process_collision .5
             assign { slots_to_wait := slots_to_wait + aux_exp ;
                      aux_exp := 2 * aux_exp }
         } ;
  // Wait until the middle section of the slot is finished.
  loc wait_tack :
    when true sync Tack goto wait_tock ;
  // Wait until the last section of the slot is finished.
  loc wait_tock :
    when true sync Tock goto wait_tick ;
  loc line_seized :
    when true goto line_seized ;
  loc gave_up :
    when true goto gave_up ;
}
```

The property under study checks the probability of reaching a state in which `Host_0` gives up. The probability of such final condition must be calculated from the beginning of the process, namely when the `Clock` is about to start and no host attempted yet to seize the line (i.e., `chan_req = 0`). The property is stated in Rapture by the following specification lines:

```
initial ( #Clock.start  and  ( chan_req = 0 ) ) ;
final   #Host_0.gave_up ;
```

The initial partition init_1 is specified in Rapture by the sentence

```
control  #Host_0.aux_val, #Host_0.exp_val ;
```

which states that states with different values for variables `aux_val` and `exp_val` must be distinguished in the initial partition. The initial partition init_2 is given by

```
control  #Host_0.exp_val ;
```

In this case, only states with different values for variable `exp_val` must be distinguished.