Verifying Randomized Distributed Algorithms with PRISM*

Marta Kwiatkowska, Gethin Norman, and David Parker

University of Birmingham, Birmingham B15 2TT, United Kingdom
{M.Z.Kwiatkowska,G.Norman,D.A.Parker}@cs.bham.ac.uk

Abstract. In this paper we describe our experience with model checking randomized distributed algorithms using PRISM, a symbolic model checker for concurrent probabilistic systems currently being developed. PRISM uses Multi-Terminal Binary Decision Diagrams (MTBDDs) as supplied by the CUDD package of Fabio Somenzi. Implemented in Java, PRISM has a system description language similar to Reactive Modules and supports model checking of probabilistic temporal logic PCTL (also under fairness constraints). Our experiments indicate that using the BDD variable ordering induced from the Kronecker representation yields very efficient MTBDD representations of randomized distributed algorithms. In particular, we are able to construct models of up to 10^{30} states in seconds. Model checking of 'with probability 1' PCTL properties is also fast. The efficiency of numerical computation with MTBDDs, however, and hence also model checking of quantitative probabilistic temporal logic properties, is still considerably poorer than e.g. for sparse matrices. Descriptions and statistics obtained for several case studies can be found at http://www.cs.bham.ac.uk/~dxp/prism/.

1 Introduction

The use of randomization, i.e. 'electronic coin flipping' and random-number generators, is known to result in elegant, asymmetric and more efficient solutions to a number of network co-ordination problems, for example, mutual exclusion [6] and consensus algorithms [2]. Reasoning about correctness of such algorithms tends to be rather complex [11, 6], particularly in the context of distributed computation, due to the need to combine classical, assertion-based, reasoning with probabilistic analysis. The situation would be substantially improved if one could *automate*, all or in part, the verification process, which could be achieved by extending the techniques of *model checking* to the domain of randomized algorithms.

PRISM (PRobabilistIc Symbolic Model checker) is an experimental symbolic model checker for concurrent probabilistic systems, such as the randomized distributed algorithms mentioned above. It has a state-based system description language similar to Reactive Modules [1], which exhibit both probabilistic and

* Supported in part by EPSRC grant GR/M04617.

nondeterministic behaviour, and allows model checking of formulae of the probabilistic temporal logic PCTL [4, 3] (with fairness). The models are represented as Multi-Terminal Binary Decision Diagrams (MTBDDs) implemented using the Colorado University Decision Diagram (CUDD) package of Fabio Somenzi. This paper describes the outcome of our experiments with PRISM obtained through the modelling and verification of several randomized algorithms. Our main observation is that the use of BDD variable heuristics derived from a Kronecker representation of systems ensures very compact MTBDDs and yields fast model construction and model checking of qualitative properties.

2 Overview of the Tool

2.1 System Description Language

A system is defined as a composition of interacting *modules*. For simplicity, we omit the global variables and synchronization from the following description.

A module is a pair $M = (Var, C)$, where Var is a set of typed, finite-valued *locally controlled variables* (which can be *read* by all modules but *written* to by M only) and C a nondeterministic, guarded probabilistic command. A *local state* of M is a valuation of Var. The set of all type-consistent valuations of Var, denoted $Vals(Var)$, determines the *local state space* of M.

The behaviour of M is specified by the command C, a nondeterministic choice ([] denotes nondeterminism) of *guarded probabilistic updates* of the form

$$g -> \; p_1 : u_1 + \cdots + p_m : u_m$$

In the above, g is a predicate over the variables of all modules in the system (not just the local variables of M), and, for $j = 1, \cdots, m$, p_j are probabilities in the interval (0,1], and each u_j is an *update* assigning a *new* value to some variable $x \in Var$. We use primed variables to distinguish between the new and old values of variables: if $x \in Var$ then x denotes its value at the beginning and x' at the end of an execution step of M. Module M proceeds in single steps, each consisting of a two-phase choice: firstly, one of the enabled guards is selected *nondeterministically*, and then the corresponding *probabilistic update* is executed, which results in a random assignment of a new value to some of the locally controlled variables.

A system $\mathcal{S}$ is given as a set of n modules, $M_i = (Var_i, C_i), i = 1, \cdots, n$, with Var_i pairwise disjoint. The set of variables $\mathcal{V}$ of $\mathcal{S}$ is the union of Var_i. A *global state* is a type-consistent valuation of all the variables $\mathcal{V}$, and thus can be viewed as a vector $s = (s_1, \cdots, s_n)$. The *global state space* is the product $S = \prod_{i=1}^{n} Vals(Var_i)$ of local state spaces.

The behaviour of the system $\mathcal{S}$ is defined in terms of an *asynchronous* concurrent composition of the modules (synchronous composition and action synchronization is also possible, but omitted from this presentation). Formally, the semantics of $\mathcal{S}$ is defined as a concurrent probabilistic system $(S, Steps)$, where S is the set of global states and $Steps : S \longrightarrow 2^{Distr(S)}$ is a mapping from

S to finite nonempty sets of probability distributions on S, each induced by a probabilistic update. Let M_i be a module and (g, u), where $u = p_1 : u_1 + p_2 : u_2 + \cdots + p_m : u_m$ be a guarded probabilistic update within M_i. View each update $u_j : S \longrightarrow S_i$ as a function from the global states to the local states of M_i, and define $En_i(g) = \{s \in S \mid s \models g\}$, the set of all global states enabling the guard g in M_i. For any $s = (s_1, \ldots, s_n) \in En_i(g)$, the *module action* of the probabilistic update u in the global state s can be modelled by the probability distribution $\mu_i^{u,s}$, where for any $t = (t_1, \ldots, t_n) \in S$:

$$\mu_i^{u,s}(t) = \begin{cases} \sum\limits_{\substack{1 \le j \le m \\ t_i = u_j(s)}} p_j & \text{if } t_k = s_k \text{ for all } 1 \le k \ne i \le n \\ 0 & \text{otherwise.} \end{cases}$$

Putting this together, we obtain

$$Steps(t) = \bigcup_{i=1}^{n} Steps_i(t)$$

where $Steps_i(t) = \{\mu_i^{u,s}(t) \mid (g, u) \in M_i, s \in En_i(g)\}$.

2.2 Example

We illustrate the system description language by way of an example. Consider the following:

module M_1
$x : [1..3];$
$[]\ \ (x = 1) \rightarrow 0.8 : (x' = 1) + 0.2 : (x' = 2);$
$[]\ \ (x = 2) \wedge (y = 3) \rightarrow (x' = 2);$
$[]\ \ (x = 2) \wedge (y \ne 3) \rightarrow (x' = 3);$
$[]\ \ (x = 3) \rightarrow 0.5 : (x' = 3) + 0.5 : (x' = 1);$
endmodule

module M_2
$y : [1..3];$
$[]\ \ (y = 1) \rightarrow 0.8 : (y' = 1) + 0.2 : (y' = 2);$
$[]\ \ (y = 2) \wedge (x = 3) \rightarrow (y' = 2);$
$[]\ \ (y = 2) \wedge (x \ne 3) \rightarrow (y' = 3);$
$[]\ \ (y = 3) \rightarrow 0.5 : (y' = 3) + 0.5 : (y' = 1);$
endmodule

It describes a simple system based on two process mutual exclusion. There are two modules, M_1 and M_2. Each has a local state space consisting of three states $\{1, 2, 3\}$, denoted by the local variables x and y respectively. State 3 is the *critical section*. Both processes cannot be in this state at the same time.

The first update of M_1 states that if the module is in state 1, then it will remain in that state with probability 0.8, and move to state 2 with probability

Model:	Breadth-first:		
	States:	NNZ:	Nodes:
mutual 3	$2,368$	$8,272$	$10,317$
mutual 4	$27,600$	$123,883$	$109,096$

Model:	Kronecker:			After reachability:		
	States:	NNZ:	Nodes:	States	NNZ:	Nodes:
mutual 3	$4,096$	$14,119$	$1,063$	$2,368$	$8,272$	$1,632$
mutual 4	$65,536$	$293,119$	$2,916$	$27,600$	$123,883$	$4,179$
mutual 5	$1,048,576$	$5,740,091$	$6,643$	$308,800$	$1,680,086$	$8,510$
mutual 8	4.29×10^9	3.61×10^{10}	$39,989$	3.9×10^8	3.2×10^9	$41,280$
mutual 10	1.1×10^{12}	1.13×10^{13}	$93,933$	4.4×10^{10}	4.41×10^{11}	$90,197$

Fig. 1. Statistics for fully probabilistic models and their MTBDD representation.

0.2. The second and third lines describe the behaviour of M_1 when it is in state 2. This depends on the state of M_2 since the two modules cannot both be in state 3 simultaneously. Hence the guards for these two lines also contain conditions on the variable y. Note that when the update corresponds to a probability distribution which selects a single state with probability 1, the probability is omitted. Module M_2 is identical to M_1, except that it controls y.

The overall system is defined as an asynchronous parallel composition of all the modules. The global state space is the product of all the local state spaces. In the above, the global state space is $\{1, 2, 3\} \times \{1, 2, 3\}$. Consider, for example, the case where the global state is $(1, 2)$, i.e. M_1 is in state 1 and M_2 is in state 2. The behaviour of both modules is defined, as a probability distribution over *local* states, in this state. If selected, M_1 would remain in state 1 with probability 0.8 and move to state 2 with probability 0.2. Likewise, M_2 would move to state 3 with probability 1. We model this situation as a nondeterministic choice between two probability distributions over the *global* state space, one of which selects states $(1, 2)$ and $(2, 2)$ with probability 0.8 and 0.2 respectively, the other selecting state $(1, 3)$ with probability 1. Note that for each nondeterministic choice (i.e. probability distribution), the state of one module can change, but the state of the other cannot.

3 Results

Based on the above representation of a randomized distributed algorithm as a concurrent probabilistic system, we build the corresponding MTBDD representation. We have experimented with two BDD variable orderings, one induced from the breadth-first search of the state space, and the other from a Kronecker representation of the system. For more details of the construc-

Model:	Breadth-first:		
	States:	NNZ:	Nodes:
mutual 3	$2,368$	$20,160$	$12,561$
mutual 4	$27,600$	$471,232$	$146,496$

Model:	Kronecker:			After reachability		
	States:	NNZ:	Nodes:	States	NNZ:	Nodes:
mutual 3	$4,096$	2.92×10^{11}	962	$2,368$	$20,160$	$1,802$
mutual 4	$65,536$	9.35×10^{12}	$1,989$	$27,600$	$471,232$	$4,100$
mutual 5	$1,048,576$	2.99×10^{14}	$3,386$	$308,800$	1.06×10^{7}	$7,149$
mutual 8	4.29×10^{9}	9.8×10^{18}	$9,797$	3.9×10^{8}	1.08×10^{11}	$20,736$
mutual 10	1.1×10^{12}	1.0×10^{22}	$15,921$	4.4×10^{10}	4.89×10^{13}	$33,494$

Fig. 2. Statistics for concurrent models and their MTBDD representation.

Model:	Construction:	Reachability:		Model checking:	
	Time (s):	Time (s):	Iterations:	Time (s):	Iterations:
mutual 3	0.49	0.13	22	0.78	54
mutual 4	0.56	0.59	28	1.59	72
mutual 5	0.69	1.51	34	6.08	90
mutual 8	1.67	10.08	52	40.02	144
mutual 10	3.25	22.58	64	84.38	180

Fig. 3. Times for construction and model checking of fully probabilistic models

tion see [5]. We have modelled and verified a number of algorithms including randomized dining philosophers (Pnueli & Zuck '86, Lehmann & Rabin '82), N-process mutual exclusion (Pnueli & Zuck '86, Rabin '86), and randomized consensus protocol (Aspnes & Herlihy '90). The results are summarised on http://www.cs.bham.ac.uk/~dxp/prism/, with the statistics for the mutual exclusion algorithm in [10] and the property *eventually process 1 enters the critical section* included in this paper.

References

1. R. Alur and T. Henzinger. Reactive modules. In *Proc. 11th Annual IEEE Symposium on Logic in Computer Science (LICS'96)*, pages 207–218. IEEE Computer Society Press, July 1996.
2. J. Aspnes and M. Herlihy. Fast randomized consensus using shared memory. *Journal of Algorithms*, 15(1):441–460, 1990.
3. C. Baier and M. Kwiatkowska. Model checking for a probabilistic branching time logic with fairness. *Distributed Computing*, 11:125–155, 1998.
4. A. Bianco and L. de Alfaro. Model checking of probabilistic and nondeterministic systems. In *Proceedings, FST&TCS*, volume 1026 of *Lecture Notes in Computer Science*, pages 499–513. Springer-Verlag, 1995.
5. L. de Alfaro, M. Kwiatkowska, G. Norman, D. Parker, and R. Segala. Symbolic model checking of concurrent probabilistic systems using MTBDDs and the Kro-

Model:	Construction:	Reachability:		Model checking:	
	Time (s):	Time (s):	Iterations:	Time (s):	Iterations:
mutual 3	0.58	0.13	22	0.26	27
mutual 4	0.67	0.48	28	1.4	36
mutual 5	0.82	1.23	34	4.65	45
mutual 8	1.45	9.02	52	42.07	72
mutual 10	2.27	19.59	64	122.05	90

Fig. 4. Times for construction and model checking of concurrent models

necker representation. In *Proceedings, TACAS'2000*, volume 1785 of *Lecture Notes in Computer Science*. Springer-Verlag, 2000.

6. E. Kushilevitz and M. O. Rabin. Randomized mutual exclusion algorithms revisited. In *Proc. of the ACM Symposium on Principles of Distributed Computing (PODC)*. ACM Press, 1992.
7. D. Lehman and M. Rabin. On the advantage of free choice: a symmetric and fully distributed solution to the Dining Philosophers problem (extended abstract). In *Proceedings, 8th POPL*, pages 133–138. ACM CS Press, 1981.
8. B. Plateau. On the Stochastic Structure of Parallelism and Synchronisation Models for Distributed Algorithms. In *Proc. 1985 ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems*, pages 147–153, May 1985.
9. A. Pnueli and L. Zuck. Verification of multiprocess probabilistic protocols. *Distributed Computing*, 1:53–72, 1986.
10. A. Pnueli and L. Zuck. Probabilistic verification. *Information and Computation*, 103:1–29, 1993.
11. A. Pogosyants, R. Segala, and N. Lynch. Verification of the randomized consensus algorithm of aspnes and herlihy: a case study. In M. Mavronicolas and P. Tsigas, editors, *Proc. WDAG 1997*, volume 1320 of *Lecture Notes in Computer Science*, pages 111–125. Springer-Verlag, 1997.
12. M. O. Rabin. N-process mutual exclusion with bounded waiting by $4{\cdot}\log_2 N$-valued shared variable. *Journal of Computer and System Sciences*, 25:66–75, 1982.