2^5 Years of Model Checking ⋆

Edmund M. Clarke and Qinsi Wang

Computer Science Department, Carnegie Mellon University, USA

Abstract. Model Checking is an automatic verification technique for large state transition systems. It was originally developed for reasoning about finite-state concurrent systems. The technique has been used successfully to debug complex computer hardware, communication protocols, and software. It is beginning to be used for analyzing cyber-physical, biological, and financial systems as well. The major challenge for the technique is a phenomenon called the *State Explosion Problem*. This issue is impossible to avoid in the worst case; but, by using sophisticated data structures and clever search algorithms, it is now possible to verify state transition systems with an astronomical number of states. In this paper, we will briefly review the development of Model Checking over the past 32 years, with an emphasis on model checking stochastic hybrid systems.

1 Model Checking and State Explosion Problem

Model Checking, as a framework consisting of powerful techniques for verifying finite-state systems, was independently developed by Clarke and Emerson [22] and by Queille and Sifakis [52] in the early 1980s. Over the last few decades, it has been successfully applied to numerous theoretical and practical problems [17, 20, 36, 37, 45, 63], such as verification of sequential circuit designs, communication protocols, software device drivers, security algorithms, cyber-physical systems, and biological systems. There are several major factors contributing to its success. Primarily, Model Checking is fully automated. Unlike deductive reasoning using theorem provers, this 'push-button' method neither requires proofs nor experts to check whether a finite-state model satisfies given system specifications. Besides verification of correctness, it permits bug detection as well. If a property does not hold, a model checker can return a diagnostic counterexample denoting an actual execution of the given system model leading to an error state. Such counterexamples can then help detect subtle bugs. Finally, from a practical aspect, Model Checking also works with partial specifications, which allows the separation of system design and development from verification and debugging.

Typically, a model checker has three basic components: a *modeling formalism* adopted to encode a state machine representing the system to be verified, a *specification language* based on Temporal Logics [51], and a *verification algorithm*

⋆ This paper has been partly supported by the Office of Naval Research (ONR) under grant 29749-1-1141240.

which employs an exhaustive searching of the entire state space to determine whether the specification holds or not. Because of the exhaustive search, when being applied to complex systems, all model checkers face an unavoidable problem in the worst case. The number of global states of a complex system can be enormous. Given n processes, each having m states, their asynchronous composition may have m^n states which is exponential in both the number of processes and the number of states per process. In Model Checking, we refer to this as the *State Explosion Problem.* Great strides have been made on this problem over the past 32 years for various types of real-world systems. In the following sections, we discuss major breakthroughs that have been made during the development of Model Checking, and then briefly review the work adopting these techniques for the analysis of stochastic hybrid systems, especially for probabilistic hybrid automata.

2 Major Breakthroughs

2.1 Symbolic Model Checking with OBDDs

In the original implementation of the first model checking algorithm [22], the transition system has an explicit representation using the adjacency lists. Such an enumerative representation is feasible for concurrent systems with small numbers of processes and states per process, but not adequate for very large transition systems. In the fall of 1987, McMillan made a fundamental breakthrough. He realized that by reformulating the original model checking procedure in a symbolic way where sets of states and sets of transitions are represented rather than individual states and transitions, Model Checking could be used to verify larger systems with more than 10^{20} states [18]. The new symbolic representation was based on Bryant's ordered binary decision diagrams (OBDDs) [14]. In this symbolic approach, the state graphs, which need to be constructed in the explicit model checking procedure, are described by Boolean formulas represented by OBDDs. Model Checking algorithms can then work directly on these OBDDs. Since OBDD-based algorithms are set-based, they cannot directly implement the depth-first search, and thus the property automaton should also be represented symbolically.

Since then, various refinements of the OBDD-based algorithms [10,16,35,54] have pushed the size of state space count up to more than 10^{120} [16]. The most widely used symbolic model checkers SMV [46], NuSMV [19], and VIS [13] are based on these ideas.

2.2 Partial Order Reduction

As mentioned in Section 1, the size of the parallel composition of n processes in a concurrent system may be exponential in n. Verifying a property of such a system requires inspecting all states of the underlying transition system. That is, $n!$ distinct orderings of the interleaved executions of n states need to be considered in the setting where there are no synchronizations between the individual processes. This is even more serious for software verification than for hardware verification, as software tends to be less structured than hardware. One of the most successful techniques for dealing with asynchronous systems is partial order reduction. Since the effect of concurrent actions is often independent of their ordering, this method aims at decreasing the number of possible orderings, and thus reducing the state space of the transition system that needs to be analyzed for checking properties. Intuitively, if executing two events in either order results in the same result, they are independent of each other. In this case, it is possible to avoid exploring certain paths in the state transition system.

Partial order reduction crucially relies on two assumptions. One is that all processes are fully asynchronous. The other is that the property to be checked does not involve the intermediate states. When coping with realistic systems where the processes may communicate and thus depend on one another, this approach attempts to identify path fragments of the full transition system, which only differ in the order of the concurrently executed activities. In this way, the analysis of state space can be restricted to one (or a few) representatives of every possible interleaving.

Godefroid, Peled, and Valmari have developed the concepts of incorporating partial order reduction with Model Checking independently in the early 1990's. Valmari's stubborn sets [60], Godefroid's persistent sets [33], and Peled's ample sets [49] differ on the actual details but contain many similar ideas. The SPIN model checker, developed by Holzmann [39], uses the ample-set reduction to great advantage.

2.3 Bounded Model Checking

Although Symbolic Model Checking (SMC) with OBDDs has successfully improved the scalability and is still widely used, OBDDs have multiple problems which restrict the size of models that can be checked with this method. Since the ordering of variables has to be identical for each path from the root of an OBDD to a leaf node, finding a space-efficient ordering is critical for this technique. Unfortunately, it is quite difficult, sometimes impossible, to find an order resulting in a small OBDD. Consider the formula for the middle output bit of a combinational multiplier for two n-bit numbers. It can be proved that, for all variable orderings, the size of the OBDD for this formula is exponential in n.

To further conquer the state explosion problem, Biere et al. proposed the Bounded Model Checking (BMC) using Boolean satisfiability (SAT) solvers [9]. The basic idea for BMC is quite straightforward. Given a finite-state transition system, a temporal logic property and a bound k (we assume $k \geq 1$), BMC generates a propositional logical formula whose satisfiability implies the existence of a counterexample of length k, and then passes this formula to a SAT solver. This formula encodes the constraints on initial states, the transition relations for k steps, and the negation of the given property. When the formula is unsatisfiable (no counterexample found), we can either increase the bound k until either a counterexample is found, or k reaches the upper bound on how much the transition relation would need to be unwound for the completeness, or stop if resource constraints are exceeded. As an industrial-strength model checking technique, BMC has been observed to surpass SMC with OBDDs in fast detection of counterexamples of minimal length, in saving memory, and by avoiding performing costly dynamic reordering. With a fast SAT solver, BMC can handle designs that are order-of-magnitude larger than those handled by OBDD-based model checkers.

As an efficient way of detecting subtle counterexamples, BMC is quite useful in debugging. In order to prove correctness when no counterexamples are found using BMC, an upper bound on steps to reach all reachable states needs to be determined. It has been shown that the diameter (i.e., the longest shortest path between any two states) of the state-transition system could be used as an upper bound [9]. But, it appears to be computationally difficult to compute the diameter when the state-transition system is given implicitly. Other ways for making BMC complete are based on induction [55], cube enlargement [47], Craig interpolants [48], and circuit co-factoring [32]. This problem remains a topic of active research.

An interesting variation of the original BMC is to adopt a Satisfiability Modulo Theories (SMT) solver instead of a SAT solver [24, 59]. SMT encodings in model checking have several advantages. The SMT encodings offers more powerful specification language. They use (unquantified) first-order formulas instead of Boolean formulas, and use more natural and compact encodings, as there is no need to convert high level constraints into Boolean logic formulas. These SMT encodings also make the BMC work the same for finite and infinite state systems. Above all, high level of automation has not been sacrificed for the above advantages. CBMC is a widely used Bounded model checker for ANSI-C and C++ programs [42] , having supports for SMT solvers such as Z3 [27], and Yices [28].

2.4 Counterexample-Guided Abstraction Refinement

When the model state space is enormous, or even infinite, it is infeasible to conduct an exhaustive search of the entire space. Another method of coping with the state explosion problem is to abstract away irrelevant details, according to the property under consideration, from the concrete state transition system when constructing the model. We call this approach *abstraction*. This simplification incurs information loss. Depending on the method used to control the information loss, abstraction techniques can be distinguished into either over-approximation or under-approximation techniques. The over-approximation methods enrich the behavior of the system by releasing constraints. They establish a relationship between the abstract model and the original system so that the correctness of the former implies the correctness of the latter. The downside is that they admit false negatives, where there are properties which hold in the original system but fail in the abstract model. Therefore, a counterexample found in the abstract system may not be a feasible execution in the original system. These counterexamples are called *spurious*. Conversely, the under-approximation techniques, which admit false positives, obtain the abstraction by removing irrelevant behavior from the system so that a specification violation at the abstract level implies a violation of the original system.

The *counterexample-guided abstraction refinement* (CEGAR) technique [21] integrates an over-approximation technique - existential abstraction [23] - and SMC into a unified, and automatic framework. It starts verification against universal properties with an imprecise abstraction, and iteratively refines it according to the returned spurious counterexamples. When a counterexample is found, its feasibility with regard to the original system needs to be checked first. If the violation is feasible, this counterexample is reported as a witness for a bug. Otherwise, a proof of infeasibility is used to refine the abstraction. The procedure then repeats these steps until either a real counterexample is reported, or there is no new counterexamples returned. When the property holds on the abstract model, by the Property Preservation Theorem [23], it is guaranteed for the property to be correct in the concrete systems. CEGAR is used in many software model checkers including the SLAM project [6] at Microsoft.

3 Model Checking and Stochastic Hybrid Systems

Stochastic hybrid systems (SHSs) are a class of dynamical systems that involve the interaction of discrete, continuous, and stochastic dynamics. Due to the generality, SHSs have been widely used in distinct areas, including biological systems, cyber-physical systems, and finance [12]. To describe uncertainties, randomness has been added to hybrid systems in a number of ways. A wealth of models has been promoted over the last decade. One class of models combines deterministic flows with probabilistic transitions. When state changes forced by continuous dynamics involve discrete random events, we refer to them as probabilistic hybrid automata (PHAs) [56]. PHAs are akin to Markov decision processes (MDPs) [8], which determine both the discrete and continuous successor states. When state changes involve continuous random events as well, we call them stochastic hybrid automata (SHAs) [29]. Some models allow that state changes may happen spontaneously, such as piecewise deterministic Markov processes (PDMPs) [26], which are similar to continuous-time Markov chains (CTMCs) [58]. Other models replace deterministic flows with stochastic ones, such as stochastic differential equations (SDEs) [5] and stochastic hybrid programs (SHPs) [50], where the random perturbation affects the dynamics continuously. When all such ingredients have been covered, there are models such as the general stochastic hybrid systems (GSHSs) [15,40].

The popularity of SHSs in real-world applications plays an important role as the motivation for putting a significant research effort into the foundations, analysis and control methods for this class of systems. Among various problems, one of the elementary questions for the quantitative analysis of SHSs is the probabilistic reachability problem. There are two main reasons why it catches researchers' attention. Primarily, it is motivated by the fact that most temporal properties can be reduced to reachability problems due to the very expressive hybrid modeling framework. Moreover, probabilistic state reachability is a hard and challenging problem which is undecidable in general. Intuitively, this class of problems is to compute the probability of reaching a certain set of states. The set may represent a set of certain unsafe states which should be avoided or visited only with some small probability, or dually, a set of good states which should be visited frequently.

Over the last decade, research efforts concerning SHSs are rapidly increasing. At the same time, Model Checking methods and tools for probabilistic systems, such as PRISM [44], MRMC [41], and Ymer [65], have been proposed and designed. Results related to the analysis and verification of SHSs are still limited. For instance, analysis approaches for GSHSs are often based on Monte-Carlo simulation [11,53]. Considering the hardness of dealing with the general class, efforts have been mainly placed on different subclasses [1–3, 29, 30, 34, 50, 56, 62, 66, 67].

For a decidable subclass which is called probabilistic initialized rectangular automata (PIRAs), Sproston offered a model checking procedure against the probabilistic branching time logic (PBTL) [56]. The procedure first translates PIRA to a probabilistic timed automaton (PTA), then constructs a finite-state probabilistic region graph for the PTA, and employs existing PBTL Model Checking techniques. For probabilistic rectangular automata (PRAs) which are less restricted than PIRAs, Sproston proposed a semi-decidable model checking procedure via using a forward search through the reachable state space [57].

For a more expressive class of models - probabilistic hybrid automata (PHAs), Zhang et al. abstracted the original PHA to a probabilistic automaton (PA), and then used the established Model Checking methods for the abstracting model [66]. Hahn et al. also discussed an abstraction-based method where the given PHA was translated into a n-player stochastic game using two different abstraction techniques [34]. All abstractions obtained by these methods are over-approximations, which means that the estimated maximum probability for a safety property on the abstracted model is no less than the one on the original model. Another method proposed is a SMT-based bounded Model Checking procedure [30]. We will discuss these methods in detail in the following subsections.

A similar class of models, which is widely used in the control theory, is called discrete-time stochastic hybrid systems (DTSHSs) [4]. Akin to PHAs, DTSHSs comprise nondeterministic as well as discrete probabilistic choices of state transitions. Unlike PHAs, DTSHSs are sampled at discrete time points, use control inputs to model nondeterminism, do not have an explicit notion of symbolic transition guards, and support a more general concept of randomness which can describe discretized stochastic differential equations. With regard to the system analysis, the control problem concerned can be understood as to find an optimal control policy that minimizes the probability of reaching unsafe states. A backward recursive procedure, which uses dynamic programming, was then proposed to solve the problem [1,4]. Another approach to a very similar problem as above, where a DTSHS model doesn't have nondeterministic control inputs, was presented in [2]. Compared to former method, the latter approach exploits the grid to construct a discrete-time Markov chain (DTMC), and then employs standard model checking procedures for it. This approach then had been used in [3] as an analysis procedure for the probabilistic reachability problems in the product of a DTSHS and a Büchi automaton representing a linear temporal property. Zuliani et al. also mentioned a simulation-based method for model checking DTSHSs against bounded temporal properties [67]. We refer to this method as Statistical Model Checking (StatMC). The main idea of StatMC is to generate enough simulations of the system, record the checking result returned from a trace checker from each simulation, and then use statistical testing and estimation methods to determine, with a predefined degree of confidence, whether the system satisfies the property. Although this statistical model checking procedure does not belong to the class of exhaustive state-space exploration methods, it usually returns results faster than the exhaustive search with a predefined arbitrarily small error bound on the estimated probability.

In [29], as an extension of PHAs, stochastic hybrid automata (SHAs) allow continuous probability distributions in the discrete state transitions. With respect to the verification procedure, a given SHA is firstly over-approximated by a PHA via discretizing continuous distributions into discrete ones with the help of additional uncountable nondeterminism. As mentioned, this over-approximation preserves safety properties. For the second step, the verification procedure introduced in [66] is exploited to model check the over-approximating PHA.

Another interesting work is about stochastic hybrid programs (SHPs) introduced in [50]. This formalism is quite expressive with regard to randomness: it takes stochastic differential equations, discrete probabilistic branching, and random assignments to real-valued variables into account. To specify system properties, Platzer proposed a logic called stochastic differential dynamic logic, and then suggested a proof calculus to verify logical properties of SHPs.

Among these different models and methods mentioned above, of particular interest for this paper are PHAs. In the remainder of this section, we will review two kinds of interesting techniques - abstraction-based, and BMC-based methods - proposed for probabilistic reachability and safety analysis for PHAs.

3.1 Probabilistic Hybrid Automata

Before going into the details of model checking algorithms, we recall the definitions of PHAs as given in [56].

Definition 1. (Probabilistic Hybrid Automata) *A probabilistic hybrid automaton H is a tuple $(M,\ \bar{m},\ k,\ \langle Post_m \rangle_{m \in M},\ Cmds)$ where*

- $M := \{m_1, m_2, \cdots, m_n\}$ *is a finite set of control modes.*
- $\bar{m} \subseteq M$ *is the set of initial modes.*
- k *is the dimension of the automaton, i.e. the number of system variables.*
- $\langle Post_m \rangle_{m \in M}$ *indicates continuous-time behaviors on each mode.*
- *Cmds is a finite set of probabilistic guarded commands of the following form:*
  $g \ \rightarrow \ p_1 : u_1 \ + \ \cdots \ + \ p_n : u_n,$
  where g is a predicate representing a transition guard, and p_i and u_i are the corresponding transition probability and updating function for the ith probabilistic choice respectively ($1 \leq i \leq n$).

The semantics of a probabilistic hybrid automaton is a probabilistic automaton [56] which is formally defined as follows.

Definition 2. (Semantics of Probabilistic Hybrid Automata) *The semantics of a probabilistic hybrid automaton H is a probabilistic automaton $PA[\![H]\!] = (S,\ \bar{s},\ Act,\ \mathcal{T})$, where*

- $S = M \times \mathbb{R}^k$ *denotes the (possibly uncountable) set of states.*
- $\bar{s} = (\bar{m}, 0, \cdots, 0)$ *is the set of initial states.*
- $Act = \mathbb{R}_{\geq 0} \uplus Cmds$ *describes the transition relation. Note that,* $\uplus$ *denotes the disjoint union.*
- $\mathcal{T}$*: for each* $s \in S$*, it may have two types of transitions. The first one is from command* $g \rightarrow p_1 : u_1 + \cdots + p_n : u_n$ *by* $u(s)$ *when* g *is fulfilled. The second one is from time* t *by* $Post_m(s,\ t)$.

3.2 Abstraction-based Methods

Zhang et al. presented an abstraction-based method for verifying safety properties in probabilistic hybrid automata (PHAs) [66]. The main underlying idea is to compute finite probabilistic automata (PAs) via abstractions for PHAs, and then estimate the reachability probabilities of the over-approximating PAs with the help of existing methods. In detail, the verification procedure works as follows. To construct a safe over-approximation for a given PHA, the method first considers a non-probabilistic hybrid automaton (HA) obtained by replacing probabilistic choices with nondeterministic ones. Then, this classical HA is abstracted into a finite-state abstraction, where PHAVer [31] can be employed. As the final step of the abstraction, the finite-state abstraction is decorated with probabilities via techniques known for Markov decision processes [25,38], resulting in a probabilistic finite-state automaton. Figure 1 illustrates the entire abstraction process for an example PHA. After building a safe over-approximation, the probability of reaching unsafe states in the probabilistic abstraction is estimated using value iteration [8]. Since it is computing over-approximations, the abstraction preserves the safety property: if the probability of reaching unsafe state regions in the abstracting probabilistic automaton is bounded by p, this is also the case in the original probabilistic hybrid automaton. In other words, p is a safe upper bound for the reachability probability of the original model, and if a safety property holds in in the abstraction, it holds also in the concrete system. Otherwise, refinement of the abstraction is required to obtain a potentially more precise upper bound. The realization of this refinement depends on the exploited abstraction technique. For example, PHAVer computes polyhedra to cover the continuous state-space for each discrete location. Refinement can be done by reducing the maximal widths of these polyhedra.

To estimate the maximum/minimum probability of reaching a certain state region, Hahn et al. proposed another abstraction-based approach [34]. This approach considers two different abstraction methods - a game-based approach [43] and an environment abstraction [61]. Both methods abstract a given PHA by an

n-player stochastic game, and allow us to obtain both lower and upper bounds for quantitative properties. In a bit more detail, the semantics of a PHA is firstly expressed as a (stochastic) 2-player game, where one player represents the controller and the other the environment. Both abstraction methods represent the obtained abstraction as a separate player in the game resulting in a 3-player stochastic game. Then, with the first method, this 3-player game is reduced to a 2-player stochastic game. The second method makes this new player collaborate with the player representing the environment in the PHA. By adjusting the strategy of the player denoting the abstraction to maximize (or minimize) the probability of reaching the target states, the upper (or lower) bound on the optimal reachability probability for the original automaton can be obtained from the abstraction. This approach establishes a verification as well as falsification procedure for probabilistic safety properties.

3.3 BMC-based Methods

Fränzle et al. presented an fully symbolic analyzing method of probabilistic bounded reachability problems of PHAs without resorting to over-approximation by intermediate finite-state abstractions [30]. When reasoning about PHAs, the authors use the SMT solving as a basis, and extends it by defining a novel randomized quantification over discrete variables. This method saves virtues of the SMT-based Bounded Model Checking, and harvests its recent advances in analyzing general hybrid systems. This new framework is referred to as Stochastic Satisfiability Modulo Theories (SSMT). In detail, an SSMT formula Φ can be defined in this format: $\Phi = Q_1 x_1 \in \mathrm{dom}(x_1) \cdots Q_n x_n \in \mathrm{dom}(x_n)$: ϕ, where ϕ is a quantifier-free SMT formula. $Q_1 x_1 \in \mathrm{dom}(x_1) \cdots Q_n x_n \in \mathrm{dom}(x_n)$ is the *prefix* of Φ, binding variables x_i to the quantifier Q_i. Note that not every variable occurring in ϕ has to be bound by a quantifier. In the framework of SSMT, a quantifier Q_i is either a classical *existential* quantifier, denoted as $\exists$, or a newly introduced *randomized* quantifier, denoted as $\reflectbox{R}_{d_i}$, where d_i is a finite discrete probability distribution over $\mathrm{dom}(x_i)$. The notation d_i is usually a list $\langle (v_1, p_1), \cdots, (v_m, p_m) \rangle$, where p_j is the probability of assigning x_i to v_i. The semantics of an SSMT problem is defined by the maximum probability of satisfaction, which is designed for computing the maximal reachability probability. Formally, the maximum probability of satisfaction $Pr(\Phi)$ of an SSMT formula Φ is defined recursively as follows.

- $Pr(\phi) = 1$ if ϕ is satisfiable, and 0 otherwise;
- $Pr(\exists x_i \in dom(x_i) \cdots Q_n x_n \in dom(x_n) : \phi) =$ $max_{v \in dom(x_i)}\, Pr(Q_{i+1} x_{i+1} \in dom(x_{i+1}) \cdots Q_n x_n \in dom(x_n) : \phi[v/x_i])$; and
- $Pr(\reflectbox{R}_{d_i} x_i \in dom(x_i) \cdots Q_n x_n \in dom(x_n) : \phi) =$ $\sum_{(v,p) \in dom(x_i)} p \cdot Pr(Q_{i+1} x_{i+1} \in dom(x_{i+1}) \cdots Q_n x_n \in dom(x_n) : \phi[v/x_i])$.

To analyze PHAs, the probabilistic bounded reachability problems need to be encoded in SSMT formulas. The construction procedure contains two steps. First of all, akin to the SMT-based BMC, an SMT formula is used to express

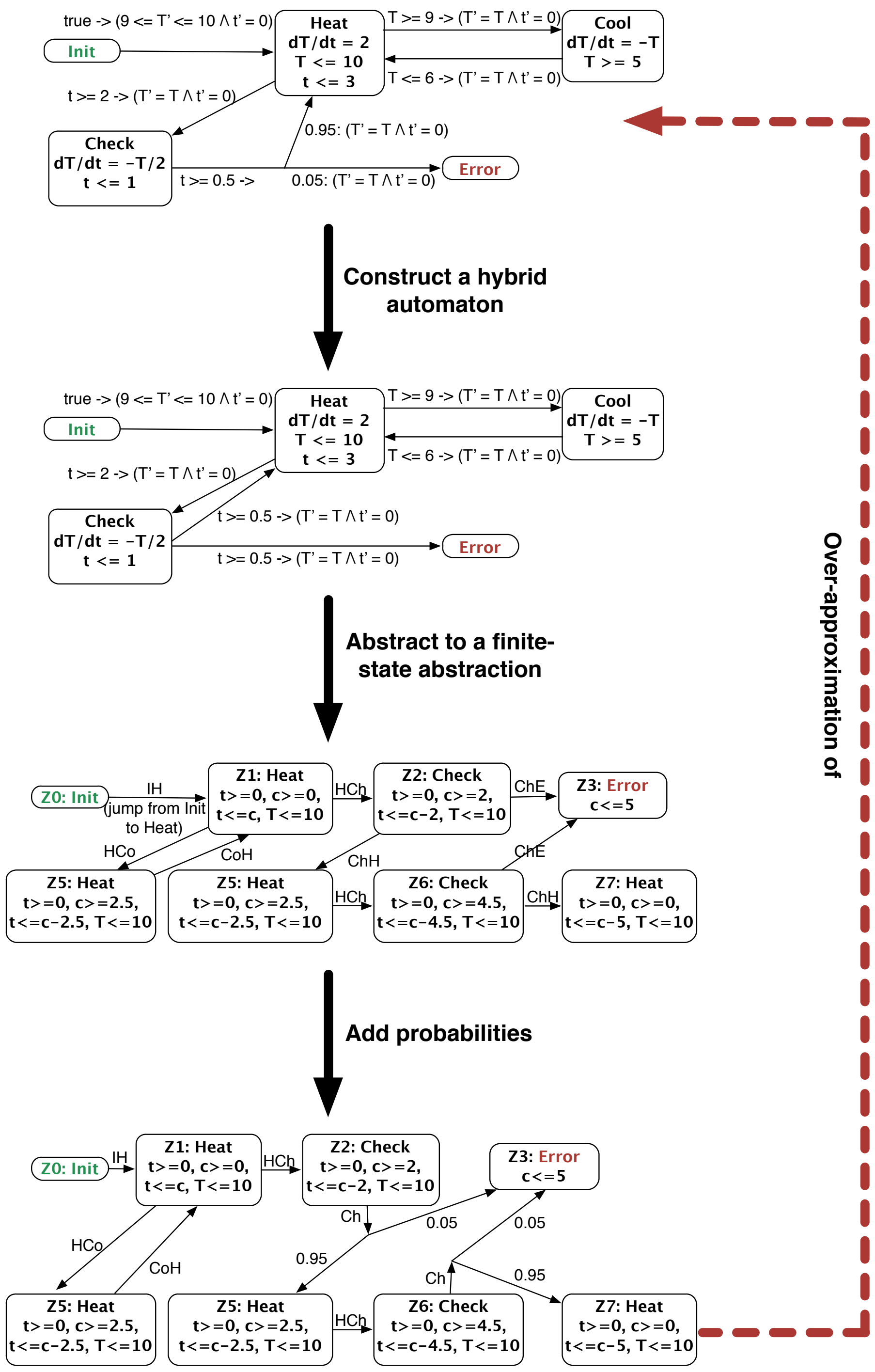
true -> (9 <= T' <= 10 ∧ t' = 0)
Init
Heat dT/dt = 2 T <= 10 t <= 3
T >= 9 -> (T' = T ∧ t' = 0)
T <= 6 -> (T' = T ∧ t' = 0)
Cool dT/dt = −T T >= 5
t >= 2 -> (T' = T ∧ t' = 0)
Check dT/dt = −T/2 t <= 1
0.95: (T' = T ∧ t' = 0)
t >= 0.5 ->
0.05: (T' = T ∧ t' = 0)
Error
Construct a hybrid automaton
t >= 0.5 -> (T' = T ∧ t' = 0)
Abstract to a finite-state abstraction
Z0: Init
IH
(jump from Init to Heat)
Z1: Heat t>=0, c>=0, t<=c, T<=10
HCh
Z2: Check t>=0, c>=2, t<=c−2, T<=10
ChE
Z3: Error c<=5
HCo
CoH
ChH
Z5: Heat t>=0, c>=2.5, t<=c−2.5, T<=10
Z6: Check t>=0, c>=4.5, t<=c−4.5, T<=10
Z7: Heat t>=0, c>=0, t<=c−5, T<=10
Add probabilities
Ch
0.05
0.95
Over-approximation of

Fig. 1: Abstraction steps for a probabilistic hybrid automaton for the thermostat

all runs of the given PHA of the given length k, ignoring both nondeterministic and probabilistic transitions. Quantification is then added to encode the missing nondeterministic and probabilistic choices. Existential quantifiers reflect nondeterministic choices and randomized quantifiers describe probabilistic transitions. With this encoding, the step-bounded reachability analysis of probabilistic hybrid automata is reduced to calculating the maximum probability of satisfaction of an SSMT formula. To compute the maximum satisfaction probability, an algorithm, which was discussed in [30], consists of three layers - a theory solver, an SMT solver, and an SSMT solver. The first two solvers are the same as the corresponding parts in widely used SMT solvers, such as Z3 [27], and CVC4 [7]. The last SSMT layer is an extension of the SMT layer to cope with existential and randomized quantification.

Another BMC-based approach to the falsification of safety properties was promoted by Wimmer et al. [64]. Although the stochastic models that they consider are discrete-time Markov chains (DTMCs), DTMCs are quite similar to PHAs except that the former do not support nondeterminism. Also, its analysis technique is closely related to the one in [30]. It works as follows. First of all, the given safety property is reduced to a state reachability one through removing edges from the given DTMC. Then, it encodes the behavior of the given DTMC with length k and the reachability property as an SAT formula as the case for SAT-based BMC. During this step, probabilistic transitions are treated as nondeterministic ones, and the transition probability matrix of the given DTMC is stored in order to be able to track the transition probabilities between states in the near future. Thereafter, the Boolean formula with the depth-bound k is solved by a SAT solver. If the formula is satisfiable, the returned satisfying assignment is used to extract a system execution of length k. The probability of this execution is computed according to the preserved probability matrix. After adding a clause representing the negation of the last returned assignment, the SAT solver is called again to find another execution reaching the target states. These steps are repeated until the SAT solver returns “unsat” for a modified formula for length k. Then, it generates a new Boolean formula for depth step $k+1$, and calls the SAT again. The overall procedure terminates if the accumulated probability of all collected system runs reaching the given unsafe states exceeds a given threshold, which is used to falsify the safety property. To reduce the number of calls to the SAT solver, the authors propose some optimizations. The most important one tries to detect loops in executions reaching the target states in order to achieve infinitely many runs from one solver invocation.

4 Conclusion and future work

Model Checking has proved to be a highly successful technology. Over the last 32 years, we have witnessed enormous progress on improving performance, on enhancing scalability, and on expanding applications in the area of Model Checking. The progress has increased our knowledge, but also opened many questions and research directions. Efforts are still needed to further conquer the state explosion problem in Software Model Checking. More effective model checking algorithms are required for real-time and hybrid systems, and are badly in need for even more complex systems, such as stochastic hybrid systems. Moreover, there are various directions, including combining model checking and static analysis, compositional model checking of complex systems, symmetry reduction and parameterized model checking, integrating model checking and theorem proving, interpreting long and complicated counterexamples, extending CEGAR for probabilistic systems, and scaling up even more!

References

1. A. Abate. *Probabilistic reachability for stochastic hybrid systems: Theory, computations, and applications.* ProQuest, 2007.
2. A. Abate, J.-P. Katoen, J. Lygeros, and M. Prandini. A two-step scheme for approximate model checking of stochastic hybrid systems. In *Proceedings of the 18th IFAC World Congress. IFAC*, 2011.
3. A. Abate, J.-P. Katoen, and A. Mereacre. Quantitative automata model checking of autonomous stochastic hybrid systems. In *Proceedings of the 14th international conference on Hybrid Systems: Computation and Control*, pages 83–92. ACM, 2011.
4. S. Amin, A. Abate, M. Prandini, J. Lygeros, and S. Sastry. Reachability analysis for controlled discrete time stochastic hybrid systems. In *Hybrid Systems: Computation and Control*, pages 49–63. Springer, 2006.
5. L. Arnold. *Stochastic Differential Equations: Theory and Applications.* Wiley - Interscience, 1974.
6. T. Ball and S. K. Rajamani. The SLAM toolkit. In *Computer-Aided Verification*, pages 260–264. Springer, 2001.
7. C. Barrett, C. L. Conway, M. Deters, L. Hadarean, D. Jovanović, T. King, A. Reynolds, and C. Tinelli. CVC4. In *Computer-Aided Verification*, pages 171–177. Springer, 2011.
8. R. Bellman. A Markovian decision process. Technical report, DTIC Document, 1957.
9. A. Biere, A. Cimatti, E. Clarke, and Y. Zhu. *Symbolic model checking without BDDs.* Springer, 1999.
10. R. Bloem, K. Ravi, and F. Somenzi. Efficient decision procedures for model checking of linear time logic properties. In *Computer-Aided Verification*, pages 222–235. Springer, 1999.
11. H. A. Blom and E. A. Bloem. Particle filtering for stochastic hybrid systems. In *43rd IEEE Conference on Decision and Control*, volume 3, pages 3221–3226. IEEE, 2004.
12. H. A. Blom, J. Lygeros, M. Everdij, S. Loizou, and K. Kyriakopoulos. *Stochastic hybrid systems: theory and safety critical applications.* Springer, 2006.
13. R. K. Brayton, G. D. Hachtel, A. Sangiovanni-Vincentelli, F. Somenzi, A. Aziz, S.-T. Cheng, S. Edwards, S. Khatri, Y. Kukimoto, A. Pardo, et al. VIS: A system for verification and synthesis. In *Computer-Aided Verification*, pages 428–432. Springer, 1996.
14. R. E. Bryant. Graph-based algorithms for boolean function manipulation. *IEEE Transactions on Computers*, 100(8):677–691, 1986.

15. M. L. Bujorianu and J. Lygeros. General stochastic hybrid systems. In *IEEE Mediterranean Conference on Control and Automation MED*, volume 4, pages 174–188, 2004.
16. J. Burch, E. M. Clarke, and D. Long. Symbolic model checking with partitioned transition relations. *Computer Science Department*, page 435, 1991.
17. J. R. Burch, E. M. Clarke, K. L. McMillan, and D. L. Dill. Sequential circuit verification using symbolic model checking. In *27th ACM/IEEE Design Automation Conference*, pages 46–51. IEEE, 1990.
18. J. R. Burch, E. M. Clarke, K. L. McMillan, D. L. Dill, and L.-J. Hwang. Symbolic model checking: 10^{20} states and beyond. In *Fifth Annual IEEE Symposium on Logic in Computer Science*, pages 428–439. IEEE, 1990.
19. A. Cimatti, E. Clarke, F. Giunchiglia, and M. Roveri. NuSMV: a new symbolic model checker. *International Journal on Software Tools for Technology Transfer*, 2(4):410–425, 2000.
20. K. Claessen, J. Fisher, S. Ishtiaq, N. Piterman, and Q. Wang. Model-checking signal transduction networks through decreasing reachability sets. In *Computer-Aided Verification*, pages 85–100. Springer, 2013.
21. E. Clarke, O. Grumberg, S. Jha, Y. Lu, and H. Veith. Counterexample-guided abstraction refinement. In *Computer-Aided Verification*, pages 154–169. Springer, 2000.
22. E. M. Clarke and E. A. Emerson. *Design and synthesis of synchronization skeletons using branching time temporal logic.* Springer, 1982.
23. E. M. Clarke, O. Grumberg, and D. E. Long. Model checking and abstraction. *ACM transactions on Programming Languages and Systems (TOPLAS)*, 16(5):1512–1542, 1994.
24. L. Cordeiro, B. Fischer, and J. Marques-Silva. SMT-based bounded model checking for embedded ANSI-C software. *IEEE Transactions on Software Engineering*, 38(4):957–974, 2012.
25. P. R. D'Argenio, B. Jeannet, H. E. Jensen, and K. G. Larsen. Reachability analysis of probabilistic systems by successive refinements. In *Process Algebra and Probabilistic Methods. Performance Modelling and Verification*, pages 39–56. Springer, 2001.
26. M. H. Davis. Piecewise-deterministic Markov processes: A general class of non-diffusion stochastic models. *Journal of the Royal Statistical Society. Series B (Methodological)*, pages 353–388, 1984.
27. L. De Moura and N. Bjørner. Z3: An efficient SMT solver. In *Tools and Algorithms for the Construction and Analysis of Systems*, pages 337–340. Springer, 2008.
28. B. Dutertre and L. De Moura. The yices SMT solver. *Tool paper at http://yices. csl. sri. com/tool-paper. pdf*, 2:2, 2006.
29. M. Fränzle, E. M. Hahn, H. Hermanns, N. Wolovick, and L. Zhang. Measurability and safety verification for stochastic hybrid systems. In *Proceedings of the 14th international conference on Hybrid Systems: Computation and Control*, pages 43–52. ACM, 2011.
30. M. Fränzle, H. Hermanns, and T. Teige. Stochastic satisfiability modulo theory: A novel technique for the analysis of probabilistic hybrid systems. In *Hybrid Systems: Computation and Control*, pages 172–186. Springer, 2008.
31. G. Frehse. PHAVer: Algorithmic verification of hybrid systems past HyTech. In *Hybrid Systems: Computation and Control*, pages 258–273. Springer, 2005.
32. M. K. Ganai, A. Gupta, and P. Ashar. Efficient SAT-based unbounded symbolic model checking using circuit co-factoring. In *Proceedings of the 2004 IEEE/ACM International conference on Computer-Aided Design*, pages 510–517. IEEE, 2004.

33. P. Godefroid. Using partial orders to improve automatic verification methods. In *Computer-Aided Verification*, pages 176–185. Springer, 1991.
34. E. M. Hahn, G. Norman, D. Parker, B. Wachter, and L. Zhang. Game-based abstraction and controller synthesis for probabilistic hybrid systems. In *Quantitative Evaluation of Systems (QEST), 2011 Eighth International Conference on*, pages 69–78. IEEE, 2011.
35. R. Hardin, R. Kurshan, S. Shukla, and M. Vardi. A new heuristic for bad cycle detection using BDDs. In *Computer-Aided Verification*, pages 268–278. Springer, 1997.
36. K. Havelund and N. Shankar. Experiments in theorem proving and model checking for protocol verification. In *FME'96: Industrial Benefit and Advances in Formal Methods*, pages 662–681. Springer, 1996.
37. T. A. Henzinger, R. Jhala, R. Majumdar, and G. Sutre. Software verification with BLAST. In *Model Checking Software*, pages 235–239. Springer, 2003.
38. H. Hermanns, B. Wachter, and L. Zhang. Probabilistic CEGAR. In *Computer-Aided Verification*, pages 162–175. Springer, 2008.
39. G. J. Holzmann. The model checker SPIN. *IEEE Transactions on Software Engineering*, 23(5):279–295, 1997.
40. J. Hu, J. Lygeros, and S. Sastry. Towards a theory of stochastic hybrid systems. In *Hybrid Systems: Computation and Control*, pages 160–173. Springer, 2000.
41. J.-P. Katoen, M. Khattri, and I. S. Zapreev. A Markov reward model checker. In *Second International Conference on the Quantitative Evaluation of Systems*, pages 243–244. IEEE, 2005.
42. D. Kroening and M. Tautschnig. CBMC – C bounded model checker. In *Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, volume 8413 of *LNCS*, pages 389–391. Springer, 2014.
43. M. Kwiatkowska, G. Norman, and D. Parker. Game-based abstraction for Markov decision processes. In *Third International Conference on Quantitative Evaluation of Systems*, pages 157–166. IEEE, 2006.
44. M. Kwiatkowska, G. Norman, and D. Parker. PRISM 4.0: Verification of probabilistic real-time systems. In G. Gopalakrishnan and S. Qadeer, editors, *Proc. 23rd International Conference on Computer-Aided Verification*, volume 6806 of *LNCS*, pages 585–591. Springer, 2011.
45. W. Marrero, E. Clarke, and S. Jha. Model checking for security protocols. Technical report, DTIC Document, 1997.
46. K. L. McMillan. *Symbolic model checking*. Springer, 1993.
47. K. L. McMillan. Applying SAT methods in unbounded symbolic model checking. In *Computer-Aided Verification*, pages 250–264. Springer, 2002.
48. K. L. McMillan. Interpolation and SAT-based model checking. In *Computer Aided Verification*, pages 1–13. Springer, 2003.
49. D. Peled. All from one, one for all: on model checking using representatives. In *Computer-Aided Verification*, pages 409–423. Springer, 1993.
50. A. Platzer. Stochastic differential dynamic logic for stochastic hybrid programs. In *Automated Deduction–CADE-23*, pages 446–460. Springer, 2011.
51. A. Pnueli. The temporal logic of programs. In *18th Annual Symposium on Foundations of Computer Science, 1977*, pages 46–57. IEEE, 1977.
52. J.-P. Queille and J. Sifakis. Specification and verification of concurrent systems in CESAR. In *International Symposium on Programming*, pages 337–351. Springer, 1982.

53. D. Riley, X. Koutsoukos, and K. Riley. Modeling and simulation of biochemical processes using stochastic hybrid systems: The sugar cataract development process. In *Hybrid Systems: Computation and Control*, pages 429–442. Springer, 2008.
54. R. Sebastiani, S. Tonetta, and M. Y. Vardi. Symbolic systems, explicit properties: on hybrid approaches for LTL symbolic model checking. In *Computer-Aided Verification*, pages 350–363. Springer, 2005.
55. M. Sheeran, S. Singh, and G. Stålmarck. Checking safety properties using induction and a SAT-solver. In *Formal Methods in Computer-Aided Design*, pages 127–144. Springer, 2000.
56. J. Sproston. Decidable model checking of probabilistic hybrid automata. In *Formal Techniques in Real-Time and Fault-Tolerant Systems*, pages 31–45. Springer, 2000.
57. J. Sproston. Model checking for probabilistic timed and hybrid systems. In *PhD thesis*. School of Computer Science, University of Birmingham, 2001.
58. H. C. Tijms. *A first course in stochastic models*. John Wiley and Sons, 2003.
59. C. Tinelli. SMT-based model checking. In *NASA Formal Methods*, page 1, 2012.
60. A. Valmari. Stubborn sets for reduced state space generation. In *Advances in Petri Nets 1990*, pages 491–515. Springer, 1991.
61. B. Wachter and L. Zhang. Best probabilistic transformers. In *Verification, Model Checking, and Abstract Interpretation*, pages 362–379. Springer, 2010.
62. 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.
63. Q. Wang, P. Zuliani, S. Kong, S. Gao, and E. M. Clarke. SReach: Combining statistical tests and bounded model checking for nonlinear hybrid systems with parametric uncertainty. Technical report, Computer Science Department, Carnegie Mellon University, 2014.
64. R. Wimmer, B. Braitling, and B. Becker. Counterexample generation for discrete-time Markov chains using bounded model checking. In *Verification, Model Checking, and Abstract Interpretation*, pages 366–380. Springer, 2009.
65. H. L. Younes. Ymer: A statistical model checker. In *Computer-Aided Verification*, pages 429–433. Springer, 2005.
66. L. Zhang, Z. She, S. Ratschan, H. Hermanns, and E. M. Hahn. Safety verification for probabilistic hybrid systems. *European Journal of Control*, 18(6):572–587, 2012.
67. P. Zuliani, A. Platzer, and E. M. Clarke. Bayesian statistical model checking with application to simulink/stateflow verification. In *Proceedings of the 13th ACM international conference on Hybrid Systems: Computation and Control*, pages 243–252. ACM, 2010.