Available online at www.sciencedirect.com
ScienceDirect
Electronic Notes in Theoretical Computer Science
ELSEVIER
Electronic Notes in Theoretical Computer Science 162 (2006) 203–207
www.elsevier.com/locate/entcs

Monte Carlo Methods for Process Algebra

Radu Grosu and Scott A. Smolka

Dept. of Computer Science, Stony Brook Univ., Stony Brook, NY, 11794, USA
E-mail: {grosu,smolka}@cs.sunysb.edu

Abstract

We review the recently developed technique of Monte Carlo model checking and show how it can be applied to the *implementation problem* for I/O Automata. We then consider some open problems in applying Monte Carlo techniques to other process-algebraic problems, such as simulation and bisimulation.

Keywords: Monte Carlo model checking, I/O automata, simulation, bisimulation

1 Introduction

Monte Carlo methods are often used in engineering and computer-science applications to compute an approximation of a solution whose exact computation proves intractable. Example applications include belief updating in Bayesian networks,computing the volume of convex bodies,and approximating the number of solutions of a DNF formula.

Recently, model-checking researchers have turned to Monte Carlo methods in order to cope with the problem of *state explosion*; see, for example, [3,6,8,1]. In this paper, we review the Monte Carlo model-checking algorithm of [1] and show how it can be applied to the *implementation problem* for I/O Automata [4]. We then consider some open problems in applying Monte Carlo techniques to other process-algebraic problems, such as simulation and bisimulation.

2 Monte Carlo Model Checking

Monte Carlo model checking, introduced in [1], is a novel technique that uses random sampling of lassos in a discrete Büchi automaton (BA) to realize a one-sided error, randomized algorithm for LTL model checking. Our approach makes use of the following idea from the automata-theoretic technique of Vardi and Wolper [7] for LTL model checking: given a specification S of a finite-state system and an LTL

doi:10.1016/j.entcs.2006.04.033

formula φ, $S \models \varphi$ (S models φ) if and only if the language of the Büchi automaton $B = B_S \times B_{\neg\varphi}$ is empty. Here B_S is the Büchi automaton representing S's state transition graph, and $B_{\neg\varphi}$ is the Büchi automaton for the negation of φ. Call a cycle reachable from an initial state of B a *lasso*, and say that a lasso is *accepting* if the cycle portion of the lasso contains a final state of B. The presence in B of an accepting lasso means that S is *not* a model of φ. Moreover, such an accepting lasso can be viewed as a *counter-example* to $S \models \varphi$.

The LTL model-checking problem is thus naturally defined in terms of the BA emptiness problem for $B = B_S \times B_{\neg\varphi}$, which reduces to finding accepting lassos in B. Instead of searching the entire state space of B for accepting lassos, we successively generate up to M lassos of B on the fly, by performing uniform random walks in B. If the currently generated lasso is accepting, we have found a counterexample for emptiness, and we stop. The number M of lassos we need to generate depends on to two parameters: the *error margin* ϵ and the *confidence ratio* δ.

To determine M for given ϵ and δ we aim to answer, with confidence $1-\delta$ and within error ϵ, to the following question: *how many independent lassos do we need to generate until one of them is accepting?* The answer is based on a *geometric random variable* X and *statistical hypothesis testing*. The geometric random variable is parameterized by a Bernoulli random variable Z (defined later in this section) that takes value 1 with probability p_Z and value 0 with probability $q_Z = 1 - p_Z$. Intuitively, p_Z is the probability that an arbitrary lasso of B is accepting.

The cumulative distribution function of X for N independent trials of Z is: $F(N) = \mathbf{P}[X \leq N] = 1 - (1 - p_Z)^N$. Requiring that $F(N) = 1 - \delta$ yields: $N = \ln(\delta)/\ln(1 - p_Z)$. Because p_Z is what we want to determine, we assume for the moment that $p_Z \geq \epsilon$. Replacing p_Z with ϵ yields $M = \ln(\delta)/\ln(1 - \epsilon)$ which is greater than N and therefore $\mathbf{P}[X \leq M] \geq \mathbf{P}[X \leq N] = 1 - \delta$. Summarizing:

$$p_Z \geq \epsilon \quad \Rightarrow \quad \mathbf{P}[X \leq M] \geq 1 - \delta \quad \textbf{where} \quad M = \ln(\delta)/\ln(1 - \epsilon) \tag{1}$$

Inequation 1 gives us the minimal number of attempts M needed to achieve success with confidence ratio δ, under the assumption that $p_Z \geq \epsilon$. The standard way of discharging such an assumption is to use *statistical hypothesis testing*.Define the *null hypothesis* H_0 as the assumption that $p_Z \geq \epsilon$. Rewriting inequation 1 with respect to H_0 we obtain:

$$\mathbf{P}[X \leq M \,|\, H_0] \geq 1 - \delta \tag{2}$$

We now perform M trials. If no counterexample is found, i.e., if $X > M$, we reject H_0. This may introduce a type-I error: H_0 may be true even though we did not find a counter-example. However, the probability of making this error is bounded by δ; this is shown in inequation 3 which is obtained by taking the complement of $X \leq M$ in inequation 2:

$$\mathbf{P}[X > M \,|\, H_0] < \delta \tag{3}$$

The Bernoulli random variable Z is associated with a uniform random walk *probability space* $(\mathcal{P}(L), \mathbf{P})$. The *sample space* L is the set of all lassos of B; L_a and L_n are the sets of all accepting and non-accepting lassos of B, respectively.

The *probability* $\mathbf{P}[\sigma]$ of a lasso $\sigma = S_0 e_0 \dots S_{n-1} e_{n-1} S_n$ is defined inductively as follows: $\mathbf{P}[S_0] = k^{-1}$ if $|\mathcal{S}_0| = k$ and $\mathbf{P}[S_0 e_0 \dots S_{n-1} e_{n-1} S_n] = \mathbf{P}[S_0 e_0 \dots S_{n-1}] \cdot \pi[S_{n-1} e_{n-1} S_n]$ where $\pi[S\, e\, S'] = m^{-1}$ if $(S, e, S') \in E$ and $|E(S)| = m$.

Example 2.1 [Probability of lassos] Consider the Büchi automaton B of Figure 1. It contains four lassos, `11`, `1244`, `1231` and `12344`, having probabilities 1/2, 1/4, 1/8 and 1/8, respectively. Lasso `1231` is accepting.

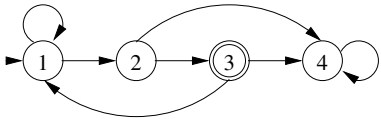

Fig. 1. Example lasso probability space.

Definition 2.2 [Lasso Bernoulli variable] The *random variable* Z associated with the probability space $(\mathcal{P}(L), \mathbf{P})$ of a Büchi automaton B is defined as follows: $p_Z = \mathbf{P}[Z = 1] = \sum_{\lambda_a \in L_a} \mathbf{P}[\lambda_a]$ and $q_Z = \mathbf{P}[Z = 0] = \sum_{\lambda_n \in L_n} \mathbf{P}[\lambda_n]$.

Example 2.3 [Lassos Bernoulli variable] For the Büchi automaton B of Figure 1, the lassos Bernoulli variable has associated probabilities $p_Z = 1/8$ and $q_Z = 7/8$.

Having defined Z, X and H_0, we are now ready to present MC², our Monte Carlo decision procedure for emptiness checking of BA. Its pseudo-code is given below, where `rInit(B)=random(`$\mathcal{S}_0$`)`, `rNext(B,S)=random(E(S))` and `acc(S,B)=(S`$\in$`F)`.

MC² algorithm
input: `B` $= (\mathcal{S}, \mathcal{S}_0, \mathtt{E}, \mathtt{F})$; $0 < \epsilon < 1$; $0 < \delta < 1$.
output: `Either (false, accepting lasso l) or (true, "`$\mathbf{P}[X > M \mid H_0] < \delta$`")`

```
(1)  M := ln δ / ln(1 − ε);
(2)  for (i := 1; i ≤ M; i++) if (RL(B)==(1,l)) return (false,l);
(3)  return (true,"P[X > M | H0] < δ");
```

The main routine consists of three statements, the first of which uses inequation 1 to determine the value for M, given parameters ϵ and δ. The second statement is a for-loop that successively samples up to M lassos by calling the *random lasso* (RL) routine. If an accepting lasso l is found, MC² decides false and returns l as a counter-example. If no accepting lasso is found within M trials, MC² decides true, and reports that with probability less than δ, $p_Z > \epsilon$.

The RL routine generates a random lasso by using the *randomized init* (`rInit`) and *randomized next* (`rNext`) routines. To determine if the generated lasso is accepting, it stores the index `i` of each encountered state `s` in `HashTbl` and records the index of the most recently encountered accepting state in variable `f`. Upon detecting a cycle, i.e., the state `s := rNext(B,s)` is in `HashTbl`, it checks if `HashTbl(s)` $\leq$ `f`; the cycle is an accepting cycle if and only if this is the case. The function `lasso()` extracts a lasso from the states stored in `HashTbl`.

Given a succinct representation S of a Büchi automaton B, one can avoid the explicit construction of B, by generating random states `rInit(B)` and `rNext(B,s)` on demand and performing the test for acceptance `acc(B,s)` symbolically.

MC² is very efficient. It runs in time $O(MD)$ and uses $O(D)$ space, where M is optimal and D is B's *recurrence diameter* (longest loop-free path starting from an initial state).

3 The Implementation Problem for I/O Automata

An I/O Automaton (IOA) is a finite-state automaton whose transitions are associated with named *actions*, which are classified as *input*, *output*, or *internal*. Input and output actions are used for communication with the automaton's environment, whereas internal actions are visible only to the automaton itself. The input actions are assumed not to be under the automaton's control (IOA are *input-enabled*, whereas the automaton itself controls which output and internal actions should be performed. See [4] for the formal definition.

The *implementation problem* for I/O Automata (IOA) is the following. Given IOA A and B, representing the implementation and specification of the system under investigation, does A implement B ($A \leq B$)? Now, $A \leq B$ holds if $L(A) \subseteq L(B)$; that is, the *traces* of A are a subset of the traces of B. This in turn is equivalent to $L(A \times \overline{B}) = \emptyset$, where $\overline{B}$ is the complement of B. Intuitively, if every observable behavior of A is an observable behavior of B then no observable behavior of A is an observable behavior of $\overline{B}$.

Specification IOA B can be viewed as a (input-enabled) Büchi automaton by treating a subset of its states as accepting. IOA A can similarly be viewed as a BA (all of whose states are accepting). Consequently, the IOA implementation problem can be reduced to the *language emptiness* problem for BA, and the MC² Monte Carlo algorithm can be directly applied. A recent paper [2] suggests how this can all be extended to the case of *Timed* I/O Automata.

4 Open Problems

It would be interesting to extend our Monte Carlo approach to the model-checking problem for branching-time temporal logics, such as CTL, the modal mu-calculus, and Hennessy-Milner logic. This extension appears to be non-trivial since the idea of sampling accepting lassos in the product graph will no longer suffice. For the similar reasons, the problem of applying Monte Carlo methods in deciding simulation [5] and bisimulation remains open.

References

[1] R. Grosu and S. A. Smolka. Monte Carlo model checking. In *Proceedings of TACAS 2005*. Springer-Verlag, 2005.

[2] R. Grosu, S. A. Smolka, W. Tan, A. Bouajjani, M. D. Bozga, and S. Tripakis. Monte Carlo model checking of timed automata, 2005. Submitted for publication.

[3] T Hérault, R. Lassaigne, F. Magniette, and S. Peyronnet. Approximate probabilistic model checking. In *Proc. Fifth International Conference on Verification, Model Checking, and Abstract Interpretation (VMCAI 2004)*, 2004.

[4] N. Lynch and M. Tuttle. An introduction to input/output automata. *CWI Quarterly*, 2(3):219–246, 1989.

[5] N. Lynch and F. Vaandrager. Forward and backward simulations I: untimed systems. *Inf. Comput.*, 121(2):214–233, 1995.

[6] K. Sen, M. Viswanathan, and G. Agha. On statistical model checking of stochastic systems. In K. Etessami and S. Rajamani, editors, *Proc, of the 17th International Conference on Computer Aided Verification*, volume 3576 of *LNCS*. Springer, 2005.

[7] M. Vardi and P. Wolper. An automata-theoretic approach to automatic program verification. In *Proc. IEEE Symposium on Logic in Computer Science*, pages 332–344, 1986.

[8] H.L.S. Younes. Probabilistic verification for black-box systems. In K. Etessami and S. Rajamani, editors, *Proc. of the 17th International Conference on Computer Aided Verification*, volume 3576 of *LNCS*, pages 253–265. Springer, 2005.