Uniform Monte-Carlo Model Checking

Conference Paper · March 2011

DOI: 10.1007/978-3-642-19811-3_10 · Source: DBLP

5 authors, including:

Johan Oudinet
Technische Universität München
12 PUBLICATIONS **195** CITATIONS
SEE PROFILE

Alain Denise
Université Paris-Saclay
106 PUBLICATIONS **2,719** CITATIONS
SEE PROFILE

Marie-Claude Gaudel
Laboratoire de Recherche en Informatique
116 PUBLICATIONS **1,882** CITATIONS
SEE PROFILE

Richard Lassaigne
Paris Diderot University
45 PUBLICATIONS **861** CITATIONS
SEE PROFILE

Uniform Monte-Carlo Model Checking

Johan Oudinet[1,2], Alain Denise[1,2,3], Marie-Claude Gaudel[1,2], Richard Lassaigne[4,5], and Sylvain Peyronnet[1,2,3]

[1] Univ Paris-Sud, Laboratoire LRI, UMR8623, Orsay, F-91405
[2] CNRS, Orsay, F-91405
[3] INRIA Saclay - Île-de-France, F-91893 Orsay Cedex
[4] Univ. Paris VII, Equipe de Logique Mathématique, UMR7056
[5] CNRS, Paris-Centre, F-75000

Abstract. Grosu and Smolka have proposed a randomised Monte-Carlo algorithm for LTL model-checking. Their method is based on random exploration of the intersection of the model and of the Büchi automaton that represents the property to be checked. The targets of this exploration are so-called *lassos*, i.e. elementary paths followed by elementary circuits. During this exploration outgoing transitions are chosen uniformly at random.

Grosu and Smolka note that, depending on the topology, the uniform choice of outgoing transitions may lead to very low probabilities of some lassos. In such cases, very big numbers of random walks are required to reach an acceptable coverage of lassos, and thus a good probability either of satisfaction of the property or of discovery of a counter-example. In this paper, we propose an alternative sampling strategy for lassos in the line of the uniform exploration of models presented in some previous work.

The problem of finding all elementary cycles in a directed graph is known to be difficult: there is no hope for a polynomial time algorithm. Therefore, we consider a well-known sub-class of directed graphs, namely the reducible flow graphs, which correspond to well-structured programs and most control-command systems.

We propose an efficient algorithm for counting and generating uniformly lassos in reducible flowgraphs. This algorithm has been implemented and experimented on a pathological example. We compare the lasso coverages obtained with our new uniform method and with uniform choice among the outgoing transitions.

1 Introduction

Random exploration of large models is one of the ways of fighting the state explosion problem. In [11], Grosu and Smolka have proposed a randomized Monte-Carlo algorithm for LTL model-checking together with an implementation called MC^2. Given a finite model M and an LTL formula Φ, their algorithm performs random walks ending by a cycle, (the resulting paths are called *lassos*) in the Büchi automaton $B = B_M \times B_{\neg \Phi}$ to decide whether $L(B) = \emptyset$ with a probability which depends on the number of performed random walks. More precisely,

D. Giannakopoulou and F. Orejas (Eds.): FASE 2011, LNCS 6603, pp. 127–140, 2011.

their algorithm samples lassos in the automaton B until an *accepting* lasso is found or a fixed bound on the number of sampled lassos is reached. If MC^2 find an accepting lasso, it means that the target property is false (by construction of B, an accepting lasso is a counterexample to the property). On the contrary, if the algorithm stops without finding an accepting lasso, then the probability that the formula is true is high. The advantage of a tool such as MC^2 is that it is fast, memory-efficient, and scalable.

Random exploration of a model is a classical approach in simulation [3] and testing (see for instance [26,6]) and more recently in model-checking [21,8,20,1]. A usual way to explore a model at random is to use isotropic random walks: given the states of the model and their successors, an isotropic random walk is a succession of states where at each step, the next state is drawn uniformly at random among the successors, or, as in [11] the next transition is drawn uniformly at random among the outgoing transitions. This approach is easy to implement and only requires local knowledge of the model.

However, as noted in [21] and [19], isotropic exploration may lead to bad coverage of the model in case of irregular topology of the underlying transition graph. Moreover, for the same reason, it is generally not possible to get any estimation of the coverage obtained after one or several random walks: it would require some complex global analysis of the topology of the model.

Not surprisingly, it is also the case when trying to cover lassos: Figure 1 from [11], shows a Büchi automaton with $q + 1$ lassos: $l_0, l_1, \ldots, l_q$ where l_i is the lasso $s_0 s_1 \ldots s_i s_0$. With an isotropic random walk, l_q has probability $1/2^q$ to be traversed.

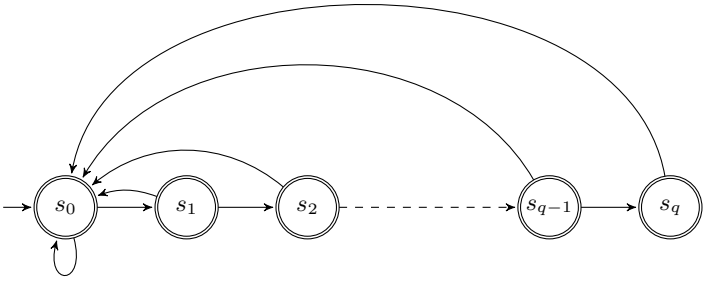

Fig. 1. A pathological example for Büchi automata

In this paper, we propose an alternative sampling strategy for lassos in the line of the uniform exploration of models presented in [7,17] and [10]. In the example above, the low probability of l_q comes from the choice done by the random walk at each state. It has to choose between a state that leads to a single lasso and a state that leads to an exponential number of lassos. In the case of an isotropic random walk, those two states have the same probability. If the number of lassos that start from each state is known, the choice of the successors can be guided to balance the probability of lassos so as to get a uniform distribution and to avoid

lassos with a too small probability. Coming back to Figure 1, with a uniform distribution on lassos, l_q has probability $1/q$ to be traversed instead of $1/2^q$.

However, the problem of counting and finding all elementary cycles in a directed graph is known to be difficult. We briefly recall in Section 2 why there is no hope of polynomial time algorithms for this problem. Therefore, we consider a well-known sub-class of directed graphs, namely the reducible flow graphs [12], which correspond to well-structured programs and most control-command systems. In Section 3, we show that the set of lassos in such graphs is exactly the set of paths that start from the initial state and that end just after the first back edge encountered during a depth-first search. Then on the basis of the methods for counting and generating paths uniformly at random, presented in [7,17] and [10], we give an algorithm for counting the number of lassos in a reducible flow-graph and uniformly generating random lassos in reducible flowgraphs. Section 4 presents how this algorithm can be used for LTL model-checking. Section 5 reports how this algorithm has been implemented and how it has been experimented on an example similar to the pathological example in Figure 1. We compare the lasso coverages obtained with our new uniform method and with uniform choice among the successors or the outgoing transitions.

2 Counting and Generating Lassos in Directed Graphs

A lasso in a graph is a finite path followed by an elementary, or simple, cycle. There are two enumeration problems for elementary cycles in a graph. The first one is counting and the second one is finding all such cycles. These two problems are hard to compute: let FP be the class of functions computable by a Turing machine running in polynomial time and $\sharp CYCLE(G)$ be the number of elementary cycles in a graph G; one can prove that $\sharp CYCLE(G) \in FP$ implies $P = NP$ by reducing the Hamiltonian circuit problem to decide if the number of elementary cycles in a graph is large.

For the problem of finding all elementary cycles in a directed graph there is no hope for a polynomial time algorithm. For example, the number of elementary cycles in a complete directed graph grow faster than the exponential of the number of vertices. Several algorithms were designed for the finding problem. In the algorithms of Tiernan [23] and Weinblatt [25] time exponential in the size of the graph may elapse between the output of a cycle and the next. However, one can obtain enumeration algorithms with a polynomial delay between the output of two consecutive cycles.

Let G be a graph with n vertices, e edges and c elementary cycles. Tarjan [22] presented a variation of Tiernan's algorithm in which at most $O(n.e)$ time elapses between the output of two cycles in sequence, giving a bound of $O(n.e(c+1))$ for the running time of the algorithm. To our knowledge, the best algorithm for the finding problem is Johnson's [14], in which time consumed between the output of two consecutive cycles as well as before the first and after the last cycle never exceeds the size of the graph, resulting in bounds $O((n+e).(c+1))$ for time and $O(n+e)$ for space.

Because of the complexity of these problems in general graphs, we consider in this paper a well-known sub-class of directed graphs, namely the reducible flow graphs [12]. Control graphs of well-structured programs are reducible. Most data-flow analysis algorithms assume that the analysed programs satisfy this property, plus the fact that there is a unique final vertex reachable from any other vertex. Similarly, well-structured control-command systems correspond to reducible dataflow graphs. But in their case, any vertex is considered as final: this makes it possible the generalisation of data-flow analysis and slicing techniques to such systems [15]. Informally, the requirement on reducible graphs is that any cycle has a unique entry vertex.

It means that a large class of critical systems correspond to such flowgraphs. However, arbitrary multi-threaded programs don't. But in many cases where there are some constraints on synchronisations, for instance in cycle-driven systems, reducibility is satisfied.

The precise definition of reducibility is given below, as well as a method for counting and uniformly generating lassos in such graphs.

3 Uniform Random Generation of Lassos in Reducible Flowgraphs

A *flowgraph* $G = (V, E)$ is a graph where any vertex of G is reachable from a particular vertex of the graph called the source (we denote this vertex by s in the following). From a flowgraph G one can extract a spanning subgraph (e.g. a directed rooted tree whose vertex set is also V) with s as root. This spanning subgraph is known as *directed rooted spanning tree* (DRST). In the specific case where a depth-first search on the flowgraph and its DRST lead to the same order over the set of vertices, we call the DRST a *depth-first search tree* (DFST). The set of back edges is denotes by B_E. Given a DFST of G, we call *back edge* any edge of G that goes from a vertex to one of its ancestors in the DFST. And we say that a vertex u *dominates* a vertex v if every path from s to v in G crosses u.

Here we focus on *reducible flowgraphs*. The intuition for reducible flowgraphs is that any loop of such a flowgraph has a unique entry, that is a unique edge from a vertex exterior to the loop to a vertex in the loop. This notion has been extensively studied (see for instance [12]). The following proposition summarizes equivalent definitions of reducible flowgraphs.

Proposition 1. *All the following items are equivalent:*

1. $G = (V, E, s)$ *is a reducible flowgraph.*
2. *Every DFS on* G *starting at* s *determines the same back edge set.*
3. *The directed acyclic graph (DAG)* $dag(G) = (V, E - B_E, s)$ *is unique.*
4. *For every* $(u, v) \in B_E$, v *dominates* u.
5. *Every cycle of* G *has a vertex which dominates the other vertices of the cycle.*

We now recall the precise definition of lassos:

Definition 1 (lassos). *Given a flowgraph $G = (V, E, s)$, a path in G is a final sequence of vertices $s_0, s_1, \ldots, s_n$ such that $s_0 = s$ and $\forall 0 \leq i < n \quad (s_i, s_{i+1}) \in E$. An elementary path is a path such that the vertices are pairwise distinct. A lasso is a path such that $s_0, \ldots, s_{n-1}$ is an elementary path and $s_n = s_i$ for some $0 \leq i < n$.*

We can now state our first result on lassos in reducible flowgraphs.

Proposition 2. *Any lasso of a reducible flowgraph ends with a back edge, and any back edge is the last edge of a lasso.*

Proof. Suppose a traversal starting from the source vertex goes through a lasso and finds a back edge before the end of the lasso. Since the graph is a reducible flowgraph this means that we have a domination, thus we see two times the same vertex in the lasso, which is a contradiction with the fact that a lasso is elementary (remember a lasso is an elementary path).

Suppose now that the traversal never sees a back edge. Then no vertex has been seen twice, thus the traversal is not a lasso.

Using this proposition, we can now design a simple algorithm for

- counting the number of lassos in a reducible flowgraph,
- uniformly generating random lassos in a reducible flowgraph.

Here uniformly means equiprobably. In other word any lasso has the same probability to be generated as the others.

Following the previous proposition, the set of lassos is exactly the set of paths that start from the source and that end just after the first back edge encountered. At first, we change the problem of generating lassos into a problem of generating paths of a given size n, by slightly changing the graph: we add a new vertex s_0 with a loop, that is an edge from itself to itself, and edge from s_0 to s. And we give n a value that is an upper bound of the length of the lassos in the new graph. A straightforward such value is the depth of the depth-first search tree plus one.

Now for any vertex u, let us denote $f_u(k)$ the number of paths of length k starting from vertex u and finish just after the first back edge encountered. The edge (s_0, s_0) is not considered as a back edge because it does not finish a lasso in the initial graph G. The number of lassos is given by $f_{s_0}(n)$. And we have the following recurrence:

- $f_{s_0}(1) = 0$.
- $f_u(1) =$ number of back edges from u if $u \neq s_0$.
- $f_u(k) = \sum_{(u,v) \in E'} f_v(k-1)$ for $k > 1$ where E' is the set of edges of the graph, except the back edges.

The principle of the generation process is simple: starting from s, draw a path step by step. At each step, the process consists in choosing a successor of the current vertex and going to it. The problem is to proceed in such a way that

only (and all) lassos of length n can be generated, and that they are equiprobably distributed. This is done by choosing successors with suitable probabilities. Suppose that, at one given step of the generation, we are on state u, which has k successors denoted $v_1, v_2, \ldots, v_k$. In addition, suppose that $m > 0$ transitions remain to be crossed in order to get a lasso of length n. Then the condition for uniformity is that the probability of choosing state v_i $(1 \leq i \leq k)$ equals $f_{v_i}(m-1)/f_u(m)$. In other words, the probability to go to any successor of u must be proportional to the number of lassos of suitable length from this successor.

Computing the numbers $f_u(i)$ for any $0 \leq i \leq n$ and any state u of the graph can be done by using the recurrence rules above.

Table 1 presents the recurrence rules which correspond to the automaton of Figure 2.

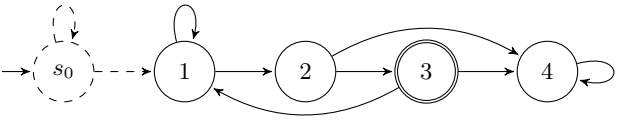

Fig. 2. An example of a Büchi automaton, from [11]. We have added the vertex s_0 and its incident edges, according to our procedure for generating lassos of length $\leq n$ for any given n.

Table 1. Recurrences for the $f_i(k)$

$$
\begin{array}{lll}
f_{s_0}(1) = f_2(1) = 0 & & \\
f_1(1) = f_3(1) = f_4(1) = 1 & & \\
f_{s_0}(k) = f_{s_0}(k-1) + f_1(k-1) & (k>1) \\
f_1(k) = f_2(k-1) & (k>1) \\
f_2(k) = f_3(k-1) + f_4(k-1) & (k>1) \\
f_3(k) = f_4(k-1) & (k>1) \\
f_4(k) = 0 & (k>1)
\end{array}
$$

Now a primitive generation scheme is as follows:

- Preprocessing stage: Compute a table of the $f_u(i)$'s for all $0 \leq i \leq n$ and any state u.
- Generation stage: Draw the lassos according to the scheme seen above.

Note that the preprocessing stage must be done only once, whatever the number of lassos to be generated. Easy computations show that the memory space requirement is $O(n \times |V|)$ integer numbers, where $|V|$ denotes the number of vertices in G. The number of arithmetic operations needed for the preprocessing stage is in the worst case in $O(nd|V|)$, where d stands for the maximum number of transitions from a state; and the generation stage is $O(nd)$ [13]. However,

the memory required is too large for very long lassos. In [18] we presented an improved version, named Dichopile, which avoids to have the whole table in memory, leading to a space requirement in $O(|V| \log n)$, at the price of a time requirement in $O(nd|V| \log n)$.

4 Application to LTL Model-Checking

This section shows how to use this generation algorithm (and its variants, see [18]) for LTL randomised model-checking. We note M and Φ the considered model and LTL formula. We propose a randomised method to check whether $M \models \Phi$. We call B_M a Büchi automaton corresponding to M: the transitions are labeled, the underlying graph is assumed to be reducible, and all the states are accepting (but this condition can be relaxed). We call $B_{\neg\,\Phi}$ the Büchi automaton corresponding to the negation of Φ. The problem is to check that $L(B) = \emptyset$ where $B = B_M \times B_{\neg\,\Phi}$.

It is possible to avoid the construction of the product $B = B_M \times B_{\neg\,\Phi}$. The idea is to exploit the fact that this product is the result of a total synchronisation between B_M and $B_{\neg\,\Phi}$: as explained in [4], the behaviours of B are exactly those behaviours of B_M accepted by $B_{\neg\,\Phi}$. It means that a lasso in B corresponds to a lasso in B_M (but not the reverse). Therefore, it is possible to draw lassos from B_M and, using $B_{\neg\,\Phi}$ as an observer, to reject those lassos that are not in B. It is well-known that such rejections preserve uniformity in the considered subset [16].

4.1 Drawing Lassos in B

There is a pre-processing phase that contains the one described in Section 3, namely:

(pre-DFS) the collection of the set B_E of back edges via a DFS in B_M, and computing n, the depth of the DFS + 1;

(pre-vector) the construction of the vector of the $|V|$ values $f_u(1)$, i.e. the numbers of lassos of length 1 starting for every vertex u;

The two steps above are only needed once for each model. They are independent of the properties to be checked. The third step below is dependent on the property:

(pre-construction of the negation automaton) the construction of the Büchi automaton $B_{\neg\,\Phi}$.

Lassos are drawn from B_M using the Dichopile algorithm [18] and then observed with $B_{\neg\,\Phi}$ to check whether they are lassos of B. Moreover, it is also checked whether an acceptance state of $B_{\neg\,\Phi}$ is traversed during the cycle. The observation may yield three possible results:

- the lasso is not a lasso in $B_M \times B_{\neg\,\Phi}$;
- the lasso is an accepting lasso in $B_M \times B_{\neg\,\Phi}$;
- the lasso is a non-accepting lasso in $B_M \times B_{\neg\,\Phi}$.

Observation of lassos
The principle of the observation algorithm is: given a lasso of B_M, and the $B_{\neg\,\Phi}$ automaton, the algorithm explores $B_{\neg\,\Phi}$ guided by the lasso: since $B_{\neg\,\Phi}$ is generally non deterministic, the algorithm performs a traversal of a tree made of prefixes of the lasso.

When progressing in $B_{\neg\,\Phi}$ along paths of this tree, the algorithm notes whether the state where the cycle of the lasso starts and comes back has been traversed; when it has been done, it notes whether an accepting state of the automaton is met.

The algorithm terminates either when it reaches the end of the lasso or when it fails to reach it: it is blocked after having explored all the strict prefixes of the lasso present in $B_{\neg\,\Phi}$. The first case means that the lasso is also a lasso of $B_M \times B_{\neg\,\Phi}$; then if an accepting state of $B_{\neg\,\Phi}$ has been seen in the cycle of the lasso, it is an accepting lasso. Otherwise it is a non-accepting lasso. The second case means that the lasso is not a lasso in $B_M \times B_{\neg\,\Phi}$.

As soon as an accepting lasso is found, the drawing is stopped.

4.2 Complexities

Given a formula Φ and a model M, let $|\Phi|$ the length of formula Φ, $|V|$ the number of states of B_M and $|E|$ its number of transitions, the complexities of the pre-processing treatments are the following:

(pre-DFS) this DFS is performed in B_M; it is $O(|E|)$ in time and $O(|V|)$ in space in the worst case;

(pre-vector) the construction of the vector of the $f_u(1)$ is $\Theta(|V|)$ in time and space;

(pre-construction of the negation automaton) the construction of $B_{\neg\,\Phi}$ is $O(2^{|\Phi|.\log|\Phi|})$ in time and space in the worst case [24].

The drawing of one lasso of length n in B_M using the Dichopile algorithm is $O(n.d.|V|.\log n)$ in time and $O(|V|.\log n)$ in space; then its observation is a DFS of maximum depth n in a graph whose size can reach $O(2^{|\Phi|)}$. Let $d_{B_{\neg\,\Phi}}$ the maximal out-degree of $B_{\neg\,\Phi}$, the worst case time complexity is $min(d^n_{B_{\neg\,\Phi}}, 2^{|\Phi|})$, and the space complexity is $O(n)$.

The main motivation for randomised model-checking is gain in space. With this respect, using isotropic random walks as in [11] is quite satisfactory since a local knowledge of the model is sufficient. However, it may lead to bad coverage of the model. For instance, [11] reports the case of the Needham-Schroeder protocol where a counter-exemple has a very low probability to be covered by isotropic random walk. The solution presented here avoids this problem, since it ensures a uniform drawing of lassos, but it requires more memory: $\Theta(|V|)$ for the pre-processing and $O(|V|.\log n)$ for the generation.

We can also compare the complexity of our approach to the complexity of practical algorithm for the model checking of LTL. The NDFS algorithm [5] has a space complexity of $O(r\log|V|)$ for the main (randomly accessed) memory,

where r is the number of reachable states. This complexity is obtained thanks to the use of hash tables. Information accessed in a more structured way (e.g. sequentially) are stored on an external memory and retrieved using prediction and cache to lower the access cost.

In this case, uniform sampling of lassos, with a space complexity of $O(|V| \log n)$ is close to the $O(r \log |V|)$ of the classic NDFS algorithm, without being exhaustive. Nevertheless, a randomized search has no bias in the sequence of nodes it traverses (as is NDFS), and may lead faster to a counterexample in practice.

4.3 Probabilities

Let $lassos_B$ the number of lassos in B, and $lassos_{B_M}$ the number of lassos in B_M, i. e. $lassos_{B_M} = f_s(n)$, we have $lassos_B \leq f_s(n)$. The probability for a lasso to be rejected by the observation is $lassos_B / f_s(n)$, where the value of $lassos_B$ is dependent on the considered LTL formula. Thus, the average time complexity for drawing N lassos in B is the complexity of the pre-processing, which is $O(|E|) + O(2^{|\Phi|}))$ plus $N \times f_s(n)/lassos_B$ [16] times the complexity of drawing one lasso in B_M and observing it, which is $O(n \log n|V|) + min(d^n_{B_{\neg \Phi}}, 2^{|\Phi|})$.

Since the drawing in B_M is uniform, and $f_s(n)$ is the number of lassos the probability to draw any lasso in B_M is $1/f_s(n)$. Since there are less lassos in B, and rejection preserves uniformity [16], the probability to draw any lasso in B is $1/lassos_B$ which is greater or equal to $1/f_s(n)$. It is true for any lasso in B, the accepting or the non accepting ones. Thus there is no accepting lasso with too low probability as it was the case in the Needham-Schroeder example in [11].

Moreover, the probability ρ that $M \models \Phi$ after N drawings of non-accepting lassos is greater than:

$$1 - \big(1 - 1/f_s(n)\big)^N$$

Increasing N may lead to high probabilities. Conversely, the choice of N may be determined by a target probability ρ:

$$N \geq \frac{\log(1-\rho)}{\log(1 - 1/f_s(n))} \tag{1}$$

Remark: A natural improvement of the method is to use the fact that during the preliminary DFS some lassos of B_M are discovered, namely one by back edge. These lassos can be checked as above for early discovery of accepting lassos in $B_M \times B_{\neg\phi}$. However, it is difficult to state general results on the gain in time and space, since this gain is highly dependent on the topology of the graph.

5 Experimental Results

In this section, we report results about first experiments, which use the algorithm presented in Section 4 to verify if an LTL property holds on some models. We

first chose a model in which it is difficult to find a counter-example with isotropic random walks. The idea is to evaluate the cost-effectiveness of our algorithm: does it find a counter-example in a reasonable amount of time and memory? How many lassos have to be checked before we get a high probability that $M \models \Phi$? [2].

5.1 Implementation and Methodology

This algorithm has been implemented using the RUKIA library[1]. This C++ library proposes several algorithms to generate uniformly at random paths in automata. In particular, the Dichopile algorithm that is mentioned in Section 3. We also use several tools that we mention here: The Boost Graph Library (BGL) for classical graph algorithms like the depth-first search algorithm; The GNU Multiple Precision (GMP) and Boost.Random libraries for generating random numbers; The LTL2BA software [9] to build a Büchi automaton from a LTL formula.

We did all our experiments on a dedicated server whose hardware is composed of an Intel Xeon 2.80GHz processor with 16GB memory. Each experiment was performed 5 times. The two extreme values are discarded and the three remaining values are averaged.

5.2 Description of the Model and the Formula

Figure 3 shows B_M, the Büchi automaton of the model. It has q states and every state, except s_q, has two transitions labeled by the action a: stay in the current state or move to the next state. The state s_q can only do the action ¬a.

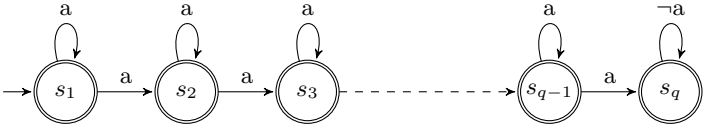

Fig. 3. The Büchi automaton, B_M, of the model. Its transitions are labeled by a or ¬a, to indicate if the action a occurs or not.

The property that we want to check on this model is: "an action a should occur infinitely often". In LTL, this property can be expressed as

$$\phi = GFa,$$

where the operator G means "for all" and the operator F means "in the future".

It is clear that $M \not\models \Phi$ because if s_q is reached, then no a will occur. Hence, there is a behavior in M where the action a will not occur infinitely often. However, it is difficult for an isotropic random walk to find the lasso that traverses s_q. In the next section, we measure the difficulty to find this lasso with our algorithm.

[1] `http://rukia.lri.fr`

5.3 LTL Model-Checking with Uniform Generation of Lassos

The first step of the algorithm is to build a Büchi automaton that represents the formula $\neg\phi$ (here, $\neg\phi = FG\neg a$). We used the tool LTL2BA and got the automaton in Figure 4.

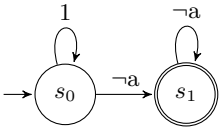

Fig. 4. The Büchi automaton, $B_{\neg\phi}$, of the formula $\neg\phi$. The label 1 means that both a and ¬a are accepted.

Then, we can apply our algorithm to generate lassos in B_M until we find an accepting lasso in $B_M \cap B_{\neg\phi}$, by observing the lasso of B_M with the automaton $B_{\neg\phi}$. Table 2 shows the time needed to our algorithm to find the counter-example in two versions of the automaton in Figure 3, one with $q = 100$ states and the other with $q = 1000$ states. We also tried to find this counter-example with isotropic random walks. It did not work, even after the generation of 2 billions lassos. Thus, using uniform generation of lassos provides a better detection power of counter-examples.

Table 2. $M \not\models \Phi$: elapsed time, used memory and numbers of lassos generated in B_M by the algorithm of Section 4 to find the counter-example of Section 5.2

# states	Time (s)	Mem (MiB)	Nb lassos
100	0.38	49	70
1000	546	50	680

As we know the probability to find the counter-example in B with both isotropic random walks and uniform random generation (i.e., $1/2^q$ for an isotropic random walk and $1/q$ for a uniform random generation), we can compute the number of lassos required to achieve a target probability ρ. Table 3 describes those numbers for some target probabilities and for the two previous versions of the automaton in Figure 3.

Note: In general, the minimal probability to find a counter example is unknown. In the case of uniform random generation of lassos, we have a lower bound of this probability. Thus, the maximal number of lassos to be generated for a given probability can always be determined with Formula 1, which it is not possible with isotropic random walks.

Table 3. $M \models \Phi$: numbers N of lassos to be generated with isotropic random walks (resp. uniform random generation) in order to ensure a probability ρ that $M \models \Phi$. The symbol ∞ means a huge number, which cannot be computed with a calculator.

		N	
# states	ρ	isotropic	uniform
100	0.9	10^{30}	227
	0.99	∞	454
	0.999	∞	681
1000	0.9	∞	2300
	0.99	∞	4599
10000	0.9	∞	23022

6 Conclusion

We have presented a randomised approach to LTL model checking that ensures a uniform distribution on the lassos in the product $B = B_M \times B_{\neg \Phi}$. Thus, there is no accepting lasso with too low probability to be traversed, whatever the topology of the underlying graph.

As presented here, the proposed algorithm still needs an exhaustive traversal of the state graph during the pre-processing stage. This could seriously limit its applicability. A first improvement of the method would be to use on the fly techniques to avoid the storage in memory of the whole model during the DFS. A second improvement would be to avoid the complete storage of the vector, parts of it being computed during the generation stage, when needed. However, it will somewhat increase the time complexity of drawing. Another possibility would be approximate lasso counting, thus approximate uniformity, under the condition that the approximation error can be taken into account in the estimation of the satisfaction probability, which is an open issue.

First experiments, on examples known to be pathological, show that the method leads to a much better detection power of counter-examples, and that the drawing time is acceptable. In Section 5, we report a case with long counter-examples difficult to reach by isotropic random walks. A counter-example is discovered after a reasonable number of drawings, where isotropic exploration would require prohibitive numbers of them. The method needs to be validated on some more realistic example. We plan to embed it in an existing model-checker in order to check LTL properties on available case studies.

The method is applicable to models where the underlying graph is a reducible flow graph: we give a method for counting lassos and drawing them at random in this class of graphs, after recalling that it is a hard problem in general. Reducible data flow graphs correspond to well-structured programs and control-command systems (i.e., the steam boiler [2]). A perspective of improvement would be to alleviate the requirement of reducibility. It seems feasible: for instance, some data flow analysis algorithms have been generalised to communicating automata

in [15]. Similarly, we plan to study the properties and numbers of lassos in product of reducible automata, in order to consider multi-threaded programs.

References

1. Abed, N., Tripakis, S., Vincent, J.-M.: Resource-aware verification using randomized exploration of large state spaces. In: Havelund, K., Majumdar, R. (eds.) SPIN 2008. LNCS, vol. 5156, pp. 214–231. Springer, Heidelberg (2008)
2. Abrial, J.-R., Börger, E., Langmaack, H. (eds.): Formal Methods for Industrial Applications, Specifying and Programming the Steam Boiler Control (the book grow out of a Dagstuhl Seminar). LNCS, vol. 1165. Springer, Heidelberg (1996)
3. Aldous, D.: An introduction to covering problems for random walks on graphs. J. Theoret. Probab. 4, 197–211 (1991)
4. Bérard, B., Bidoit, M., Finkel, A., Laroussinie, F., Petit, A., Petrucci, L., Schnoebelen, P.: Systems and Software Verification. In: Model-Checking Techniques and Tools, Springer, Heidelberg (2001)
5. Courcoubetis, C., Vardi, M., Wolper, P., Yannakakis, M.: Memory-efficient algorithms for the verification of temporal properties. Formal methods in system design 1(2), 275–288 (1992)
6. Denise, A., Gaudel, M.-C., Gouraud, S.-D.: A generic method for statistical testing. In: 15th International Symposium on Software Reliability Engineering (ISSRE 2004), pp. 25–34. IEEE Computer Society, Los Alamitos (2004)
7. Denise, A., Gaudel, M.-C., Gouraud, S.-D., Lassaigne, R., Peyronnet, S.: Uniform random sampling of traces in very large models. In: 1st International ACM Workshop on Random Testing, pp. 10–19 (July 2006)
8. Dwyer, M.B., Elbaum, S.G., Person, S., Purandare, R.: Parallel randomized state-space search. In: 29th International Conference on Software Engineering (ICSE 2007), pp. 3–12 (2007)
9. Gastin, P., Oddoux, D.: Fast LTL to Büchi automata translation. In: Berry, G., Comon, H., Finkel, A. (eds.) CAV 2001. LNCS, vol. 2102, pp. 53–65. Springer, Heidelberg (2001)
10. Gaudel, M.-C., Denise, A., Gouraud, S.-D., Lassaigne, R., Oudinet, J., Peyronnet, S.: Coverage-biased random exploration of large models. In: 4th ETAPS Workshop on Model Based Testing. Electronic Notes in Theoretical Computer Science, vol. 220(1,10), pp. 3–14 (2008) (invited lecture)
11. Grosu, R., Smolka, S.A.: Monte Carlo model checking. In: Halbwachs, N., Zuck, L.D. (eds.) TACAS 2005. LNCS, vol. 3440, pp. 271–286. Springer, Heidelberg (2005)
12. Hecht, M.S., Ullman, J.D.: Characterizations of reducible flow graphs. J. ACM 21(3), 367–375 (1974)
13. Hickey, T., Cohen, J.: Uniform random generation of strings in a context-free language. SIAM J. Comput. 12(4), 645–655 (1983)
14. Johnson, D.B.: Finding all the elementary circuits of a directed graph. SIAM J. Comput. 4(1), 77–84 (1975)
15. Labbé, S., Gallois, J.-P.: Slicing communicating automata specifications: polynomial algorithms for model reduction. Formal Asp. Comput. 20(6), 563–595 (2008)
16. Devroye, L.: Non-Uniform Random Variate Generation. Springer, Heidelberg (1986)

17. Oudinet, J.: Uniform random walks in very large models. In: RT 2007: Proceedings of the 2nd International Workshop on Random Testing, pp. 26–29. ACM Press, New York (2007)
18. Oudinet, J., Denise, A., Gaudel, M.-C.: A new dichotomic algorithm for the uniform random generation of words in regular languages. In: Conference on random and exhaustive generation of combinatorial objects (GASCom), Montreal, Canada, p. 10 (September 2010)
19. Pelánek, R., Hanžl, T., Černá, I., Brim, L.: Enhancing random walk state space exploration. In: Proc. of Formal Methods for Industrial Critical Systems (FMICS 2005), Lisbon, Portugal, pp. 98–105. ACM Press, New York (2005)
20. Rungta, N., Mercer, E.G.: Generating counter-examples through randomized guided search. In: Bošnački, D., Edelkamp, S. (eds.) SPIN 2007. LNCS, vol. 4595, pp. 39–57. Springer, Heidelberg (2007)
21. Sivaraj, H., Gopalakrishnan, G.: Random walk based heuristic algorithms for distributed memory model checking. In: Proc. of Parallel and Distributed Model Checking (PDMC 2003). Electr. Notes Theor. Comput. Sci., vol. 89(1) (2003)
22. Tarjan, R.E.: Enumeration of the elementary circuits of a directed graph. SIAM J. Comput. 2(3), 211–216 (1973)
23. Tiernan, J.C.: An efficient search algorithm to find the elementary circuits of a graph. Commun. ACM 13(12), 722–726 (1970)
24. Vardi, M.Y.: An automata-theoretic approach to linear temporal logic. In: Moller, F., Birtwistle, G. (eds.) Logics for Concurrency. LNCS, vol. 1043, pp. 238–266. Springer, Heidelberg (1996)
25. Weinblatt, H.: A new search algorithm for finding the simple cycles of a finite directed graph. J. ACM 19(1), 43–56 (1972)
26. West, C.H.: Protocol validation in complex systems. In: SIGCOMM 1989: Symposium proceedings on Communications architectures & protocols, pp. 303–312. ACM, New York (1989)