ProbReach: Verified Probabilistic Delta-Reachability for Stochastic Hybrid Systems

Fedor Shmarov
School of Computing Science
Newcastle University
Newcastle upon Tyne, UK
f.shmarov@ncl.ac.uk

Paolo Zuliani
School of Computing Science
Newcastle University
Newcastle upon Tyne, UK
paolo.zuliani@ncl.ac.uk

ABSTRACT

We present ProbReach, a tool for verifying probabilistic reachability for stochastic hybrid systems, i.e., computing the probability that the system reaches an unsafe region of the state space. In particular, ProbReach will compute an arbitrarily small interval which is guaranteed to contain the required probability. Standard (non-probabilistic) reachability is undecidable even for linear hybrid systems. In ProbReach we adopt the weaker notion of delta-reachability, in which the unsafe region is overapproximated by a user-defined parameter (delta). This choice leads to false alarms, but also makes the reachability problem decidable for virtually any hybrid system. In ProbReach we have implemented a probabilistic version of delta-reachability that is suited for hybrid systems whose stochastic behaviour is given in terms of random initial conditions. In this paper we introduce the capabilities of ProbReach, give an overview of the parallel implementation, and present results for several benchmarks involving highly non-linear hybrid systems.

Categories and Subject Descriptors

C.3 [**Special-purpose and application-base systems**]: Real-time and embedded systems; D.2.4 [**Software Engineering**]: Software/Program Verification—*Model checking*

Keywords

Probabilistic model checking, hybrid systems, stochastic systems, bounded model checking

1. INTRODUCTION

In modern society, we interact with cyber-physical systems (*e.g.*, cars and airplanes) on a daily basis. Some of these systems are safety-critical, with human lives crucially depending on their reliability and correctness. Thus, verification of cyber-physical systems is extremely important.

Verifying cyber-physical systems is a very difficult task and can be performed in various ways. We employ hybrid systems as an expressive framework for modelling and verification of cyber-physical systems. One of the most important properties investigated by researchers in hybrid systems is reachability. The main reason being that many verification problems can be presented as reachability problems. In other words, we wish to verify whether a hybrid system reaches an *unsafe* region — a subset of the state space of the system representing an unwanted behaviour. The reachability problem is undecidable in general (even for linear hybrid systems [1]). We avoid undecidability issues by solving instead the weaker δ-reachability problem [?], which asks whether a hybrid system reaches an *overapproximation* of the unsafe region.

In this paper we focus on hybrid systems featuring stochastic behaviour. Such systems frequently arise when modelling real-world cyber-physical systems. For example, random behaviour can happen due to soft errors in some components of the system. Without a doubt this can cause the whole system behaving in a faulty way. By investigating a problematic component, its characteristics (*e.g.*, error distribution) can be obtained. In this case it might be necessary not only to predict an undesired behaviour but also show that the probability of occurrence of a bad event is below (or above) some required threshold. This problem is called probabilistic reachability, and it can be expressed for stochastic hybrid systems. In particular, we consider hybrid systems with random continuous/discrete initial parameters. Such parameters are assigned in the initial mode and remain unchanged throughout the system's evolution. Having a probability measure on random parameters we can assess quantitative properties of hybrid systems such as the probability of reaching an *unsafe* set of states.

We implemented the tool *ProbReach* which performs verified computation of the probability that a hybrid system reaches an unsafe region within a finite number of discrete steps. In particular, our tool implements a general procedure for computing an interval of arbitrarily small length which is *guaranteed* to contain the *exact* value of the probability. *ProbReach* works for general hybrid systems whose continuous dynamics is given, *e.g.*, as a solution of ordinary differential equations (ODEs). Our tool uses δ-complete decision procedures [6] and implements a verified integration procedure [10] used for integrating probability measures of random variables.

Related work.

To the best of our knowledge, SiSAT [3] is the only tool that can perform verified reachability analysis in hybrid systems with random parameters. However, it supports only discrete random variables, while *ProbReach* accepts continuous and discrete random initial parameters. A recent work [2] proposes a statistical model checking technique for verifying hybrid systems with continuous nondeterminism, thereby significantly expanding the class of systems analysable. However, the approach is based on statistical planning algorithms from AI, and therefore it cannot offer the absolute guarantees provided by *ProbReach*. A similar approach has been taken by the SReach tool [13], which combines statistical techniques with δ-complete procedures. The advantages of SReach are its ability to handle large numbers of initial random variables and probabilistic transitions. Again, SReach can only offer statistical guarantees, while *ProbReach* focuses on absolutely correct results. Also, in Section 4 we essentially show that *ProbReach* can be as fast as statistical (Monte Carlo) methods.

In this paper we explain the theoretical background of *ProbReach*, its implementation details and consider several case studies such as an insulin glucose regulatory system [11], a controlled bouncing ball [9], and a thermostat model.

2. BACKGROUND

We give here a brief overview of the theory underlying *ProbReach*. For simplicity we focus on one continuous random parameter only — more details can be found in [12]. *ProbReach* addresses the following problem:

> *what is the probability that a hybrid system with random initial parameters reaches the unsafe region U in k steps?*

As this problem is in general undecidable, we adopt the weaker notion of δ-reachability. In our setting it means that *ProbReach* will actually compute an interval of a user-specified length $\epsilon > 0$ that is guaranteed to contain the reachability probability. The main idea of the approach implemented in the tool is to compute the probability by integrating an indicator function over the probability measure of the random variable as:

$$\int_\Omega I_U(r) dP(r)$$

where $P(r)$ is a probability measure of the random variable, Ω is the domain of the random variable, and I_U is the indicator function defined as:

$$I_U(r) = \begin{cases} 1, \text{system with parameter } r \text{ reaches } U \text{ in } k\text{-steps} \\ 0, \text{otherwise.} \end{cases}$$

The procedure for solving probabilistic reachability combines a validated integration procedure and a decision procedure. The first one integrates a probability measure (probability density function) of a random variable and obtains a partition of the random variable domain which guarantees that the probability interval is not larger than the desired length ϵ. The second procedure evaluates the indicator function on each of the intervals from the obtained partition and performs a partial analysis of the interval if necessary.

Validated Integration Procedure.

The problem here is to compute the integral function defined by

$$\mathcal{I}([a,b]) = \int_a^b f(x) dx$$

up to an error ϵ. In the implementation of our validated integration procedure we employ the (1/3) Simpson rule which, by applying interval arithmetics [4], can be formulated as:

$$\mathcal{I}([a,b]) \in [\mathcal{I}]([a,b]) = \frac{b-a}{6}([f](a) + 4[f](\frac{a+b}{2})) +$$
$$[f](b)) - \frac{(b-a)^5}{2880}[f]^{(4)}([a,b])$$

where $[\mathcal{I}]$ and $[f]$ are the interval extensions of functions $\mathcal{I}$ and f. Then by the definition of integral:

$$\mathcal{I}([a,b]) \in \Sigma_{i=1}^n [\mathcal{I}]([r]_i)$$

where n is a number of disjoint intervals $[r]_i$ that partition $[a,b]$. Interval extensions can be readily computed using interval arithmetics libraries such as FILIB++ [8].

Decision Procedure.

Our decision procedure encodes bounded δ-reachability in hybrid systems as a first-order logic formula. This formula is then passed to a δ-complete decision procedure [5] which uses the notion of δ-weakening of a logical formula. Basically, the main idea is to perform evaluation of a weaker (decidable) formula and make a conclusion about the initial formula on this basis. Given an arbitrary first order formula the δ-complete procedure returns **unsat** if the formula is false and δ-**sat** if its weakening is true. Hence, unlike **unsat**, δ-**sat** is a *weak* answer as it does not imply the satisfiability of the formula. We use this fact to define a decision procedure for verifying the indicator function above. The decision procedure comprises two formulas ϕ and ϕ^C which are defined as following:

- $\phi([r]_i)$ is **true** if the interval $[r]_i$ contains a value r such that $I_U(r) = 1$ and **false** if $I_U(r) = 0$ for all the points of the interval

- $\phi^C([r]_i)$ is **true** if there is a value in $[r]_i$ such that $I_U(r) = 0$ and **false** if $I_U(r) = 1$ everywhere on the interval.

Verifying now both formulas using dReach[1], we obtain four outcomes which can be interpreted as follows:

- $\phi([r]_i)$ is **unsat**. Hence, $I_U(r) = 0$ in all points on $[r]_i$ *for sure.*

- $\phi([r]_i)$ is δ-**sat**. Then there is a value in the interval $[r]_i$ such that the system reaches the unsafe region U or its weaker definition (δ-weakening).

- $\phi^C([r]_i)$ is **unsat**. Thus, $I_U(r) = 1$ point-wise on $[r]_i$ *for sure.*

- $\phi^C([r]_i)$ is δ-**sat**. Then there is a value in the interval $[r]_i$ such that the system stays outside the unsafe region or its weakening within the k-th step.

---

[1] http://dreal.cs.cmu.edu/dreach.html

As it was stated above, only **unsat** returned for either of the formulas guarantees the correctness of the interval validation. Therefore, if both formulas evaluates as δ-**sat** then either a *false alarm* is obtained (when a formula which should be unsatisfiable is verified as δ-sat because of a relatively large value of δ used for verification) or the analysed interval is *mixed* (*i.e.*, it contains a value r for which $I_U(r) = 0$ and also a value s for which $I_U(s) = 1$) which means that the interval should be partitioned and verified again. The pseudo-code of the algorithm implemented in *ProbReach* is presented in Algorithm 1.

3. SYSTEM OVERVIEW

This section aims giving an overview of the main components of *ProbReach*, their interaction, and implementation details. The architecture of the tool is shown in Figure 1.

Algorithm 1: *ProbReach* (one cont. random parameter)

Input : probability density f, $t \in (0,1) \cap \mathbb{Q}$, $\epsilon \in (0,1] \cap \mathbb{Q}$, formula ϕ, ϕ^C
Output: interval $[I]$: $\int_B f \in [I]$ and $width([I]) \leq \epsilon$

1 $\epsilon_{inf} = t\epsilon$
2 $\epsilon_{prob} = (1-t)\epsilon$
3 $[a,b] = bounds(f, \epsilon_{inf})$ {obtain bounds}
4 $B.push(integral(f, [a,b], \epsilon_{prob}))$ {get partition}
5 $[P_{lower}] = [0.0, 0.0]$ {interval for lower approx}
6 $[P_{upper}] = [1.0, 1.0]$ {interval for upper approx}
7 **while** $\overline{[P_{upper}]} - \underline{[P_{lower}]} > \epsilon_{prob}$ **do**
8 $D = \emptyset$ {extra interval divisions}
9   **while** $size(B) > 0$ **do**
10 $\{[x], [S]([x])\} = B.pop()$ {get an interval}
11   **if** $\phi([x]) == \delta\text{-}sat$ **then**   {call dReach}
12   **if** $\phi^C([x]) == \delta\text{-}sat$ **then**   {call dReach}
13 $D.push(\{[\underline{x}, mid([x])], [S([\underline{x}, mid([x])])]\})$
              $D.push(\{[mid([x]), \overline{x}], [S([mid([x]), \overline{x}])]\})$
14   **else** $[P_{lower}] = [P_{lower}] + [S]([x])$ {update int}
15   **else** $[P_{upper}] = [P_{upper}] - [S]([x])$ {update int}
16 $B = D$
17 $[P_{upper}] = [P_{upper}] + 1 - \int_a^b f(x)\, dx$ {add leftovers}
18 **return** $[\underline{[P_{lower}]}, \overline{[P_{upper}]}]$

Figure 1: Architecture of ProbReach

In the first step *ProbReach* validates the input and extracts all the necessary data. The application requires a single input file (containing ϕ and ϕ^C) in PDRH format. This file is used further as templates by the **Formula Generator**. An example of the PDRH model of a two-mode thermostat is given below. Note in particular the declaration of a random parameter x distributed as a normal with mean 30 and standard deviation 1.

```
1   #define K 1.0
2   [0, 5] time;
3   [0, 1000] tau;
4   //random parameter declaration
5   N(30, 1) x;
6   //cooling mode
7   { mode 1;
8   invt:
9   (x >= 18);
10  flow:
11  d/dt[x] = - x * K;
12  d/dt[tau] = 10.0;
13  jump:
14  (x <= 18) ==> @2 (and (x' = x) (tau' = tau));
15  }
16  //heating mode
17  { mode 2;
18  invt:
19  (x <= 22);
20  flow:
21  d/dt[x] = - K * (x - 30);
22  d/dt[tau] = 10.0;
23  jump:
24  (x >= 22) ==> @1 (and (x' = x) (tau' = tau));
25  }
26  //initial state
27  init:
28  @1(and (tau = 0));
29  //unsafe region
30  goal:
31  @2(and (x >= 19.9) (x <= 20.1) (tau = 6));
32  //unsafe region complement
33  goal_c:
34  @2(or (x < 19.9) (x > 20.1) (tau = 6));
```
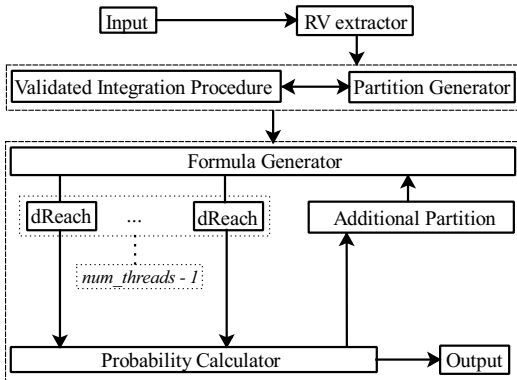
The details of how to use *ProbReach* are given in **Application Usage** section.

The aim of the **RV extractor** is to read all the random variables from the model file containing ϕ, ignoring any other parameter declarations. The tool recognises most of the frequently used distributions (*e.g.*, uniform, normal, exponential), and once the random variables are successfully extracted, their probability density function is automatically generated. Hence, *ProbReach* is not restricted to some set of predefined random variables and can be extended to allow user-defined distributions (by simply providing a probability density function).

Verified integration and Partition generation.

Many useful random variables are defined over unbounded intervals (*e.g.*, normal distribution). However, it was shown in the previous section how to perform verified integration and reachability analysis over bounded intervals only. We cope with unbounded intervals by making a trade-off. Given

a desired length ϵ of the probability interval we choose a value $t \in (0, 1)$ (can be also defined by the user) and obtain an interval $[a, b]$ such that:

$$\int_a^b f(r)\, dr > (1 - t)\epsilon$$

Finding a and b can be actually encoded as a logical formula which can be solved by dReal [6].

The intuition behind this is that we assume that the indicator function equals to 1 outside the interval $[a, b]$. In case if it is not true (the indicator function is 0 in some points outside the considered bounded domain) the integral of the indicator function over the unbounded intervals will be still bounded by $t\epsilon$ (as the integral of a probability density function on interval $(-\infty, \infty)$ is 1).

Then, the **Validated Integration Procedure** computes a definite integral of the probability density function on the obtained finite interval. This is achieved through an iterative partitioning (by **Partition Generator**) of the integration domain until on each interval $[r]_i$ the value of the integral is enclosed by an interval of the length $(1 - t)\epsilon \frac{width([r]_i)}{width([a,b])}$. For such a partition it is guaranteed that the value of the integral over the bounded domain belongs to an interval of length $(1 - t)\epsilon$.

Partition verification.

Once the *correct* partition is obtained, each interval $[r]_i$ is used to generate two model files (encoding $\phi([r]_i)$ and $\phi^C([r]_i)$) in DRH format which are then verified by dReach. This routine was parallelised using the OpenMP shared memory library (see the code below).

```
1   //setting a number of threads
2   int num_threads = omp_get_max_threads();
3   if (num_threads > 1)
4   {
5       omp_set_num_threads(num_threads - 1);
6   }
7   //Algorithm 1 line 6 loop
8   {
9       #pragma_omp_for
10      {
11          //Algorithm 1 line 8
12      }
13      //Algorithm 1 line 15
14      while (B.size() < num_threads - 1)
15      {
16          //partition B to reduce CPU idle
17      }
18  }
```

Initially, the application gets the maximum number of available cores (`num_threads`) and uses `num_threads - 1` (if more then one is available) of them to perform the computation leaving one core to let the computer executing background tasks. Then the partition is distributed between `num_threads - 1` threads and each of them evaluates its interval with dReach.

Now, if for the analysed interval either of the formulas is **unsat** then **Probability Calculator** modifies the probability bounds:

- if $\phi([r]_i)$ is **unsat** then $[r]_i$ is used for calculating P_{upper}

(probability upper bound). The integral of the probability density over the interval $[r]_i$ is subtracted from P_{upper}; initially we of course have $P_{upper} = 1$.

- if $\phi^C([r]_i)$ is **unsat** then $[r]_i$ is used for calculating P_{lower} (probability lower bound). The integral of the probability density over the interval $[r]_i$ is added to P_{under}, starting initially with $P_{under} = 0$.

However, both formulas may be evaluated as δ-**sat** for a given interval from the partition. This suggests that either a *false alarm* is obtained or the interval is *mixed* (it contains values satisfying both formulas). Then, such an interval is subject to **Additional Partition**, which should further undergo the described cycle once again. In the parallel implementation, all *mixed* intervals are partitioned until their number reaches `num_threads - 1`, to reduce CPU idle time. Extra partitioning can be performed arbitrarily many times as it does not alter the correctness of the result. The described routine stops when the length of the interval $[P_{lower}, P_{upper}]$ is shorter than $(1 - t)\epsilon$. Hence, taking into account the assumption about the value of the indicator function outside the bounded domain the probability is guaranteed to be contained inside the interval of the length $t\epsilon + (1 - t)\epsilon = \epsilon$.

Finally, we note that at *any* point in time during the computation, the *exact* value of the probability belongs to the interval $[P_{lower}, P_{upper}]$, which is written in output when the interval bounds change. This might be advantageous for time-critical verification scenarios, as the user can specify a computation timeout. Thus, despite the fact that the desired precision might not be achievable within the specified timeframe, the obtained result is still complete in the sense that the desired probability is *guaranteed* to be inside the computed interval.

Implementation details.

ProbReach has been implemented in C++, using the CAPD library[2] for interval operations. Input analysis is performed using the C++11 regular expression engine. Parallelisation of the code was achieved using OpenMP, and both versions of the tool (parallel and sequential) were built and tested. The parallel implementation running on 24 cores demonstrated a 8-10 times speed up in comparison to the sequential one.

Application usage.

Once the tool has been compiled, the executable is put into `<ProbReach-directory>/bin`. Then the tool can be called from the command line as `./ProbReach <options> <model-file.pdrh> --dreach <dReach-options> --dreal <dReal-options>`. The *ProbReach* options are specified below:

---

[2]`http://capd.ii.uj.edu.pl`

```
options:
-e <double> - length of probability interval
        or max length of box edge (default 0.001)
-l <string> - path to dReach binary (default dReach)
-t <int> - number of CPU cores (default 1)
-h/--help - help message
--version - version of the tool
--verbose - output computation details
--dreach - delimits dReach options
        (e.g., reachability depth)
--dreal - delimits dReal options
        (e.g., precision, ode step)
```

Tool availability.

The source code of *ProbReach* and installation instructions are available on `https://github.com/dreal/probreach`. We also implemented a web application to display *ProbReach*'s results. *ProbReach* outputs intermediate probability intervals to a `JSON` file which can be visualised by `https://homepages.ncl.ac.uk/f.shmarov/probreach/`.

4. EXPERIMENTS

The description of all the models and verification scenarios are given in the Appendix. All the experiments were carried out on a Intel Xeon E5-2690 2.90GHz multi-core system running Linux Ubuntu 14.04LTS. The parallel version of *ProbReach* ran on 24 cores. The results were also validated using a Monte Carlo method in MATLAB. We calculated confidence intervals using the sample size returned by the Chernoff-Hoeffding [7] bound $N = \frac{\log \frac{1}{1-c}}{2\zeta^2}$, where ζ is the interval half-width and c is the coverage probability. The results are presented in Table 1.

Results analysis.

In most of the experiments *ProbReach* demonstrated a better performance than the Monte Carlo method. However, for the Insulin-Glucose (IG) model the Monte Carlo method was faster for the two scenarios considered. Nevertheless, reducing the length of the confidence interval causes a quadratic growth in the sample size. For example, obtaining a confidence interval of size 10^{-4} with coverage 0.999 requires 1.3815510558×10^9 samples, with an estimated CPU time of 2.3×10^9 seconds. *ProbReach* computes a guaranteed enclosure of size smaller than 10^{-4} in about 3.5×10^6 seconds. Hence, for stronger precisions (*i.e.*, smaller ϵ) *ProbReach* performs better than Monte Carlo method.

Considering the results for the thermostat model (see rows T4(1.7) in Table 1), the Monte Carlo method returned a probability estimate (number of successes divided by number of samples) of 9.438088×10^{-8} with a relatively large confidence interval (10^{-5}) using 33,015 seconds of CPU time. *ProbReach* can compute an interval of size about 10^{-9} in just 268 seconds. Computing a confidence interval of length 10^{-9} with coverage 0.99999 requires $2.3025850929 \times 10^{19}$ samples, which suggests that *ProbReach* can be very efficient for rare event verification.

5. CONCLUSIONS AND FUTURE WORK

We have presented the *ProbReach* tool which computes an arbitrarily small interval containing the probability that a hybrid system reaches an unsafe region of its state space. *ProbReach* is not limited to a set of predefined random vari-

ables, as it works with probability density functions. Thus, it can be extended to support user-defined distributions. We have successfully benchmarked *ProbReach* and in many cases it demonstrated a better performance in comparison to Monte Carlo simulations while providing stronger guarantees of result correctness. Finally, it was shown that *ProbReach* is very efficient for rare event verification.

In the future, we plan to implementing a more efficient parallelisation scheme. This will be performed modifying the partition verification approach. Instead of adding *mixed* intervals to a separate queue and verifying them after the main partition, newly partitioned intervals will be pushed to the end of the main queue. Then, a parallelisation manager monitoring the available cores will be dynamically distributing the load equally between the threads, thus reducing CPU idle. According to our estimations, this modification will significantly increase the performance of the tool.

Another extension is to allow probabilistic jumps in the model. We plan to allow jumps whose probabilities may depend on the (continuous) variables and parameters. Finally, we plan to support both nondeterministic and random continuous parameters. For such systems, probabilistic reachability becomes in general an optimisation problem, as the nondeterministic parameters may generate *ranges* of probabilities. These two additions will enlarge very much the class of models analyzable by *ProbReach*.

6. ACKNOWLEDGMENTS

7. REFERENCES

[1] R. Alur, C. Courcoubetis, T. A. Henzinger, and P.-H. Ho. Hybrid automata: An algorithmic approach to the specification and verification of hybrid systems. In *Hybrid Systems*, volume 736 of *LNCS*, pages 209–229, 1992.

[2] C. Ellen, S. Gerwinn, and M. Fränzle. Statistical model checking for stochastic hybrid systems involving nondeterminism over continuous domains. *STTT*, 2014. To appear.

[3] M. Fränzle, T. Teige, and A. Eggers. Engineering constraint solvers for automatic analysis of probabilistic hybrid automata. *J. Log. Algebr. Program.*, 79(7):436–466, 2010.

[4] S. Galdino. Interval integration revisited. *Open Journal of Applied Sciences*, 2(4B):108–111, 2012.

[5] S. Gao, J. Avigad, and E. M. Clarke. Delta-complete decision procedures for satisfiability over the reals. In *IJCAR*, pages 286–300, 2012.

[6] S. Gao, S. Kong, and E. M. Clarke. dReal: An SMT solver for nonlinear theories over the reals. In *CADE*, pages 208–214, 2013.

[7] W. Hoeffding. Probability inequalities for sums of bounded random variables. *J. Amer. Statist. Assoc.*, 58(301):13–30, 1963.

[8] M. Lerch, G. Tischler, J. W. V. Gudenberg, W. e. Hofschuster, and W. Krämer. FILIB++, a fast interval library supporting containment computations. *ACM Trans. Math. Softw.*, 32(2):299–324, 2006.

[9] P. J. Mosterman, J. Zander, G. Hamon, and B. Denckla. Towards computational hybrid system

Tool	Model	k	ϵ	length		Probability interval	CPU_{seq}	CPU_{par}
Prob Reach	BB	0	10^{-9}	5.0e-10		[8.21757e-05, 8.21762e-05]	64	7
		1	10^{-9}	1.0e-09		[0.1379483631, 0.1379483641]	192	29
		2	10^{-9}	9.9e-10		[0.50868960502, 0.50868960601]	927	164
		3	10^{-9}	8.0e-10		[0.7387674005, 0.7387674013]	3806	563
			ζ	c	P	Confidence interval	CPU_{seq}	Sample size
Monte Carlo	BB	0	$5 \cdot 10^{-6}$	0.99999	8.220032e-05	[7.720032e-05, 8.720032e-05]	16,455	230,258,509,300
		1	$5 \cdot 10^{-6}$	0.99999	0.1379449	[0.1379399, 0.1379499]	19,646	230,258,509,300
		2	$5 \cdot 10^{-6}$	0.99999	0.5086939	[0.5086889, 0.5086989]	21,197	230,258,509,300
		3	$5 \cdot 10^{-6}$	0.99999	0.7387684	[0.7387634, 0.7387734]	20,975	230,258,509,300
			ϵ	length		Probability interval	CPU_{seq}	CPU_{par}
Prob Reach	T2(0.6)	1	10^{-9}	9.46e-10		[0.006678444555, 0.0066784456]	71	7
	T2(1.8)	5	10^{-9}	1.0e-9		[0.0026170599, 0.0026170609]	213	23
	T2(2.4)	7	10^{-9}	1.0e-9		[0.0015794358, 0.0015794368]	364	49
			ζ	c	P	Confidence interval	CPU_{seq}	Sample size
Monte Carlo	T2(0.6)	1	$5 \cdot 10^{-6}$	0.99999	0.006679496	[0.006674496, 0.006684496]	31,822	230,258,509,300
	T2(1.8)	5	$5 \cdot 10^{-6}$	0.99999	0.002616634	[0.002611634, 0.002621634]	33,287	230,258,509,300
	T2(2.4)	7	$5 \cdot 10^{-6}$	0.99999	0.001579243	[0.001574243, 0.001584243]	33,772	230,258,509,300
			ϵ	length		Probability interval	CPU_{seq}	CPU_{par}
Prob Reach	T4(0.6)	2	10^{-9}	8.55e-11		[0.0, 8.55e-11]	52	4
	T4(1.7)	6	10^{-9}	7.962e-10		[9.43986e-08, 9.51948e-08]	268	28
	T4(1.8)	6	10^{-9}	9.0e-10		[0.0039559433, 0.0039559442]	578	75
			ζ	c	P	Confidence interval	CPU_{seq}	Sample size
Monte Carlo	T4(0.6)	2	$5 \cdot 10^{-6}$	0.99999	0	[0, 5e-06]	32,883	230,258,509,300
	T4(1.7)	6	$5 \cdot 10^{-6}$	0.99999	9.438088e-08	[0, 5.094381e-06]	33,015	230,258,509,300
	T4(1.8)	6	$5 \cdot 10^{-6}$	0.99999	0.003955074	[0.003950074, 0.003960074]	33,354	230,258,509,300
			ϵ	length		Probability interval	CPU_{seq}	CPU_{par}
Prob Reach	CBB	2	10^{-2}	8.0e-3		[0.199, 0.207]	70	15
		2	10^{-9}	3.0e-10		[0.2049030217, 0.204903022]	8,332	2,581
			ζ	c	P	Confidence interval	CPU_{seq}	Sample size
Monte Carlo	CBB	2	$5 \cdot 10^{-3}$	0.99	0.2045948	[0.1995948, 0.2095948]	50,528	92,104
			ϵ	length		Probability interval	CPU_{seq}	CPU_{par}
Prob Reach	IG	1	10^{-2}	5.328e-3		[0.994589, 0.999917]	2,805,634	165,404
		1	10^{-3}	8.1e-4		[0.999107, 0.999917]	3,326,581	443,910
		1	10^{-4}	5.5e-5		[0.999657, 0.999712]	3,498,765	490,257
			ζ	c	P	Confidence interval	CPU_{seq}	Sample size
Monte Carlo	IG	1	$5 \cdot 10^{-3}$	0.99	0.997266555	[0.9945331, 1]	58,069	92,104
		1	$2.5 \cdot 10^{-3}$	0.99	0.99853	[0.99706, 1]	219,623	368,416

Table 1: **Computing probabilistic reachability with ProbReach and MATLAB.** k = number of discrete transitions; ϵ = desired size of probability interval; *length* = length of probability interval returned by *ProbReach*; ζ, c = half-interval width and coverage probability for Chernoff bound; Sample size = number of simulations (Chernoff bound); P = probability estimate (successes/Sample size); CPU_{seq}, CPU_{par} = CPU time (sec) of sequential and parallel version; BB = bouncing ball model; CBB = controlled bouncing ball model; T2(0.6), T2(1.8), T2(2.4) = thermostat model with 2 modes at $t = 0.6, 1.8, 2.4$ respectively; T4(0.6), T4(1.7), T4(1.8) = thermostat model with 4 modes at $t = 0.6, 1.7, 1.8$ respectively.

semantics for time-based block diagrams. In *3rd IFAC Conference on Analysis and Design of Hybrid Systems (ADHS'09)*, pages 376–385, 2009.

[10] K. Petras. Principles of verified numerical integration. *Journal of Computational and Applied Mathematics*, 199(2):317 – 328, 2007.

[11] S. Sankaranarayanan and G. Fainekos. Simulating insulin infusion pump risks by in-silico modeling of the insulin-glucose regulatory system. In *CMSB*, volume 7605 of *LNCS*, pages 322–341, 2012.

[12] F. Shmarov and P. Zuliani. Probabilistic bounded reachability for hybrid systems with continuous nondeterministic and probabilistic parameters. *CoRR*, abs/1406.1920, 2014.

[13] Q. Wang, P. Zuliani, S. Kong, S. Gao, and E. M. Clarke. SReach: A bounded model checker for stochastic hybrid systems. *CoRR*, abs/1404.7206, 2014.