StocHy: automated verification and synthesis of stochastic processes

Nathalie Cauchi[1] and Alessandro Abate[1]

Department of Computer Science,
University of Oxford, United Kingdom,

Abstract. StocHy is a software tool for the quantitative analysis of discrete-time *stochastic hybrid systems* (SHS). StocHy accepts a high-level description of stochastic models and constructs an equivalent SHS model. The tool allows to (i) simulate the SHS evolution over a given time horizon; and to automatically construct formal abstractions of the SHS. Abstractions are then employed for (ii) formal verification or (iii) control (policy, strategy) synthesis. StocHy allows for modular modelling, and has separate simulation, verification and synthesis engines, which are implemented as independent libraries. This allows for libraries to be easily used and for extensions to be easily built. The tool is implemented in C++ and employs manipulations based on vector calculus, the use of sparse matrices, the symbolic construction of probabilistic kernels, and multi-threading. Experiments show StocHy's markedly improved performance when compared to existing abstraction-based approaches: in particular, StocHy beats state-of-the-art tools in terms of precision (abstraction error) and computational effort, and finally attains scalability to large-sized models (12 continuous dimensions). StocHy is available at `www.gitlab.com/natchi92/StocHy`.

1 Introduction

Stochastic hybrid systems (SHS) are a rich mathematical modelling framework capable of describing systems with complex dynamics, where uncertainty and hybrid (that is, both continuous and discrete) components are relevant. Whilst earlier instances of SHS have a long history, SHS proper have been thoroughly investigated only from the mid 2000s, and have been most recently applied to the study of complex systems, both engineered and natural. Amongst engineering case studies, SHS have been used for modelling and analysis of micro grids [?], smart buildings [23], avionics [7], automation of medical devices [3]. A benchmark for SHS is also described in [10]. However, a wider adoption of SHS in real-world applications is stymied by a few factors: (i) the complexity associated with modelling SHS; (ii) the generality of their mathematical framework, which requires an arsenal of advanced and diverse techniques to analyse them; and (iii) the undecidability of verification/synthesis problems over SHS and the curse of dimensionality associated with their approximations.

This paper introduces a new software tool - StocHy - which is aimed at simplifying both the modelling of SHS and their analysis, and which targets

the wider adoption of SHS, also by non-expert users. With focus on the three limiting factors above, StocHy allows to describe SHS by parsing or extending well-known and -used state-space models and generates a standard SHS model automatically and formats it to be analysed. StocHy can (i) perform verification tasks, e.g., compute the probability of staying within a certain region of the state space from a given set of initial conditions; (ii) automatically synthesise policies (strategies) maximising this probability, and (iii) simulate the SHS evolution over time. StocHy is implemented in C++ and modular making it both extendible and portable.

Related work. There exist only a few tools that can handle (classes of) SHS. Of much inspiration for this contribution, FAUST^2 [28] generates abstractions for uncountable-state discrete-time stochastic processes, natively supporting SHS models with a single discrete mode and finite actions, and performs verification of reachability-like properties and corresponding synthesis of policies. FAUST^2 is naïvely implemented in MATLAB and lacks in scalability to large models. The MODEST TOOLSET [18] allows to model and to analyse classes of continuous-time SHS, particularly probabilistic hybrid automata (PHA) that combine probabilistic discrete transitions with deterministic evolution of the continuous variables. The tool for stochastic and dynamically coloured petri nets (SDCPN) [13] supports compositional modelling of PHA and focuses on simulation via Monte Carlo techniques. The existing tools highlight the need for a new software that allows for (i) straightforward and general SHS modelling construction and (ii) scalable automated analysis.

Contributions. The StocHy tool newly enables

- *formal verification* of SHS via either of two abstraction techniques:
  - for discrete-time, continuous-space models with additive disturbances, and possibly with multiple discrete modes, we employ formal abstractions as general Markov chains or Markov decision processes [28]; StocHy improves techniques in the FAUST^2 tool by simplifying the input model description, by employing sparse matrices to manipulate the transition probabilities and by reducing the computational time needed to generate the abstractions.
  - for models with a finite number of actions, we employ interval Markov decision processes and the model checking framework in [22]; StocHy provides a novel abstraction algorithm allowing for efficient computation of the abstract model, by means of an adaptive and sequential refining of the underlying abstraction. We show that we are able to generate significantly smaller abstraction errors and to verify models with up to 12 continuous variables.
- *control* (strategy, policy) *synthesis* via formal abstractions, employing:
  - stochastic dynamic programming; StocHy exploits the use of symbolic kernels.
  - robust synthesis using interval Markov decision processes; StocHy automates the synthesis algorithm with the abstraction procedure and the temporal property of interest, and exploits the use of sparse matrices;

– *simulation* of complex stochastic processes, such as SHS, by means of Monte Carlo techniques; StocHy automatically generates statistics from the simulations in the form of histograms, visualising the evolution of both the continuous random variables and the discrete modes.

This contribution is structured as follows: Sec. 2 crisply presents the theoretical underpinnings (modelling and analysis) for the tool. We provide an overview of the implementation of StocHy in Sec. 3. We highlight features and use of StocHy by a set of experimental evaluations in Sec. 4: we provide four different case studies that highlight the applicability, ease of use, and scalability of StocHy. Details on executing all the case studies are detailed in this paper and within a Wiki page that accompanies the StocHy distribution.

2   Theory: Models, Abstractions, Simulations

2.1   Models - Discrete-time Stochastic Hybrid Systems

StocHy supports the modelling of the following general class of SHS [1,4].

Definition 1. *A* SHS *[4] is a discrete-time model defined as the tuple*

$$\mathcal{H} = (\mathcal{Q}, n, \mathcal{U}, T_x, T_q), \quad where \tag{1}$$

– $\mathcal{Q} = \{q_1, q_2, \ldots, q_m\}$, $m \in \mathbb{N}$, *represents a finite set of modes (locations);*
– $n \in \mathbb{N}$ *is the dimension of the continuous space* $\mathbb{R}^n$ *of each mode; the hybrid state space is then given by* $\mathcal{D} = \cup_{q \in \mathcal{Q}} \{q\} \times \mathbb{R}^n$;
– $\mathcal{U}$ *is a continuous set of actions, e.g.* $\mathbb{R}^v$;
– $T_q : \mathcal{Q} \times \mathcal{D} \times \mathcal{U} \to [0,1]$ *is a discrete stochastic kernel on Q given* $\mathcal{D} \times \mathcal{U}$, *which assigns to each* $s = (q, x) \in \mathcal{D}$ *and* $u \in \mathcal{U}$, *a probability distribution over* $\mathcal{Q} : T_q(\cdot|s, u)$;
– $T_x : \mathcal{B}(\mathbb{R}^n) \times \mathcal{D} \times \mathcal{U} \to [0,1]$ *is a Borel-measurable stochastic kernel on* $\mathbb{R}^n$ *given* $\mathcal{D} \times \mathcal{U}$, *which assigns to each* $s \in \mathcal{D}$ *and* $u \in \mathcal{U}$ *a probability measure on the Borel space* $(\mathbb{R}^n, \mathcal{B}(\mathbb{R}^n)) : T_x(\cdot|s, u)$.

In this model the discrete component takes values in a finite set $\mathcal{Q}$ of modes (a.k.a. locations), each endowed with a continuous domain (the Euclidean space $\mathbb{R}^n$). As such, a point s over the hybrid state space $\mathcal{D}$ is pair (q, x), where $q \in \mathcal{Q}$ and $x \in \mathbb{R}^n$. The semantics of transitions at any point over a discrete time domain, are as follows: given a point $s \in \mathcal{D}$, the discrete state is chosen from T_q, and depending on the selected mode $q \in \mathcal{Q}$ the continuous state is updated according to the probabilistic law T_x. Non-determinism in the form of actions can affect both discrete and continuous transitions.

Remark 1. A rigorous characterisation of SHS can be found in [1], which introduces a general class of models with probabilistic resets and a hybrid actions space. Whilst we can deal with general SHS models, in the case studies of this paper we focus on special instances, as described next.   □

Remark 2 (Special instance). In Case Study 2 (see Sec. 4.2) we look at models where actions are associated to a deterministic selection of locations, namely $T_q : \mathcal{U} \to \mathcal{Q}$ and $\mathcal{U}$ is a finite set of actions.   □

Remark 3 (Special instance). In Case Study 4 (Section 4.4) we consider non-linear dynamical models with bilinear terms, which are characterised for any $q \in \mathcal{Q}$ by $x_{k+1} = A_q x_k + B_q u_k + x_k \sum_{i=1}^{v} N_{q,i} u_{i,k} + G_q w_k$, where $k \in \mathbb{N}$ represents the discrete time index, A_q, B_q, G_q are appropriately sized matrices, $N_{q,i}$ represents the bilinear influence of the $i-$th input component u_i, and $w_k = w \sim \mathcal{N}(\cdot; 0, 1)$ and $\mathcal{N}(\cdot; \eta, \nu)$ denotes a Gaussian density function with mean η and covariance matrix ν^2. This expresses the continuous kernel $T_x : \mathcal{B}(\mathbb{R}^n) \times \mathcal{D} \times \mathcal{U} \to [0, 1]$ as

$$\mathcal{N}(\cdot; A_q x + B_q u + x \sum_{i=1}^{v} N_{q,i} u_i + F_q, G_q). \tag{2}$$

In Case Study 1-2-3 (Sec. 4.1-4.3), we look at the special instance from [22], where the dynamics are autonomous (no actions) and linear: here T_x is

$$\mathcal{N}(\cdot; A_q x + F_q, G_q), \tag{3}$$

where in Case Studies 1, 3 $\mathcal{Q}$ is a single element.   □

Definition 2. *A Markov decision process (*MDP*) [5] is a discrete-time model defined as the tuple*

$$\mathcal{H} = (\mathcal{Q}, \mathcal{U}, T_q), \quad where \tag{4}$$

  – $\mathcal{Q} = \{q_1, q_2, \ldots, q_m\}$, $m \in \mathbb{N}$, *represents a finite set of modes;*
  – $\mathcal{U}$ *is a finite set of actions;*
  – $T_q : \mathcal{Q} \times \mathcal{Q} \times \mathcal{U} \to [0, 1]$ *is a discrete stochastic kernel that assigns, to each $q \in \mathcal{Q}$ and $u \in \mathcal{U}$, a probability distribution over $\mathcal{Q} : T_q(\cdot|q, u)$.*

Whenever the set of actions is trivial or a policy is synthesised and used (cf. discussion in Sec. 2.2) the MDP reduces to a Markov chain (MC), and a kernel $T_q : \mathcal{Q} \times \mathcal{Q} \to [0, 1]$ assigns to each $q \in \mathcal{Q}$ a distribution over $\mathcal{Q}$ as $T_q(\cdot|q)$.

Definition 3. *An interval Markov decision process (*IMDP*) [26] extends the syntax of an* MDP *by allowing for uncertain T_q, and is defined as the tuple*

$$\mathcal{H} = (\mathcal{Q}, \mathcal{U}, \check{P}, \hat{P}), \quad where \tag{5}$$

  – $\mathcal{Q}$ *and $\mathcal{U}$ are as in Definition 2;*
  – $\check{P}$ *and $\hat{P} : \mathcal{Q} \times \mathcal{U} \times \mathcal{Q} \to [0, 1]$ is a function that assigns to each $q \in \mathcal{Q}$ a lower (upper) bound probability distribution over $\mathcal{Q} : \check{P}(\cdot|q, u)$ $(\hat{P}(\cdot|q, u)$ respectively).*

For all $q, q' \in \mathcal{Q}$ and $u \in \mathcal{U}$, it holds that $\check{P}(q'|q, u) \leq \hat{P}(q'|q, u)$ and,

$$\sum_{q' \in \mathcal{Q}} \check{P}(q'|q, u) \leq 1 \leq \sum_{q' \in \mathcal{Q}} \hat{P}(q'|q, u).$$

Note that when $\check{P}(\cdot|q, u) = \hat{P}(\cdot|q, u)$, the IMDP *reduces to the* MDP *with $\check{P}(\cdot|q, u) = \hat{P}(\cdot|q, u) = T_q(\cdot|q, u)$.*

2.2   Formal Verification and Strategy Synthesis via Abstractions

Formal verification and strategy synthesis over SHS are in general not decidable [4, 29], and can be tackled via quantitative finite abstractions. These are precise approximations that come in two main different flavours: abstractions into MDP [4, 28] and into IMDP [22]. Once the finite abstractions are obtained, and with focus on specifications expressed in (non-nested) PCTL or fragments of LTL [5], formal verification or strategy synthesis can be performed via probabilistic model checking tools, such as PRISM [21], STORM [12], ISCASMC [17]. We overview next the two alternative abstractions, as implemented in StocHy.

Abstractions into Markov decision processes Following [27], MDP are generated by either (i) uniformly gridding the state space and computing an abstraction error, which depends on the continuity of the underlying continuous dynamics and on the chosen grid; or (ii) generating the grid adaptively and sequentially, by splitting the cells with the largest local abstraction error until a desired global abstraction error is achieved. The two approaches display an intuitive trade-off, where the first in general requires more memory but less time, whereas the second generates smaller abstractions. Either way, the probability to transit from each cell in the grid into any other cell characterises the MDP matrix T_q. Further details can be found in [28]. StocHy newly provides a C++ implementation and employs sparse matrix representation and manipulation, in order to attain faster generation of the abstraction and use in formal verification or strategy synthesis.

Verification via MDP (when the action set is trivial) is performed to check the abstraction against non-nested, bounded-until specifications in PCTL [5] or *co-safe linear temporal logic* (CSLTL) [20].

Strategy synthesis via MDP is defined as follows. Consider, the class of deterministic and memoryless Markov strategies $\pi = (\mu_0, \mu_1, \dots)$ where $\mu_k : \mathcal{Q} \to \mathcal{U}$. We compute the strategy $\pi^\star$ that maximises the probability of satisfying a formula, with algorithms discussed in [28].

Abstraction into Interval Markov decision processes (IMDP) is based on a procedure in [11] performed using a uniform grid and with a finite set of actions $\mathcal{U}$ (see Remark 2). StocHy newly provides the option to generate a grid using adaptive/sequential refinements (similar to the case in the paragraph above) [27], which is performed as follows: (i) define a required minimal maximum abstraction error ε_{max}; (ii) generate a coarse abstraction using the Algorithm in [11] and compute the local error ε_q that is associated to each abstract state q; (iii) split all cells where $\varepsilon_q > \varepsilon_{max}$ along the main axis of each dimension, and update the probability bounds (and errors); and (iv) repeat this process until $\forall q, \ \varepsilon_q < \varepsilon_{max}$.

Verification via IMDP is run over properties in CSLTL or bounded-LTL (BLTL) form using the IMDP model checking algorithm in [22].

Synthesis via IMDP [11] is carried out by extending the notions of strategies of MDP to depend on memory, that is on prefixes of paths.

2.3   Analysis via Monte Carlo simulations

Monte Carlo techniques generate numerical sampled trajectories representing the evaluation of a stochastic process over a predetermined time horizon. Given a sufficient number of trajectories, one can approximate the statistical properties of the solution process with a required confidence level. This approach has been adopted for simulation of different types of SHS. [19] applies sequential Monte Carlo simulation to SHS to reason about rare-event probabilities. [13] performs Monte Carlo simulations of classes of SHS described as Petri nets. [8] proposes a methodology for efficient Monte Carlo simulations of continuous-time SHS. In this work, we analyse a SHS model using Monte Carlo simulations following the approach in [4]. Additionally, we generate histogram plots at each time step, providing further insight on the evolution of the solution process.

3   Overview of StocHy

Installation StocHy is set up using the provided GET_DEP file found within the distribution package, which will automatically install all the required dependencies. The executable RUN.SH builds and runs StocHy. This basic installation setup has been successfully tested on machines running Ubuntu 18.04.1 LTS GNU and Linux operating systems.

Input interface The user interacts with StocHy via the MAIN file and must specify (i) a high-level description of the model dynamics and (ii) the task to be performed. The description of model dynamics can take the form of a list of the transition probabilities between the discrete modes, and of the state-space models for the continuous variables in each mode; alternatively, a description can be obtained by specifying a path to a MATLAB file containing the model description in state-space form together with the transition probability matrix. Tasks can be of three kinds (each admitting specific parameters): simulation, verification, or synthesis. The general structure of the input interface is illustrated via an example in Listing 1.1: here the user is interested in simulating a SHS with two discrete modes $\mathcal{Q} = \{q_0, q_1\}$ and two continuous variables evolve according to (3). The model is autonomous and has no control actions. The relationship between the discrete modes is defined by a fixed transition probability (line 1). The evolution of the continuous dynamics are defined in lines 2-14. The initial condition for both the discrete modes and the continuous variables are set in lines 16-21 (this is needed for simulation tasks). The equivalent SHS model is then set up by instantiating an object of type `shs_t<arma::mat,int>` (line 23). Next, the task is defined in line 27 (simulation with a time horizon $K = 32$, as specified in line 25 and using the simulator library, as set in line 26). We combine the model and task specification together in line 29. Finally, StocHy carries out the simulation using the function `performTask` (line 31).

Modularity StocHy comprises independent libraries for different tasks, namely (i) FAUST², (ii) IMDP, and (iii) simulator. Each of the libraries is separate and depends only on the model structure that has been entered. This allows for

```
1    arma::mat Tq = { {0.4, 0.6},{0.7,0.3}};        // Transition probabilities
2    // Evolution of the continuous variables for each discrete mode
3    // First model
4    arma::mat Aq0 = {{0.5, 0.4},{0.2,0.6}};
5    arma::mat Fq0 = { {0},{0}};
6    arma::mat Gq0 = {{0.4,0},{0.3, 0.3}};
7    ssmodels_t modelq0(Aq0, Fq0, Gq0);
8    // Second model
9    arma::mat Aq1 = {{0.6, 0.3},{0.1,0.7}};
10   arma::mat Fq1 = { {0},{0}};
11   arma::mat Gq1 = {{0.2,0},{0.1, 0}};
12   ssmodels_t modelq1(Aq1,Fq1, Gq1);
13   std::vector<ssmodels_t> models =
14   {modelq1,modelq2};
15   // Set initial conditions
16   // Initial state q_0
17   arma::mat q_init = arma::zeros<arma::mat>(1,1);
18   // Initial continuous variables
19   arma::mat x1_init = arma::ones<arma:mat>(2,1);
20   exdata_t data(x1_init,q_init);
21   // Build shs
22   shs_t<arma::mat,int> mySHS(Tq,models,data);
23   // Time horizon
24   int K = 32;
25   // Task definition  (1 = simulator, 2 = faust^2, 3 = imdp)
26   int lb = 1;
27   taskSpec_t mySpec(lb,K);
28   // Combine
29   inputSpec_t<arma::mat,int> myInput(mySHS,mySpec);
30   // Perform task
31   performTask(myInput);
```
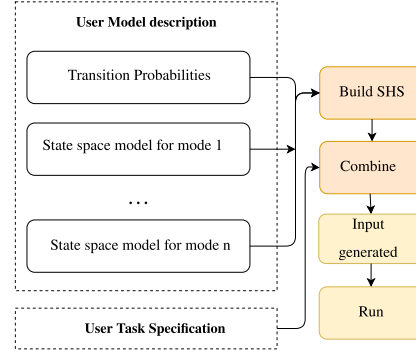
Listing 1.1: Description of MAIN file for simulating a SHS consisting of two discrete modes and two continuous variables evolving according to (2).

seamless extensions of individual sub-modules with new or existing tools and methods. The function `performTask` acts as multiplexer for calling any of the libraries depending on the input model and task specification.

Data structures StocHy makes use of multiple techniques to minimise computational overhead. It employs vector algebra for efficient handling of linear operations, and whenever possible it stores and manipulates matrices as sparse structures. It uses the linear algebra library Armadillo [24, 25], which applies multi-threading and a sophisticated expression evaluator that has been shown to speed up matrix manipulations in C++ when compared to other libraries. FAUST[2] based abstractions define the underlying kernel functions symbolically using the library GiNaC [6], for easy evaluation of the stochastic kernels.

Output interface We provide outputs as text files for all three libraries, which are stored within the RESULTS folder. We also provide additional PYTHON scripts

for generating plots as needed. For abstractions based on FAUST^2, the user has the additional option to export the generated MDP or MC to PRISM format, to interface with the popular model checker [21] (StocHy prompts the user this option following the completion of the verification or synthesis task). As a future extension, we plan to export the generated abstraction models to the model checker STORM [12] and to the modelling format JANI [9].

4   StocHy: Experimental Evaluation

We apply StocHy on four different case studies highlighting different models and tasks to be performed. All the experiments are run on a standard laptop, with an Intel Core i7-8550U CPU at 1.80GHz × 8 and with 8 GB of RAM.

4.1   Case Study 1 - Formal Verification

We consider the SHS model first presented in [2]. The model takes the form of (1), and has one discrete mode and two continuous variables representing the level of CO_2 (x_1) and the ambient temperature (x_2), respectively. The continuous variables evolve according to

$$x_{1,k+1} = x_{1,k} + \frac{\Delta}{V}(-\rho_m x_{1,k} + \varrho_c(C_{out} - x_{1,k})) + \sigma_1 w_k, \tag{6}$$

$$x_{2,k+1} = x_{2,k} + \frac{\Delta}{C_z}(\rho_m C_{pa}(T_{set} - x_{2,k}) + \frac{\varrho_c}{R}(T_{out} - x_{2,k})) + \sigma_2 w_k,$$

where Δ the sampling time $[min]$, V is the volume of the zone $[m^3]$, ρ_m is the mass air flow pumped inside the room $[m^3/min]$, ϱ_c is the natural drift air flow $[m^3/min]$, C_{out} is the outside CO_2 level $[ppm/min]$, T_{set} is the desired temperature $[°C]$, T_{out} is the outside temperature $[°C/min]$, C_z is the zone capacitance $[Jm^3/ °C]$, C_{pa} is the specific heat capacity of air $[J/ °C]$, R is the resistance to heat transfer $[°C/J]$, and $\sigma_{(.)}$ is a variance term associated to the noise $w_k \sim \mathcal{N}(0,1)$.

   We are interested in verifying whether the continuous variables remain within the safe set $X_{safe} = [405, 540] \times [18, 24]$ over 45 minutes ($K = 3$). This property can be encoded as a BLTL property, $\varphi_1 := \square^{\leq K} X_{safe}$, where $\square$ is the "*always*" temporal operator considered over a finite horizon. The semantics of BLTL is defined over finite traces, denoted by $\zeta = \{\zeta_j\}_{j=0}^K$. A trace ζ satisfies φ_1 if $\forall j \leq K, \zeta_j \in X_{safe}$, and we quantify the probability that traces generated by the SHS satisfy φ_1.

   When tackled with the method based on FAUST^2 that hinges on the computation of Lipschitz constants, this verification task is numerically tricky, in view of difference in dimensionality of the range of x_1 and x_2 within the safe set X_{safe} and the variance associated with each dimension $G_{q_0} = \begin{bmatrix} \sigma_1 & 0 \\ 0 & \sigma_2 \end{bmatrix} = \begin{bmatrix} 40.096 & 0 \\ 0 & 0.511 \end{bmatrix}$. In order to mitigate this, StocHy automatically rescales the state space so all the dynamics evolve in a comparable range.

Case study 1: Listings explaining task specification for (a) FAUST² and (b) IMDP

```
 1    // Dynamics definition          // Dynamics definition
 2    shs_t<arma::mat,int>            shs_t<arma::mat,int>
          myShs('../CS1.mat');            myShs('../CS1.mat');
 3    // Specification for FAUST^2    // Specification for IMDP
 4    // safe set                     // safe set
 5    arma::mat safe =                arma::mat safe
          {{405,540},{18,24}};            {{405,540},{18,24}};
 6    // max error                    // grid size for each dimension
 7    double eps = 1;                 arma::mat grid =
 8    // grid type                        {{0.0845,0.0845}};
 9    // (1 = uniform, 2 = adaptive)  // relative tolerance
10    int gridType = 1;               arma::mat reft = {{1,1}};
11    // time horizon                 // time horizon
12    int K = 3;                      int K = 3;
13    // task and property type       // task and property type
14    // (1 = verify safety , 2 =     // (1 = verify safety , 2 =
          verify reach-avoid,            verify reach-avoid,
15    // 3 = safety synthesis, 4 =    // 3 = safety synthesis, 4 =
          reach-avoid synthesis)         reach-avoid synthesis)
16    int p = 1;                      int p = 1;
17    // library (1 = simulator, 2 =  // library (1 = simulator, 2 =
          faust^2, 3 = imdp)             faust^2, 3 = imdp)
18    int lb = 2;                     int lb = 3;
19    // task specification           // task specification
20    taskSpec_t                      taskSpec_t
          mySpec(lb,K,p,safe,eps,gridType); mySpec(lb,K,p,safe,grid,reft);
```

Listing 1.2: (a) FAUST²   Listing 1.3: (b) IMDP

Implementation StocHy provides two verification methods, one based on FAUST² and the second based on IMDP. We parse the model from file CS1.MAT (see line 2 of Listings 1.2(a) and 1.3(b), corresponding to the two methods). CS1.MAT sets parameter values to (6) and uses a $\Delta = 15$ [*min*]. As anticipated, we employ both techniques over the same model description:

- for FAUST² we specify the safe set (X_{safe}), the maximum allowable error, the grid type (whether uniform or adaptive grid), the time horizon, together with the type of property of interest (safety or reach-avoid). This is carried out in lines 5-21 in Listing 1.2(a).
- for the IMDP method, we define the safe set (X_{safe}), the grid size, the relative tolerance, the time horizon and the property type. This can be done by defining the task specification using lines 5-21 in Listing 1.3 (b).

Finally, to run either of the methods on the defined input model, we combine the model and the task specification using `inputSpec_t<arma::mat,int> myInput(myShs,mySpec)`, then run the command `performTask(myInput)`. The verification results for both methods are stored in the RESULTS directory:

- for FAUST^2, StocHy generates four text files within the RESULTS folder: REPRESENTATIVE_POINTS.TXT contains the partitioned state space; TRANSITION_MATRIX.TXT consists of the transition probabilities of the generated abstract MC; PROBLEM_SOLUTION.TXT contains the sat probability for each state of the MC; and E.TXT stores the global maximum abstraction error.
- for IMDP, StocHy generates three text files in the same folder: STEPSMIN.TXT stores $\check{P}$ of the abstract IMDP; STEPSMAX.TXT stores $\hat{P}$; and SOLUTION.TXT contains the sat probability and the errors ε_q for each abstract state q.

Tool Method	Impl. Platform	$|\mathcal{Q}|$ [states]	Time [s]	Error ε_{max}
FAUST^2	MATLAB	576	186.746	1
FAUST^2	C++	576	51.420	1
IMDP	C++	576	87.430	0.236
FAUST^2	MATLAB	1089	629.037	1
FAUST^2	C++	1089	78.140	1
IMDP	C++	1089	387.940	0.174
FAUST^2	MATLAB	2304	2633.155	1
FAUST^2	C++	2304	165.811	1
IMDP	C++	2304	1552.950	0.121
FAUST^2	MATLAB	3481	7523.771	1
FAUST^2	C++	3481	946.294	1
IMDP	C++	3481	3623.090	0.098
FAUST^2	MATLAB	4225	10022.850	0.900
FAUST^2	C++	4225	3313.990	0.900
IMDP	C++	4225	4854.580	0.089

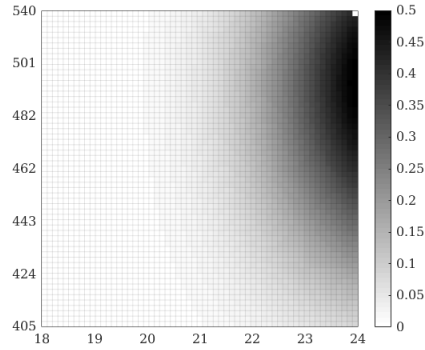Table 1: *Case study 1:* Comparison of verification results for φ_1 when using FAUST^2 vs IMDP.

Fig. 1: *Case study 1:* Lower bound probability of satisfying φ_1 generated using IMDP with 3481 states.

Outcomes We perform the verification task using both FAUST^2 and IMDP, over different sizes of the abstraction grid. We employ uniform gridding for both methods. We further compare the outcomes of StocHy against those of the FAUST^2 tool, which is implemented in MATLAB [28]. Note that the IMDP consists of $|\mathcal{Q}|+1$ states, where the additional state is the sink state $q_u = \mathcal{D} \backslash X_{safe}$. The results are shown in Table 1. We saturate (conservative) errors output that are greater than 1 to this value. We show the probability of satisfying the formula obtained from IMDP for a grid size of 3481 states in Fig. 1 – similar probabilities are obtained for the remaining grid sizes. As evident from Table 1, the new IMDP method outperforms the approach using FAUST^2 in terms of the maximum error associated to the abstraction (FAUST^2 generates an abstraction error < 1 only with 4225 states). Comparing the FAUST^2 within StocHy and the original FAUST^2 implementation (running in MATLAB), StocHy offers computational speed-up for the same grid size. This is due to the faster computation of the transition probabilities, through StocHy's use of matrix manipulations. FAUST^2 within StocHy also simplifies the input of the dynamical model description: in the original FAUST^2 implementation, the user is asked to manually input
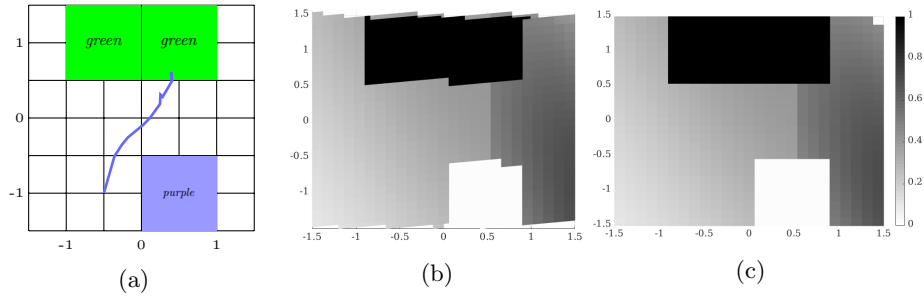
Fig. 2: *Case study 2:* (a) Gridded domain together with a superimposed simulation of trajectory initialised at $(-0.5, -1)$ within q_0, under the synthesised optimal switching strategy π^*. Lower probabilities of satisfying φ_2 for mode q_0 (b) and for mode q_1 (c), as computed by StocHy.

the stochastic kernel in the form of symbolic equations in a MATLAB script. This is not required when using StocHy, automatically generates the underlying symbolic kernels from the input state-space model descriptions.

4.2   Case Study 2 - Strategy Synthesis

We consider a stochastic process with two modes $\mathcal{Q} = \{q_0, q_1\}$, which continuously evolves according to (3) with

$$A_{q_0} = \begin{bmatrix} 0.43 & 0.52 \\ 0.65 & 0.12 \end{bmatrix}, G_{q_0} = \begin{bmatrix} 1 & 0.1 \\ 0 & 0.1 \end{bmatrix}, A_{q_0} = \begin{bmatrix} 0.65 & 0.12 \\ 0.52 & 0.43 \end{bmatrix}, G_{q_1} = \begin{bmatrix} 0.2 & 0 \\ 0 & 0.2 \end{bmatrix}, F_{q_i} = \begin{bmatrix} 0 \\ 0 \end{bmatrix},$$

and $i \in \{0, 1\}$. Consider the continuous domain shown in Fig.2a over both discrete locations. We plan to synthesise the optimal switching strategy $\pi^\star$ that maximises the probability of reaching the *green* region, whilst avoiding the *purple* one, over an unbounded time horizon, given any initial condition within the domain. This can be expressed with the LTL formula, $\varphi_2 := (\neg purple) \cup green$, where $\cup$ is the "*until*" temporal operator, and the atomic propositions {*purple, green*} denote regions within the set $X = [-1.5, 1.5]^2$ (see Fig. 2a).

Implementation We define the model dynamics following lines 3-14 in Listing 1.1, while we use Listing 1.3 to specify the synthesis task and together with its associated parameters. The LTL property φ_2 is over an unbounded time horizon, which leads to employing the IMDP method for synthesis (recall that the FAUST^2 implementation can only handle time-bounded properties, and its abstraction error monotonically increases with the time horizon of the formula). In order to encode the task we set the variable `safe` to correspond to X the grid size to 0.12 and the relative tolerance to 0.06 along both dimensions (cf. lines 5-10 in Listing 1.3). We set the time horizon `K = -1` to represent an unbounded time horizon, let `p = 4` to trigger the synthesis engine over the given specification and make `lb = 3` to use IMDP method (cf. lines 12-19 in Listing 1.3). This task specification partitions the set X into the underlying IMDP via

uniform gridding. Alternatively, the user has the option to make use of the adaptive-sequential algorithm by defining a new variable `eps_max` which characterise the maximum allowable abstraction error and then specify the task using `taskSpec_t mySpec(lb,K,p,boundary,eps_max,grid,rtol);`. Next, we define two files (PHI1 .TXT and PHI2.TXT) containing the coordinates within the gridded domain (see Fig.2a) associated with the atomic propositions *purple* and *green*, respectively. This allows for automatic labelling of the state-space over which synthesis is to be performed. Running the main file, StocHy generates a SOLUTION.TXT file within the RESULTS folder. This contains the synthesised $\pi^{\star}$ policy, the lower bound for the probabilities of satisfying φ_2, and the local errors ε_q for any region q.

Outcomes The case study generates an abstraction with a total of 2410 states, a maximum probability of 1, a maximum abstraction error of 0.21, and it requires a total time of 1639.3 [s]. In this case, we witness a slightly larger abstraction error via the IMDP method then in the previous case study. This is due the non-diagonal covariance matrix G_{q_0} which introduces a rotation in X within mode q_0. When labelling the states associated with the regions *purple* and *green*, an additional error is introduced due to the over- and under-approximation of states associated with each of the two regions. We further show the simulation of a trajectory under $\pi^{\star}$ with a starting point of $(-0.5, -1)$ in q_0, within Fig.2a.

4.3   Case Study 3 - Scaling in Continuous Dimension of Model

We now focus on the continuous dynamics by considering a stochastic process with $\mathcal{Q} = \{q_0\}$ (single mode) and dynamics evolving according to (3), characterised by $A_{q_0} = 0.8\mathbf{I}_d$, $F_{q_0} = \mathbf{0}_d$ and $G_{q_0} = 0.2\mathbf{I}_d$, where d corresponds to the number of continuous variables. We are interested in checking the LTL specification $\varphi_3 := \Box X_{safe}$, where $X_{safe} = [-1, 1]^d$, as the continuous dimension d of the model varies. Here "$\Box$" is the "*always*" temporal operator and a trace ζ satisfies φ_3 if $\forall k \geq 0$, $\zeta_k \in X_{safe}$. In view of the focus on scalability for this Case Study 3, we disregard discussing the computed probabilities, which we instead covered in Section 4.1.

Implementation Similar to Case Study 2, we follow lines 3-14 in Listing 1.1 to define the model dynamics, while we use Listing 1.3 to specify the verification task using the IMDP method. For this example, we employ a uniform grid having a grid size of 1 and relative tolerance of 1 for each dimension (cf. lines 5-10 in Listing 1.3). We set `K = -1` to represent an unbounded time horizon, `p = 1` to perform verification over a safety property and `lb = 3` to use the IMDP method (cf. lines 12-19 in Listing 1.3). In Table 2 we list the number of states required for each dimension, the total computational time, and the maximum error associated with each abstraction.

Outcomes From Table 2 we can deduce that by employing the IMDP method within StocHy, the generated abstract models have manageable state spaces, thanks to the tight error bounds that is obtained. Notice that since the number of cells per dimension is increased with the dimension d of the model, the

Dimensions [d]	2	3	4	5	6	7	8	9	10	11	12
$|\mathcal{Q}|$ [states]	4	14	30	62	126	254	510	1022	2046	4094	8190
Time taken [s]	0.004	0.06	0.21	0.90	4.16	19.08	79.63	319.25	1601.31	5705.47	21134.23
Error (ε_{max})	4.15e-5	3.34e-5	2.28e-5	9.70e-5	8.81e-6	1.10e-6	2.95e-6	4.50e-7	1.06e-7	4.90e-8	4.89e-8

Table 2: *Case study 3:* Verification results of the IMDP-based approach over φ_3, for varying dimension d of the stochastic process.

associated abstraction error ε_{max} is decreased. The small error is also due to the underlying contractive dynamics of the process. This is a key fact leading to scalability over the continuous dimension d of the model: StocHy displays a significant improvement in scalability over the state of the art [28] and allows abstracting stochastic models with relevant dimensionality. Furthermore, StocHy is capable to handle specifications over infinite horizons (such as the considered *until* formula).

4.4   Case Study 4 - Simulations

For this last case study, we refer to the CO_2 model described in Case Study 1 (Sec. 4.1). We extend the CO_2 model to capture (i) the effect of occupants leaving or entering the zone within a time step (ii) the opening or closing of the windows in the zone [2]. ρ_m is now a control input and is an exogenous signal. This can be described as a SHS comprising two-dimensional dynamics, over discrete modes in the set $\{q_0 = (E,C), q_1 = (F,C), q_2 = (F,O), q_3 = (E,O)\}$ describing possible configurations of the room (empty (E) or full (F), and with windows open (O) or closed (C)). A MC representing the discrete modes and their dynamics is in Figure 3a. The continuous variables evolve according to Eqn. (6), which now captures the effect of switching between discrete modes, as

$$x_{1,k+1} = x_{1,k} + \frac{\Delta}{V}(-\rho_m x_{1,k} + \varrho_{o,c}(C_{out} - x_{1,k})) + \mathbf{1}_F C_{occ,k} + \sigma_1 w_k, \tag{7}$$

$$x_{2,k+1} = x_{2,k} + \frac{\Delta}{C_z}(\rho_m C_{pa}(T_{set} - x_{2,k}) + \frac{\varrho_{o,c}}{R}(T_{out} - x_{2,k})) + \mathbf{1}_F T_{occ,k} + \sigma_2 w_k,$$

where the additional terms are: $\varrho_{(.)}$ is the natural drift air flow that changes depending whether the window is open (ϱ_o) or closed (ϱ_c) $[m^3/min]$; C_{occ} is the generated CO_2 level when the zone is occupied (it is multiplied by the indicator function $\mathbf{1}_F$) $[ppm/min]$; T_{occ} is the generated heat due to occupants $[\,^{\circ}C/min]$, which couples the dynamics in (7) as $T_{occ,k} = v x_{1,k} + \hbar$.

Implementation The provided file CS4.MAT sets the values of the parameters in (7) and contains the transition probability matrix representing the relationships between discrete modes. We select a sampling time $\Delta = 15$ $[min]$ and simulate the evolution of this dynamical model over a fixed time horizon $K = 8$ hours (i.e. 32 steps) with an initial CO$_2$ level $x_1 \sim \mathcal{N}(450, 25)$ $[ppm]$ and a temperature level of $x_2 \sim \mathcal{N}(17, 2)$ $[\,^{\circ}C]$. We define the initial conditions using Listing 1.4. Line 2 defines the number of Monte Carlo simulations using by
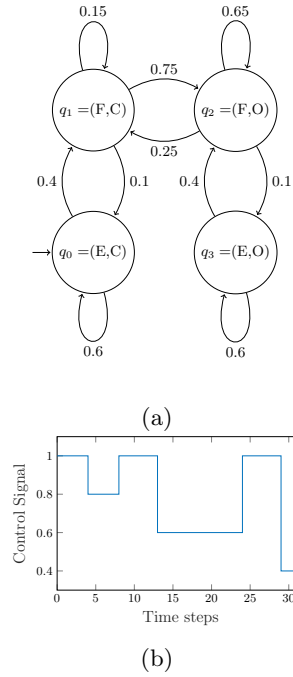
(a)

(b)

Fig. 3: *Case study 4:* (a) MC for the discrete modes of the CO_2 model and (b) input control signal.

```
1   // Number of simulations
2   int monte = 5000;
3   // Initial continuous variables
4   arma::mat x_init =
        arma::zeros<arma::mat>(2,monte);
5   // Initialise random generators
6   std::random_device rand_dev;
7   std::mt19937 generator(rand_dev());
8   // Define distributions
9   std::normal_distribution<double>
        d1{450,25};
10  std::normal_distribution<double> d2{17,2};
11  for(size_t i = 0; i < monte; ++i)
12  {
13   x_init(0,i) = d1(generator);
14   x_init(1,i) = d2(generator);
15  }
16  // Initial discrete mode q_0 = (E,C)
17  arma::mat q_init =
        arma::zeros<arma::mat>(1,monte);
18  // Definition of control signal
19  // Read from .txt/.mat file or define here
20  arma::mat u =readInputSignal("../u.txt");
21  //Combining
22  exdata_t data(x_init,u,q_init);
```

Listing 1.4: *Case study 4:* Definition of intial conditions for simulation

the variable `monte` and sets this to 5000. We instantiate the initial values of the continuous variables using the term `x_init`, while we set the initial discrete mode using the variable `q_init`. This is done using lines 4-17 which defines independent normal distribution for each of the continuous variable from which we sample 5000 points for each of the continuous variables and defines the initial discrete mode to $q_0 = (E, C)$. We define the control signal ρ_m in line 20, by parsing the `u.txt` which contains discrete values of ρ_m for each time step (see Fig. 3b). Once the model is defined, we follow Listing 1.1 to perform the simulation. The simulation engine also generates a PYTHON script, `simPlots.py`, which gives the option to visualise the simulation outcomes offline.

Outcomes The generated simulation plots are shown in Fig. 4, which depicts: (i) a sample trace for each continuous variable (the evolution of x_1 is shown in Fig. 4a, x_2 in Fig. 4b) and for the discrete modes (see Fig. 4c); and (ii) histograms depicting the range of values the continuous variables can be in during each time step and the associated count (see Fig. 4d for x_1 and Fig. 4e for x_2); and a histogram showing the likelihood of being in a discrete mode within each time step (see Fig. 4f). The total time taken to generate the simulations is 48.6 [s].
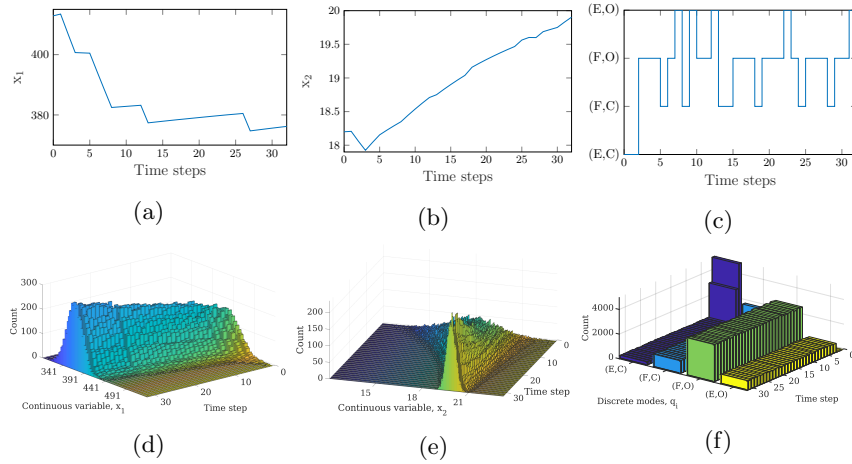
Fig. 4: *Case study 4:* Simulation single traces for continuous variables (a) x_1, (b) x_2 and discrete modes (c) q. Histogram plots with respect to time step for (d) x_1, (e) x_2 and discrete modes (f) q.

5   Conclusions and Extensions

We have presented StocHy, a new software tool for the quantitative analysis of stochastic hybrid systems. There is a plethora of enticing extensions that we are planning to explore. In the short term, we intend to: (i) interface with other model checking tools such as STORM [12] and the MODEST TOOLSET [16]; (ii) embed algorithms for policy refinement, so we can generate policies for models having numerous continuous input variables [15]; (iii) benchmarking the tool against a set of SHS models [10]. In the longer term, we plan to extend StocHy such that (i) it can employ a graphical user-interface; (ii) it can allow analysis of continuous-time SHS; and (iii) it can make use of data structures such as multi-terminal binary decision diagrams [14] to reduce the memory requirements during the construction of the abstract MDP or IMDP.

References

1. Abate, A., Prandini, M., Lygeros, J., Sastry, S.: Probabilistic reachability and safety for controlled discrete time stochastic hybrid systems. Automatica **44**(11), 2724–2734 (2008)
2. Abate, A.: Formal verification of complex systems: model-based and data-driven methods. In: Proceedings of the 15th ACM-IEEE International Conference on Formal Methods and Models for System Design, MEMOCODE 2017, Vienna, Austria, September 29 - October 02, 2017. pp. 91–93 (2017)

3. Abate, A., Blom, H., Cauchi, N., Haesaert, S., Hartmanns, A., Lesser, K., Oishi, M., Sivaramakrishnan, V., Soudjani, S., Vasile, C.I., Vinod, A.P.: ARCH-COMP18 category report: Stochastic modelling. EPiC Series in Computing **54**, 71 – 103 (2018)
4. Abate, A., Katoen, J.P., Lygeros, J., Prandini, M.: Approximate model checking of stochastic hybrid systems. European Journal of Control **16**(6), 624–641 (2010)
5. Baier, C., Katoen, J.P.: Principles of model checking. MIT press (2008)
6. Bauer, C., Frink, A., Kreckel, R.: Introduction to the GiNaC framework for symbolic computation within the C++ programming language. Journal of Symbolic Computation **33**(1), 1–12 (2002)
7. Blom, H., J. Lygeros (Eds.): Stochastic Hybrid Systems: Theory and Safety Critical Applications. No. 337 in Lecture Notes in Control and Information Sciences, Springer Verlag, Berlin Heidelberg (2006)
8. Bouissou, M., Elmqvist, H., Otter, M., Benveniste, A.: Efficient monte carlo simulation of stochastic hybrid systems. In: Proceedings of the 10 th International Modelica Conference; March 10-12; 2014; Lund; Sweden. pp. 715–725. No. 96, Linköping University Electronic Press (2014)
9. Budde, C.E., Dehnert, C., Hahn, E.M., Hartmanns, A., Junges, S., Turrini, A.: JANI: Quantitative model and tool interaction. In: International Conference on Tools and Algorithms for the Construction and Analysis of Systems. pp. 151–168. Springer (2017)
10. Cauchi, N., Abate, A.: Benchmarks for cyber-physical systems: A modular model library for building automation systems. IFAC-PapersOnLine **51**(16), 49 – 54 (2018), 6th IFAC Conference on Analysis and Design of Hybrid Systems ADHS 2018
11. Cauchi, N., Laurenti, L., Lahijanian, M., Abate, A., Kwiatkowska, M., Cardelli, L.: Efficiency through uncertainty: Scalable formal synthesis for stochastic hybrid systems. In: Proceedings of the 22Nd ACM International Conference on Hybrid Systems: Computation and Control. pp. 240–251. HSCC '19, ACM, New York, NY, USA (2019). https://doi.org/10.1145/3302504.3311805, `http://doi.acm.org/10.1145/3302504.3311805`
12. Dehnert, C., Junges, S., Katoen, J.P., Volk, M.: A storm is coming: A modern probabilistic model checker. In: Majumdar, R., Kunčak, V. (eds.) Computer Aided Verification. pp. 592–600. Springer International Publishing, Cham (2017)
13. Everdij, M.H., Blom, H.A.: Hybrid petri nets with diffusion that have intomappings with generalised stochastic hybrid processes. In: Stochastic Hybrid Systems, pp. 31–63. Springer (2006)
14. Fujita, M., McGeer, P.C., Yang, J.Y.: Multi-terminal binary decision diagrams: An efficient data structure for matrix representation. Formal methods in system design **10**(2-3), 149–169 (1997)
15. Haesaert, S., Cauchi, N., Abate, A.: Certified policy synthesis for general markov decision processes: An application in building automation systems. Performance Evaluation **117**, 75–103 (2017)
16. Hahn, E.M., Hartmanns, A., Hermanns, H., Katoen, J.P.: A compositional modelling and analysis framework for stochastic hybrid systems. Formal Methods in System Design **43**(2), 191–232 (2013)
17. Hahn, E.M., Li, Y., Schewe, S., Turrini, A., Zhang, L.: iscasMc: A web-based probabilistic model checker. In: Jones, C., Pihlajasaari, P., Sun, J. (eds.) FM 2014: Formal Methods. pp. 312–317. Springer International Publishing, Cham (2014)
18. Hartmanns, A., Hermanns, H.: The Modest Toolset: An Integrated Environment for Quantitative Modelling and Verification, pp. 593–598. Springer Berlin Heidelberg, Berlin, Heidelberg (2014)

19. Krystul, J., Blom, H.A.: Sequential monte carlo simulation of rare event probability in stochastic hybrid systems. IFAC Proceedings Volumes **38**(1), 176–181 (2005)
20. Kupferman, O., Vardi, M.Y.: Model checking of safety properties. Formal Methods in System Design **19**(3), 291–314 (2001)
21. Kwiatkowska, M., Norman, G., Parker, D.: PRISM 4.0: Verification of probabilistic real-time systems. In: Gopalakrishnan, G., Qadeer, S. (eds.) Proc. 23$^{\text{rd}}$ International Conference on Computer Aided Verification (CAV'11). LNCS, vol. 6806, pp. 585–591. Springer (2011)
22. Lahijanian, M., Andersson, S.B., Belta, C.: Formal verification and synthesis for discrete-time stochastic systems. IEEE Transactions on Automatic Control **60**(8), 2031–2045 (2015)
23. Larsen, K.G., Mikučionis, M., Muniz, M., Srba, J., Taankvist, J.H.: Online and Compositional Learning of Controllers with Application to Floor Heating. In: International Conference on Tools and Algorithms for the Construction and Analysis of Systems. pp. 244–259. Springer (2016)
24. Sanderson, C., Curtin, R.: Armadillo: a template-based c++ library for linear algebra. Journal of Open Source Software (2016)
25. Sanderson, C., Curtin, R.R.: A user-friendly hybrid sparse matrix class in C++. In: Mathematical Software - ICMS 2018 - 6th International Conference, South Bend, IN, USA, July 24-27, 2018, Proceedings. pp. 422–430 (2018)
26. Škulj, D.: Discrete time markov chains with interval probabilities. International journal of approximate reasoning **50**(8), 1314–1329 (2009)
27. Soudjani, S.E.Z.: Formal Abstractions for Automated Verification and Synthesis of Stochastic Systems. Ph.D. thesis, TU Delft (2014)
28. Soudjani, S.E.Z., Gevaerts, C., Abate, A.: FAUST^2: Formal Abstractions of Uncountable-STate STochastic processes. In: TACAS. vol. 15, pp. 272–286 (2015)
29. Summers, S., Lygeros, J.: Verification of discrete time stochastic hybrid systems: A stochastic reach-avoid decision problem. Automatica **46**(12), 1951–1961 (2010)