Statistical Model Checking for Distributed Probabilistic-Control Hybrid Automata with Smart Grid Applications

João Martins[1,2], André Platzer[1], and João Leite[2]

[1] Computer Science Department, Carnegie Mellon University, Pittsburgh PA
{jmartins,aplatzer}@cs.cmu.edu
[2] CENTRIA and Departamento de Informática, FCT, Universidade Nova de Lisboa
jleite@di.fct.unl.pt

Abstract. The power industry is currently moving towards a more dynamical, intelligent power grid. This *Smart Grid* is still in its infancy and a formal evaluation of the expensive technologies and ideas on the table is necessary before committing to a full investment. In this paper, we argue that a good model for the Smart Grid must match its basic properties: it must be hybrid (both evolve over time, and perform control/computation), distributed (multiple concurrently executing entities), and allow for asynchronous communication and stochastic behaviour (to accurately model real-world power consumption). We propose Distributed Probabilistic-Control Hybrid Automata (DPCHA) as a model for this purpose, and extend Bounded LTL to Quantified Bounded LTL in order to adapt and apply existing statistical model-checking techniques. We provide an implementation of a framework for developing and verifying DPCHAs. Finally, we conduct a case study for Smart Grid communications analysis.

1 Introduction

The ultimate promise of the Smart Grid is that of a more stable, energy-efficient, adaptable, secure, resilient power grid, while delivering cheaper electricity. Currently, energy consumption follows fairly predictable patterns that need to be very closely matched by power generation (otherwise blackouts or damage to the infrastructure may occur). There are peak hours (e.g., people arrive home on a hot summer day and turn on the AC), and low hours (e.g., during the night). Certain power generators run permanently at 100% capacity, providing support to what is known as the base load. More adaptable but more expensive generators change their output to match demand, varying the price of energy throughout the day. During peak hours, it might be necessary to turn on highly adaptable and expensive peak load generators, making energy extremely expensive for those few hours.

One of the core ideas of the Smart Grid is that generators will no longer passively adapt to consumption. Instead, power consumers both at the lower level (e.g., appliances such as washing machines) and higher level (utilities serving

S. Qin and Z. Qiu (Eds.): ICFEM 2011, LNCS 6991, pp. 131–146, 2011.
© Springer-Verlag Berlin Heidelberg 2011

some geographical area) will feed their desired consumption back into the Grid. Indeed, utilities and home-owners have already begun deploying smart meters and appliances that make available more detailed, up-to-date energy consumption information. This gives the smarter Grid better foresight, increasing its robustness and its ability to reschedule non-critical appliances (e.g., the dishwasher) to off-peak hours, reducing energy costs.

Given the size and criticality of the power infrastructure, it is clear that Smart Grid technologies have to be analysed very carefully. Furthermore, the cost of providing real test-beds for all technologies is prohibitive, especially if the infrastructure can sustain damaged when things go wrong. Formal verification, on the other hand, allows us to study a *model* of the system in question, sidestepping the above issues. Given an appropriate model, we may then check the system for properties: how much can be saved with appliance rescheduling? Do Smart Meters help in predicting and optimizing load? Does that prediction help balance load across generators? Not only are the answers to these questions useful in furthering our understanding of the technologies, they also give us hints about how they may be improved upon before real-world deployment.

For the above reasons, we believe that formal verification of new technologies is fundamental for the Smart Grid. The first step in this endeavour is to find adequate models for the Grid. The models need to be flexible and generic so they can be reused for multiple projects and ideas, yet match the nature and properties of the Grid. Forcing models that do not fit the properties of the Grid lead to modelling idiosyncrasies, effectively making modelling and verification much harder than they need to or should be (humanly and computationally).

What, then, are the properties of the Smart Grid? Most importantly, it is a *cyber-physical* system. Its infrastructure exists in the real-world and follows the laws of physics (e.g., a generator increasing its output), but it also contains control components that make decisions and change the state instantaneously. Thus, it is a hybrid system, i.e., has both continuous and discrete dynamics. Some mathematical models and verification techniques for hybrid systems extend automata theory by allowing continuous evolution to occur in each state (e.g., [6,3,1]), but verification is known to be undecidable for most cases [3]. Differential Dynamic Logic ($d\mathcal{L}$) allows the specification of both the properties and the behaviour of a hybrid system [9] and provides a proof calculus for verification.

Another property of the Smart Grid is that it is *distributed*. The Grid is not one monolithic system, but composed of a large number of distributed and *communicating* entities, from the power generators down to electrical substations to the utilities, households, appliances and meters. All of these elements coexist, communicate and cooperate with one another in real-time. Most automata models support the notion of composition, allowing a fixed number of automata to execute concurrently. The Grid, however, is *dynamically* distributed: appliances are turned on and off, power lines can be cut, and meters may fail. The model must allow entities/elements to enter, leave and communicate as part of the system dynamics. Dynamic I/O Automata [5] allow a dynamic number of elements, but are not hybrid. Quantified Differential Dynamic Logic ($Qd\mathcal{L}$) [10] also allows

a dynamic number of elements and is hybrid, but like Dynamic I/O automata, has a shared-memory communication model. Proposals for Smart Grid communication are currently based on IP and message-passing protocols, so that forcing a fundamentally different shared-memory paradigm appears unwise.

Finally, the Grid exhibits *stochastic behaviour*. As we have seen, power consumption follows known but not completely fixed patterns so that using nondeterminism to model these patterns encodes little information about the actual behaviour of the Grid. Hybrid system models are generally non-deterministic, attempting to verify safety properties that hold even in the worst-case. In the Grid, worst-case scenarios (e.g., all lines cut at the same time, all appliances always on) are sure to bring about a complete collapse, and most safety properties *will not* hold! Alternatively, a probabilistic model enables 1) a more detailed and accurate representation of the Grid's consumption patterns and 2) a more comprehensive quantitative study. Since we know most interesting properties are not always true, we may *estimate the probability* that they hold. This precludes $Qd\mathcal{L}$ [10], which is not stochastic, whereas Stochastic $d\mathcal{L}$ [11] is not distributed. I/O Automata have a stochastic extension, but it is not distributed [5].

Petri Nets are inherently dynamically distributed and there have been stochastic and differential extensions [12,2]. However, the notion of markings flowing place to place is not one that we find when designing the participants of the Smart Grid. They would be composed of multiple markings scattered over different places of the Petri graph. Several entities of the same class (e.g., microwaves), sharing the same "control" graph, would create a multitude of markings superimposed in the same places. Markings would have to be associated with one another to keep track of the entities as a whole, instead of considering an entity as an indivisible structure. There is also no immediately available communication mechanism for transmitting messages (with a payload). In conclusion, while many Petri Net variants feature mechanisms very similar to those of the Smart Grid, it is our belief their actual implementation is generally differs enough to warrant the Grid a model of its own.

In [8] the state of the system is given by a composition of objects and messages. All objects evolve continuously as long as no invariant is violated, and fire probabilistic discrete transitions when they are. Asynchronous communication is achieved by assigning a delivery time to all messages upon creation. The decision to do a discrete or continuous transition depends exclusively on whether an invariant is violated, making the dynamics of this model very restricted.

In summation, we need a model that is composed of many different entities. These entities should be able to enter and leave the system at will, representing failures and appliances being turned on or off. Furthermore, the entities must be able to communicate asynchronously: given the scale of the Grid and the impact of message delivery delays, it is unrealistic to assume synchronous (instant) communication. Finally, the system must be able to behave probabilistically, in order to encode uncertain environments, e.g., power consumption. To the best of our knowledge, no existing model naturally incorporates all of these properties.

In this paper, we propose Distributed Probabilistic-Control Hybrid Automata (DPCHA) as such a model. We take care that the system can be easily sampled from (to obtain execution traces), with the objective of applying existing efficient statistical verification techniques.

Previous work has shown that statistical model checking (SMC) is a promising approach for the verification of probabilistic systems [4,15,14]. Given a property and a model, SMC techniques will repeatedly sample traces from the model and check if they satisfy the property. Every new result provides more information on whether the property holds for arbitrary traces. While known to be unsound, SMC can arbitrarily approximate the *probability* that the property holds very efficiently, making many otherwise intractable problems accessible.

Logics traditionally used in the specification of properties for these hybrid systems generally consider a fixed state-space. This makes them insufficient for the representation of properties of distributed systems. We propose Quantified Bounded Linear Temporal Logic, an extension to Bounded Linear Temporal Logic that handles the dynamic state space of DPCHA using quantification over the elements of the system. A similar phenomenon has been studied in the context of Java threads [13], for example, but not for cyber-physical systems.

The main contribution of this paper is the proposal of a model that naturally adapts to Smart Grid scenarios and for which these techniques are applicable, enabling meaningful studies of the system.

We present some technical background in Section 2, and our DPCHA model in Section 3. To specify properties we define QBLTL in Section 4. We briefly explain Bayesian statistical model checking in Section 5, and develop an initial case study in the Smart Grid domain in Section 6. We conclude in Section 7.

2 Preliminaries

Before developing the distributed model, we will begin by introducing how a single entity behaves (e.g., microwave, generator). Thus, we briefly recall discrete-time hybrid automata (DTHA) [15]. Each entity must have a *state*, e.g., current and desired power output of a generator. The entity is in a *location* that specifies how the state should *flow* as time passes, e.g., spooling up generator to match desired output. Finally, the entity may decide to *jump* from one location to another, e.g., the microwave switches to "defrost". We refer to an entity's *situation* as the pair of its location and state. Thus, DTHA are hybrid because they allow continuous evolution (time passing) and discrete transitions between locations.

Definition 1 (DTHA). *A discrete-time hybrid automaton consists of*

- $\langle Q, E\rangle$*, a "control graph" with* Q *as* locations *and* $E \subseteq Q \times Q$ *as the* edges
- $\mathbb{R}^n$ *is the state space of the automaton's state*
- $\mathit{jump}_e : \mathbb{R}^n \rightharpoonup \mathbb{R}^n$*, a partial function defining how the state changes when jumping along edge* e
- $\varphi_q : \mathbb{R}_{\geq 0} \times \mathbb{R}^n \to \mathbb{R}^n$, flows. $\varphi_q(t; x)$ *is the result of a continuous evolution at location* $q \in Q$ *after time* t *when starting in state* $x \in \mathbb{R}^n$
- $(q_0; x_0) \in Q \times \mathbb{R}^n$*, an initial situation*

Suppose an entity is in a situation $(q; x)$. It may jump along an edge e originating from q, updating its state according to $\texttt{jump}_e$. Or it may remain in q for some time t, updating its state according to the flow φ_q (which can be, for instance, the solution of a differential equation system). Since there might be multiple options for the next step, the automaton is non-deterministic.

Definition 2 (Transition relation for DTHA). *The transition relation for a DTHA is defined as:*

$$(q; x) \xrightarrow{\alpha} (\overline{q}; \overline{x}), \text{ where}$$

- *When* $\alpha = t \in \mathbb{R}_{\geq 0}$ *is a time, then* $(q; x) \xrightarrow{t} (q; \overline{x})$ *iff* $\overline{x} = \varphi_q(t; x)$
- *When* $\alpha = e \in E$ *is an edge from* q *to* $\overline{q}$*, then* $(q; x) \xrightarrow{e} (\overline{q}; \overline{x})$ *iff* $\overline{x} = \texttt{jump}_e(x)$

It is only possible to jump along an edge e if the entity's state is in the domain of $\texttt{jump}_e$. In this case, we say that e is *enabled* and that $\texttt{jump}_e$ works as a *guard* for e. A *scheduler* $\delta : Q \times \mathbb{R}^n \to \mathbb{R}_{\geq 0} \cup E$ that, given a situation $(q; x)$, decides the next action α (a flow or an edge to jump), can be applied repeatedly to an entity's situation to obtain a *trace* for that entity.

We now have defined the dynamics of a single entity. To make it behave probabilistically, all we need to do is to make $\delta : Q \times \mathbb{R}^n \to \mathcal{D}(E \cup \mathbb{R}_{\geq 0} \cup \mathbb{R}_{\geq 0})$ return a *probability distribution* over all possible actions instead of a single action α. Sampling from this distribution gives the entity its next step.

3 Distributed Probabilistic Control Hybrid Automata

A single entity's behaviour is given by its control graph, flows and jumps. We have mentioned that traditional notions of automata composition are not dynamic enough for the Grid. *Configuration automata* [5] keep track of multiple executing entities (also automata) that can enter and leave the system, resulting in two layers of automata that have no particular intuition in the Grid. Furthermore, the automata force communication to be immediate and synchronous.

Instead, like in Petri Nets, we assume all the control graphs are given, so we understand how microwaves (for example) behave, but are not required to know how many. With these control graphs, jumps and flows for each type of entity (e.g., microwave, meter), it is trivial to encode the behaviour of all types of entities in one global control graph.

The global control graph accommodates several entities, not unlike Petri Net markings. These entities are characterised by their situation and execute like a single entity from the previous section. This is the basic intuition for the notions of Distributed Discrete-Time Hybrid Automata (DDTHA) and Distributed Probabilistic Control Hybrid Automata (DPCHA) that we define below.

To maintain a sensible global notion of time, the DDTHA will do a continuous transition only if all executing entities decide to do so. If any entity decides to do a jump, then the other entities must either jump as well or *flounder* (i.e. doing a discrete transition with no effect), keeping time unchanged. In this sense, discrete transitions take precedence over continuous transitions, but consume no time.

We must still address the ability to communicate and to allow entities to enter or exit the system. We reduce these two concerns (communication and dynamic number of entities) to five elementary *actions*: `new`$[N]$, `die`, `snd`$[l][T]$, `recv`$[l][R]$, `jmp`. Each edge in the control graph features an action. When an entity jumps along that edge, its action is executed. `jmp` is a null action so that the entity simply follows $\texttt{jump}_e$. `new`$[N]$ additionally creates a new entity with a situation specified by a function N, and `die` makes the jumping entity exit the system. `snd`$[l][T]$ and `recv`$[l][R]$ send and receive messages through a *channel* determined by function l. The content of the sent message is given by function T, whereas the receiving entity's state is updated according to R, taking into account both its current state and message content. To achieve asynchronous communication, sent messages are stored at the global automaton level in a "buffer", and are removed later when received. The content of messages is a real vector computed by function T, and affects the receiver's state according to R. Thus, each action is characterised by functions determining exactly how it is executed. We let $\mathcal{A}_e$ denote the action of edge e, which happens in addition to the effects of $\texttt{jump}_e$.

In summary, the control graph retains its general structure, but annotates each edge with actions for communication and dynamism. Instead of a single initial situation, we have an initial set of active entities and an initial situation for each entity. Active entities all evolve time-synchronously, each following the rules of DTHA. As entities jump along edges, they execute the associated actions, enabling communication and complex interactions.

Definition 3 (DDTHA). *A Distributed Discrete-Time Hybrid Automaton is composed of*

- $\mathbb{R}^n$*, the state space for each entity, with* $n \in \mathbb{N}$
- $\mathbb{R}^{n_m}$*, the state space of each message's content, with* $n_m \in \mathbb{N}$
- $\mathcal{A} = \{\texttt{new}[N], \texttt{die}, \texttt{snd}[l][T], \texttt{recv}[l][R], \texttt{jmp}\}$*, the set of all actions, with channel specification functions* $l : \mathbb{R}^n \to C$*, new entity creation functions* $N : \mathbb{R}^n \to Q \times \mathbb{R}^n$*, message transmission functions* $T : \mathbb{R}^n \to \mathbb{R}^{n_m}$ *and message reception functions* $R : \mathbb{R}^{n_m} \times \mathbb{R}^n \to \mathbb{R}^n$
- $\langle Q, E \rangle$*, control graph with locations* Q *and edges* $E \subseteq Q \times \mathcal{A} \times Q$
- $\texttt{jump}_e : \mathbb{R}^n \times \mathbb{R}^n$*, a relation when* $\mathcal{A}_e = \texttt{recv}$ *or function* $\texttt{jump}_e : \mathbb{R}^n \to \mathbb{R}^n$ *otherwise, defining acceptable state updates when jumping along edge* e
- φ_q*, as in Def. 1*
- L*: a (countable) set of entity identifiers*
- $A_0 \subset L$*: a finite set of initial* active entities
- $S_0 : A_0 \to Q \times \mathbb{R}^n$*, a function with a* situation *for each initial active entity*
- C*, a (countable) set of communication channels*

The state of a DDTHA consists of the situations of all its entities, active and past. Information about past entities is kept so that checking properties of them is well-defined. The state also maintains a set of "in transit" messages (sent but not received), enabling asynchrony of communication.

Definition 4 (State of a DDTHA). *The state of a DDTHA is given by* $AS = (A, S, M)$ *with*

- $A \subset L$ *a finite set of the labels of active entities*
- $S : L \rightharpoonup Q \times \mathbb{R}^n$*, a partial function with the situation of active/past entities*
- $M \subseteq C \times \mathbb{R}^{n_m}$*, a set of unreceived messages and respective channels*

One interesting issue arises when an entity a decides (through a scheduler) to flow for t time units but sometime at $t' < t$ some other entity b finishes its own flow and schedules a jump. In this situation, the DDTHA also schedules a discrete transition, but a cannot be allowed to reevaluate its previous decision to flow for t time (e.g., the washing machine should not stop because someone turned on the TV). Therefore, we assume without loss of generality that each entity stores in its state (e.g., in its first coordinate) how long it must still flow, denoted by δ-time. In state (A, S, M), an entity a's δ-time is denoted $\delta\text{-time}^S(a)$. When the DDTHA schedules a discrete transition, any entity with non-zero δ-time will flounder, thus only truly rescheduling once its flow decision finishes executing.

Another important element of discrete transitions is message reception. There must be an injective mapping from "in transit" messages to receiving entities so that they get exactly one message. Injection ensures each receiving entity gets at most one message. Of course, the entities must react accordingly, and received/sent messages are removed/added to M.

The following example justifies our choice of probabilities and asynchronous communication and illustrates a simplified modelling of the Smart Grid.

Example 1. Newest generation smart meters feed up-to-date information into the Grid, including power consumption from the appliance level up to substation and utility levels and so on; the Grid also needs to match generator output with power consumption. There is a Grid control infrastructure that maintains this fragile balance.

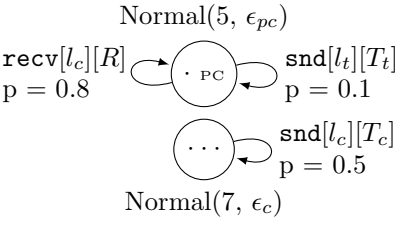

Fig. 1. Simplified Smart Grid

An ideal model for this scenario would have entities representing appliances, consuming energy, shutting off and powering on, and sending messages into the Grid through channel l_c. Another (unique) entity, called the Power Controller (PC), would react to messages from l_c and control generator output by sending it messages through l_t. Unfortunately, it is computationally infeasible to model every appliance in a country-wide Grid (except maybe Monaco or Nauru!). A sensible simplification instead represents *classes* of appliances that get turned on at around the same time for a very similar duration, like ACs/computers in offices, the TVs at home, etc. Ideally, the exact times and durations are given by probability distributions, simulating real-world behaviour. We obtain a much more manageable number of entities by using *classes* of appliances instead of *individual* appliances.

Real networks become congested so that messages are not delivered instantly. Probabilities can be used to simulate this delay: the PC's choice for flow time, for instance, could be given as a normal random variable Normal(5, ϵ_{pc}). The PC

then waits around 5s before getting the message. We may assign its individual scheduler a 0.8 probability of jumping along a `recv` edge, so that there is a 0.2 chance it will be further delayed (simulating message loss and retransmission or congestion). These remaining 0.2 can be split between sending control messages to generators or deciding to do another flow.

To justify our choice of asynchronous communication, suppose that whenever an appliance class entity sends a message, the PC is forced to ignore its δ-time and synchronise to receive that message. The PC is deviating from its original specification of continuous evolution according to a Normal RV, which would make the semantics, meaning and usefulness of the model unclear.

Definition 5 (Transition for DDTHA). *The transition relation of a DDTHA is defined inductively as*

$$(A, S, M) \xrightarrow{\alpha} (\overline{A}, \overline{S}, \overline{M})$$

where A is non-empty, $\alpha \in \mathbb{R}_{\geq 0} \cup (A \to (E \cup \mathbb{R}_{\geq 0} \cup \{F\}))$, iff

- *If $\alpha = t \in \mathbb{R}_{\geq 0}$, then $\forall_{a \in A}\ a \in \overline{A}$, $S(a) \xrightarrow{t} \overline{S}(a)$, $\delta\text{-time}^{\overline{S}}(a) = \delta\text{-time}^{S}(a) - t$ and $\delta\text{-time}^{\overline{S}}(a) \geq 0$*
- *If $\alpha = \tau : A \to E \cup \mathbb{R}_{\geq 0} \cup \{F\}$, then there are partial injective mappings $\mu_c : \{(c, \mathbb{R}^{n_m}) \in M\} \to \{a \in A : \tau(a) = (q, \mathtt{recv}[l][R], \overline{q})\}$ from messages of each channel to entities scheduled to receive on that channel, $\overline{M} = (M \setminus \bigcup\{range(\mu_c) : c \in C\}) \cup \{(l(S(a)), T(S(a))) : a \in A, \tau(a) = (q, \mathtt{snd}[l][T], \overline{q})\}$ and $\forall_{a \in A}$ if $\delta\text{-time}^{S}(a) > 0$, then $\tau(a) = F$, $a \in \overline{A}$ and $S(a) = \overline{S}(a)$; otherwise if $\delta\text{-time}^{S}(a) = 0$*
  - *If $\tau(a) = t \in \mathbb{R}_{\geq 0}$, then $S(a) = \overline{S}(a)$ except $\delta\text{-time}^{\overline{S}}(a) = t$*
  - *If $\tau(a) = (q, \mathtt{jmp}, \overline{q}) \in E$, then $a \in \overline{A}$ and $S(a) \xrightarrow{(q,\overline{q})} \overline{S}(a)$*
  - *If $\tau(a) = (q, \mathtt{new}[N], \overline{q}) \in E$, then $a \in \overline{A}$, $S(a) \xrightarrow{(q,\overline{q})} \overline{S}(a)$, and there exists a completely new $\overline{a} \notin A$, $\overline{a} \in \overline{A}$ such that $\overline{S}(\overline{a}) = N(S(a))$*
  - *If $\tau(a) = (q, \mathtt{die}, \overline{q}) \in E$, then $a \notin \overline{A}$ and $S(a) \xrightarrow{(q,\overline{q})} \overline{S}(a)$*
  - *If $\tau(a) = (q, \mathtt{snd}[l][T], \overline{q}) \in E$, then both $a \in \overline{A}$, $S(a) \xrightarrow{(q,\overline{q})} \overline{S}(a)$ and $(l(S(a)), T(S(a))) \in \overline{M}$*
  - *If $\tau(a) = (q, \mathtt{recv}[l][R], \overline{q}) \in E$, then $\mu_c(a) = (c, y) \notin \overline{M}$ with $c = l(S(a))$, $a \in \overline{A}$ and $(S(a), R(y, S(a)) \in \mathtt{jump}_{\tau(a)}$ and $\overline{S}(a) = R(y, S(a))$*

There may be multiple messages to deliver to an entity, and vice-versa. To remove this source of non-determinism, we simply use a combination of lexicographical and temporal ordering to choose a single assignment.

Given a single-entity scheduler $\delta : Q \times \mathbb{R}^n \to \mathbb{R}_{\geq 0} \cup E$ like those of DTHA, we define a DDTHA scheduler Δ for an automaton state (A, S, M) as follows

1. If $\forall_{a \in A}\ \delta(S(a)) \in \mathbb{R}$, then $\Delta(AS) = \min\{\delta(S(a)) : a \in A\}$
2. If $\exists_{a \in A}\ \delta(S(a)) \in E$, then $\Delta(AS) = \tau$, where for each $a \in A$:

$$\tau(a) = \begin{cases} \delta(S(a)) & , \delta\text{-time}^{S}(a) = 0 \\ F & , \delta\text{-time}^{S}(a) > 0 \end{cases}$$

Each scheduler Δ yields a single execution of the system. These valid executions are called traces, and are formalised as follows.

Definition 6 (Trace of a DPCHA). *A* trace *of a DPCHA is a sequence* $\sigma = (AS_0, t_0), (AS_1, t_1), ...,$ *with* AS_i *as in Def. 4,* $t_i \in \mathbb{R}_{\geq 0}$ *such that 1)* $AS_0 = (A_0, S_0, \emptyset)$ *and 2) for each* $i \in \mathbb{N}_{>0}$ *(up to the size of the trace if it is finite):*

1. $AS_{i-1} \xrightarrow{\Delta(AS_{i-1})} AS_i$
2. $t_{i-1} = \begin{cases} \Delta(AS_{i-1}) & , \text{ if } \Delta(AS_{i-1}) \in \mathbb{R}_{\geq 0} \\ 0 & , \text{ if } \Delta(AS_{i-1}) \in (L \to E \cup \{F\}) \end{cases}$

Given the priority of discrete transitions, we make the assumption of *divergence of time*, i.e., we do not consider schedulers whose traces have infinitely many transitions in finite time. This ensures there is no infinite sequence of jumps, i.e. that time actually passes and the system evolves.

To obtain DPCHA, we probabilise the single entity scheduler δ, from which the global scheduler Δ is obtained. In effect, we sample from each entity's distribution sequentially until all entities have decided on their course of action. From this set of actions we construct the global action, and the distribution of the global DPCHA action is derived from the distribution of the entities' actions. This results in *Distributed Probabilistic-Control Hybrid Automata*, allowing us to formally specify the model in Example 1.

4 Quantified Bounded Linear Temporal Logic

The next step towards applying SMC techniques is to define a way to specify properties and to check whether they are satisfied by the execution traces of the system. These properties must deal with the distributed nature of the Grid. For example, we want to be able to aggregate power demand, or how much power is being generated in total.

We start from Bounded Linear Temporal Logic (BLTL), featuring a strong *bounded until* $\mathbf{U}^t$ operator to deal with time. $\phi_1 \mathbf{U}^t \phi_2$ states that ϕ_1 must hold until ϕ_2 holds and ϕ_2 holds before the time bound t. It does not require ϕ_1 to hold when ϕ_2 first holds, but it does require ϕ_2 to hold at some point before t.

It has been proven that that BLTL formulae can be checked with only finite traces as long as the system guarantees divergence of time [15]. Unfortunately, BLTL lacks the capability to express properties about a system with a dynamic number of entities, and existing alternatives are domain-specific or bounded [13]. Each entity contains its variables (e.g., refrigerator temperature), but to refer to those variables we must first get a handle on the entity itself. We do this by allowing for quantification over active entities in the system (i.e. actualist quantification). Similarly, we allow any computable aggregation function to range over the entities and return some aggregate value (e.g., max, $\sum$). This results in Quantified Bounded linear Temporal Logic, whose syntax is defined as follows:

Definition 7 (Syntax of QBLTL). *Formulae of QBLTL are given by the following grammar, with* $* \in \{+, -, \div, \times, \hat{}\}$ *and* $\sim \in \{\leq, \geq, =\}$*:*

$\theta ::= c \mid \theta_1 * \theta_2 \mid \pi_i(e) \mid \mathsf{E}(e) \mid \boldsymbol{ag}[e](\theta)$*, with* $i \in \mathbb{N}, c \in \mathbb{Q}$

$\phi ::= \mathsf{E}(e) \mid \theta_1 \sim \theta_2 \mid \phi_1 \vee \phi_2 \mid \neg\phi_1 \mid \phi_1 \, \boldsymbol{U}^t \phi_2 \mid \exists e.\phi_1$

In the above, e is a variable denoting an entity. $\pi_i(e)$ is the ith variable of entity e. $\mathbf{ag}[e](\theta)$ stands as a template for *any* computable, associative and commutative aggregation function (e.g., $\sum[e](\pi_{\text{temp}}(e))$). We abuse notation to define $\mathsf{E}(e)$ as 1) an indicator function for whether e is active, 2) formula evaluating to true iff e is active. This is useful for filtering out entities in aggregations and specifying properties quantifying over entities that exit the system. For example, $\sum[e](\mathsf{E}(e))$ evaluates to the number of active entities in the automaton.

As usual, we define the other logical operators from Def. 7, e.g., $\phi_1 \wedge \phi_2 \equiv \neg(\neg\phi_1 \vee \neg\phi_2)$, and temporal operators such as $\mathbf{F}^t\phi \equiv \text{true}\,\mathbf{U}^t\phi$ (eventually ϕ holds before t) and $\mathbf{G}^t\phi \equiv \neg\mathbf{F}^t\neg\phi$ (ϕ always holds until t).

The semantics of QBLTL are given with respect to traces and a variable assignment $\eta : Vars(\phi) \to L$ to entity labels (cf. Def. 3), where $Vars(\phi)$ is the set of variables occurring in ϕ. η is used to keep track of which entity variables refer to, as in first order logic.

Let $\sigma = (AS_0, t_0), (AS_1, t_1), ...$ be a trace of a DPCHA. We define that trace σ and assignment η satisfy a formula ϕ by a relation $\sigma, \eta \models \phi$. Let σ^i be the trace suffix of σ starting at position i, e.g., $\sigma^0 = \sigma$ and $\sigma^k = (AS_k, t_k), (AS_{k+1}, t_{k+1}), ...$ Let $[\![\theta]\!]^\eta_{\sigma^k}$ represent the value of interpreting θ under AS_k and assignment η, and $AS_i = (A_i, S_i, M_i)$ for all $i \geq 0$.

Definition 8 (Semantics of QBLTL). *The semantics of QBLTL for a trace* $\sigma^k = (AS_k, t_k), (AS_{k+1}, t_{k+1}), ...$ *are defined by the interpretation of terms:*

- $[\![c]\!]^\eta_{\sigma^k} = c$,
- $[\![\theta_1 * \theta_2]\!]^\eta_{\sigma^k} = [\![\theta_1]\!]^\eta_{\sigma^k} * [\![\theta_2]\!]^\eta_{\sigma^k}$, *interpreting the syntactic operator * by the corresponding semantic operator **,
- $[\![\pi_i(e)]\!]^\eta_{\sigma^k} = x_i$, *where* $S_k(\eta(e)) = (q; x) \in Q \times \mathbb{R}^n$, *and* x_i *is the projection to the* i*th coordinate of* x,
- $[\![\mathsf{E}(e)]\!]^\eta_{\sigma^k} = 1$ *if* $\eta(e) \in A_k$ *and* 0 *otherwise.*
- $[\![\boldsymbol{ag}[e](\theta)]\!]^\eta_{\sigma^k} = \boldsymbol{ag}\left([\![\theta]\!]^{\eta\{e\mapsto l_1\}}_{\sigma^k}, \boldsymbol{ag}\left(..., [\![\theta]\!]^{\eta\{e\mapsto l_n\}}_{\sigma^k}\right)\right)$, *where* $(l_1, l_2, ..., l_n)$ *is some ordering of* A_k *(well-defined since* $\boldsymbol{ag}$ *is associative and commutative),*

and the following relation:

- $\sigma^k, \eta \models \mathsf{E}(e)$ *iff* $\eta(e) \in A_k$
- $\sigma^k, \eta \models \theta_1 \sim \theta_2$ *iff* $[\![\theta_1]\!]^\eta_{\sigma^k} \sim [\![\theta_2]\!]^\eta_{\sigma^k}$, *extending the syntactic comparison operator* $\sim$ *to the corresponding semantic* $\sim$,
- $\sigma^k, \eta \models \phi_1 \vee \phi_2$ *iff* $\sigma^k, \eta \models \phi_1$ *or* $\sigma^k, \eta \models \phi_2$,
- $\sigma^k, \eta \models \neg\phi_1$ *iff* $\sigma^k, \eta \not\models \phi_1$ *or it is false that* $\sigma^k, \eta \models \phi_1$,
- $\sigma^k, \eta \models \phi_1 \, \boldsymbol{U}^t \phi_2$ *iff there exists* $i \in \mathbb{N}$ *such that 1)* $\sum_{l=0}^{i} t_{k+l} \leq t$, *2) for all* j *such that* $0 \leq j < i$, $\sigma^{k+j}, \eta \models \phi_1$ *and 3)* $\sigma^{k+i}, \eta \models \phi_2$,
- $\sigma^k, \eta \models \exists e.\phi_1$ *iff there exists* $l \in A_k$ *such that* $\sigma^k, \eta\{e \mapsto l\} \models \phi_1$

As usual in logic, $\sigma^k, \eta \models \phi$ is only well-defined if η contains an assignment for every free variable of ϕ. In $\exists e$, e is a variable ranging over *currently existing* entities. However, these entities may leave the system in the future, leaving us with a "dangling" variable. We illustrate this next.

Example 2. Consider a model where a consumer entity is created whenever an appliance is turned on, and that disappears when it is turned off. While verifying this model we may want to check that some appliances are always running at high power, e.g., a refrigerator with a consumption minimum of 300 watts. This property can be expressed in the following QBLTL formula $\exists e.\mathbf{G}^{24\cdot 3600}\pi_{\text{consumption}}(e) \geq 300$.

Given a trace for a sample day, we attempt to evaluate the formula. For instance, suppose e represents a washing machine that is running at first, but finishes its program and leaves the active Grid sometime later. What is the meaning of $\pi_{\text{consumption}}(e) \geq 300$ after the washing machine leaves the system?

The actualist semantics that we chose achieve what we believe is a good compromise that avoids semantic pitfalls, in the same vein as [10]. The key is to keep track of past entities' state in S so that the semantics are well-defined even with exiting entities. The main point, however, is that the special predicate/term $\Subset(\cdot)$ can be used to handle entities that have left the system. The property above should have been $\exists e.\mathbf{G}^{24\cdot 3600}\Subset(e) \wedge \pi_{\text{consumption}}(e) \geq 300$, i.e. is there an entity that is permanent and that is always consuming above 300.

We have made sure that our extensions are compatible with earlier SMC approaches so that we can lift the theory of SMC directly to our scenario. First, we guarantee that finite simulations are sufficient for checking whether a QBLTL formula is satisfied, because we cannot run infinite simulations. Due to our setting, this is a straightforward extension of results from [15]. We define a bound $\#(\phi)$ of a QBLTL formula by having $\#(\theta) = 0$ for any term θ. For any other logical connective excluding the until operator (e.g., $\neg\phi_1, \phi_1 \vee \phi_2$), we define the bound as the maximum of the bound of its direct subformulae, e.g., $\#(\phi_1 \vee \phi_2) = \max(\#(\phi_1), \#(\phi_2))$, and $\#(\exists e.\phi) = \#(\phi)$. Finally, $\#(\phi_1 \,\mathbf{U}^t \phi_2) = t + \max(\#(\phi_1), \#(\phi_2))$. We can now show that ϕ is satisfied by two infinite traces as long as the prefixes bounded by $\#(\phi)$ of those traces are the same.

Lemma 1 (QBLTL has bounded simulation traces). *Let ϕ be a QBLTL formula and $k \in \mathbb{N}$. Then for any two infinite traces $\sigma = (AS_0, t_0), (AS_1, t_1), \ldots$ and $\overline{\sigma} = (\overline{AS}_0, \overline{t}_0), (\overline{AS}_1, \overline{t}_1), \ldots$ with $AS_{k+I} = \overline{AS}_{k+I}$ and $t_{k+I} = \overline{t}_{k+I}$, for all $I \in \mathbb{N}$ with $\sum_{0 \leq l < I} t_{k+l} \leq \#(\phi)$ we have that $\sigma^k \models \phi$ iff $\overline{\sigma}^k \models \phi$.*

The proof is done by induction on QBLTL formulae. The original proof for BLTL [15] extends directly to our additions. It then follows that sampling can be bounded with $\#(\phi)$.

Lemma 2 (Bounded sampling). *The problem $\sigma \models \phi$ is well defined and can be checked for QBLTL formulae ϕ and traces σ based only on a finite prefix of σ of bounded duration.*

Again, thanks to our compatible setting, the proof for this lemma lifts directly from [15]. Without this result, SMC would not be applicable in our scenario.

5 Bayesian Statistical Model Checking

Statistical Model Checking [4,15,14] is a simple technique that has received attention due to its application to many practical situations. We follow the presentation of a Bayesian approach to the method closely, as presented in [15].

SMC tries to determine the probability p that an arbitrary trace of an automaton satisfies a QBLTL formula ϕ. Two core Bayesian approaches have been proposed: interval estimation and hypothesis testing. These methods diverge from the traditional model checking problem in that a trace that does not satisfy a formula ϕ is not a counter-example, but instead *evidence* that $p < 1$. For simplicity, we present the hypothesis testing algorithm and refer to [15] for an interval estimation algorithm, which is directly applicable in our scenario.

The hypothesis testing algorithm attempts to solve the problem "is the probability that property ϕ holds greater or equal to θ", also represented as $P_{\geq\theta}\phi$. That is, we compare the null hypothesis $H_0 : p \geq \theta$ with the alternate hypothesis $H_1 : p < \theta$. We can represent the result of each sampled trace satisfying ϕ by Bernoulli random variables with the real probability p. After n samples, we have $d = \{x_1, ..., x_n\}$ draws from those Bernoulli RV's, and each result gives us further evidence either for H_0 or for H_1. Since these hypothesis are mutually exclusive, we can assume that the prior probabilities add to 1, $P(H_0)+P(H_1) = 1$. Bayes' theorem gives us the posterior probabilities as $P(H_i|d) = \frac{P(d|H_i)P(H_i)}{P(d)}$ with $i \in \{0, 1\}$, for every d with $P(d) = P(d|H_0)P(H_0) + P(d|H_1)P(H_1) > 0$, which is always the case in this instance.

Definition 9 (Bayes factor). *The Bayes factor $\mathcal{B}$ of sample d and hypotheses H_0, H_1 is $\frac{P(d|H_0)}{P(d|H_1)}$.*

The value of the Bayes factor as defined above, obtained from data d by sampling and testing the property, can be seen as evidence in favour of the acceptance of hypothesis H_0. The inverse $\frac{1}{\mathcal{B}}$, on the other hand, is evidence in favour of H_1. We can then choose a threshold T for how much evidence is required before we accept one of the hypotheses.

From [15], we know an efficient way to calculate the Bayes factor for H_0, H_1:

$$\mathcal{B}_n = \frac{1 - \int_\theta^1 g(u)du}{\int_\theta^1 g(u)du} \left(\frac{1}{F_{(x+\alpha, n-x+\beta)}(\theta)} - 1 \right),$$

in the case of beta priors, where x is the number of successes in the draws $d = (x_1, ..., x_n)$ and $F_{(s,t)}(\cdot)$ is the Beta distribution function with parameters s, t. The actual algorithm can be found in Figure 2.

The algorithm samples traces from the DPCHA, then checks them against the given formula ϕ. Since the result of these checks can be seen as drawing from a Bernoulli RV with the desired probability, the algorithm then uses the Bayes factor to calculate how much evidence is in favour of either H_0 or H_1. The amount of evidence changes with each new draw, resulting in an algorithm that adapts termination to the amount of information it can extract at each

Input: DPCHA automaton A, QBLTL property ϕ, probability θ, threshold $T \geq 1$ and Beta prior density g for unknown parameter p
$n := 0$ {// Total number of traces drawn}
$x := 0$ {// Total number of traces satisfying ϕ}
loop
  $\sigma :=$ sample trace from DPCHA A {// according to probabilistic Δ, cf. Sect. 3}
  $n := n + 1$
  if $\sigma \models \phi$ **then** {// according to Def. 8}
    $x := x + 1$
  end if
  $\mathcal{B} :=$ BayesFactor(n, x)
  if $\mathcal{B} > T$ **then**
    return H_0 accepted
  else if $\mathcal{B} < \frac{1}{T}$ **then**
    return H_1 accepted
  end if
end loop

Fig. 2. Bayesian Statistical Model Checking for estimation

iteration. Eventually, enough evidence is amassed for one of the hypotheses, and it is accepted. More details about this and a more sophisticated *estimation* algorithm (that we use in the following) can be found in [15].

6 Case Study: Smart Grid

We now develop a case study using a simplified Smart Grid model. We show the versatility of our model, how smoothly it fits to the verification methods defined previously, and how easily SMC can be used to check important properties. Recall that the Smart Grid is a fusion of the Power Grid and the Cyber Grid. The hope is that communication capabilities and direct feedback from the consumer level will allow the Smart Grid to provide energy more efficiently and cost-effectively. We use the techniques implemented in our framework to study what properties of the communications layer of the Grid are important for achieving this goal. We focus on the trade-offs between cost-relevant parameters of the network and overall system performance and safety.

As in the examples above, consumer entities represent *classes* of appliances. Their demand follows a bell-shaped curve over time, representing a number of individual appliances being gradually turned on, then off and exiting the system. Consumers are managed by a Consumer Controller, which is the environment's probabilistic core. It spawns and maintains consumer entities, ensuring Grid consumption follows the patterns we observe in real life. The probability of creating a consumer (and its characteristics) depends on the hour of the day. Consumers appearing during the night or late evening request less energy but last for longer (2-3h vs 7-8h). The Power Controller (PC) receives feedback about consumption and matches generator output to demand. The generator only changes its output acceleration, so timing is essential. Refer to [7] for details.

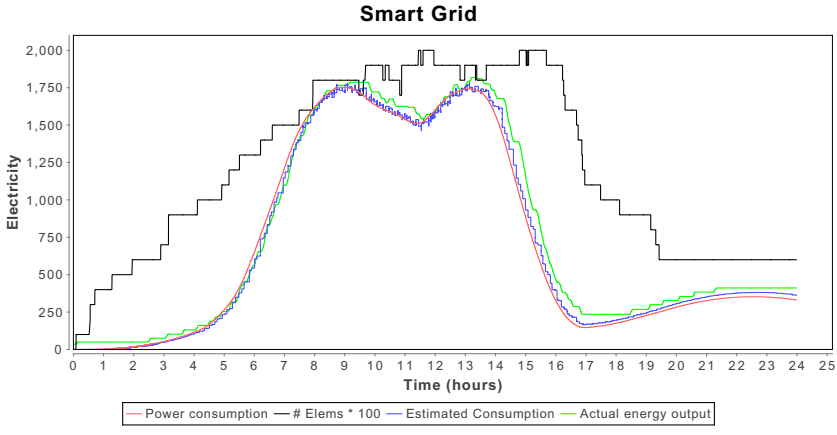

Fig. 3. Smart Grid scenario with one generator

Figure 3 shows aggregate power consumption, generator output, the PC's estimated consumption and the number of active consumer entities during one day. The shape of sample curves matches the patterns observed in reality, with peak times and a break for lunch. This indicates that our model, even simplified, simulates reasonable Grid behaviour. The intervals between control decisions for the consumers, generator and PC are given by Normal random variables with mean of 5 and variance between 1 and 3. During these control decisions, the entity decides whether not to jump along a `recv` edge, emulating message loss.

We wish to investigate what the impact of network reliability on the system level properties is by checking how resilient the Grid is to message loss. We use a benchmark of two core properties. Property (1) $\mathbf{G}^{1440}|\sum[e](Gen(e)\cdot\pi_{\text{output}}(e))-\sum[e](Cons(e)\cdot\pi_{\text{consumption}}(e))|<400$ states that the output of the generator is always within 400 units of energy of the actual demand within the horizon of observation (1440 time units). Property (2) $\mathbf{G}^{1440}|\sum[e](Gen(e)\cdot\pi_{\text{output}}(e))-\sum[e](PC(e)\cdot(\pi_0(e)+\ldots+\pi_{19}(e)))|<250$ states that the PC's estimate of power consumption is not too far from the truth. The PC's variables 0 through 19 store how much the consumers tell the PC they are consuming. Here, *PC*, *Gen* and *Cons* are simply indicator functions for whether the element is the power controller, a generator or a consumer. We would expect that property (2) is a prerequisite to property (1), because regulating generator output depends on having good estimates of the demand.

In our experiments to test message loss resilience, we vary the delivery probability of messages for the PC. In other words, whenever there is a control decision the probability that the PC will receive a message (indicating there was no message loss) can be 0.9, 9.95, 0.97, 0.98, 0.99 and 1.00. To test these properties we use Bayesian interval estimation [15], which is a variation of the algorithm in Section 5. This algorithm returns a confidence interval where the probability that the properties are satisfied lie. We can specify the size of the interval, as well as the confidence coefficient, allowing it to be used for cursory and in-depth

Table 1. Experimental results for Bayesian hypothesis testing for Smart Grid

	(1) 1.00	(2) 1.00	(1) 0.99	(2) 0.99	(1) 0.98	(2) 0.98
Prob.	[0.89, 0.93]	[0.95, 0.99]	[0.87, 0.91]	[0.91, 0.95]	[0.86, 0.90]	[0.86, 0.90]
# correct/total	508/557	180/183	582/651	399/426	634/720	608/685
	(1) 0.97	(2) 0.97	(1) 0.95	(2) 0.95	(1) 0.9	(2) 0.9
Prob.	[0.83, 0.86]	[0.82, 0.86]	[0.75, 0.79]	[0.66, 0.70]	[0.28, 0.32]	[0.16, 0.20]
# correct/total	745/879	754/893	914/1180	998/1461	431/1423	169/971

analyses. Table 1 summarises the results for intervals of 0.04 and a confidence coefficient of 0.95.

As one would expect, a higher probability of message delivery errors will exponentially decrease the probability that the Grid is "safe" by making the generator output deviate too far from what the actual consumption is. In this scenario, we could now focus on the message delivery probability interval between 0.97 and 1.00. This helps companies and utilities decide whether to invest in more reliable communication infrastructures or not, depending on what they perceive the risk to be. It is unclear whether higher levels justify investment in 0.99 or 0.995 reliable infrastructures, because they are more expensive at Grid scale.

We also see that the stronger property (1) holds less often than the weaker (2), as we foresaw. Furthermore, the discrepancy is proportional to the error rate. This tells us that communication is central in the Smart Grid. Property (1), by requiring communication from consumers to the PC to the generator, is clearly affected by compounded delays of two hops, while (2) only requires one.

Network bandwidth is another very configurable network parameter that greatly affects deployment costs. The Grid industry still deploys networks that send a few thousand bits *per day*. Using the above model with 0.98 message delivery probability but *doubling* the consumer feedback interval from 5 to 10 minutes, we obtain the following intervals: for property (1), [0.80, 0.84] and for (2), [0.78, 0.82], a much lower performance decrease than we expected. We omit a similar analysis to the one above due to space constraints, and refer to [7].

7 Conclusions

In order to check for desirable properties of Smart Grid technologies, we defined Distributed Probabilistic-Control Hybrid Automata as a model for hybrid systems with a dynamic number of probabilistic elements, and Quantified Bounded Linear Temporal Logic to specify properties in the distributed scenario. We also showed that Bayesian statistical model checking techniques are applicable in this context for verifying QBLTL properties. Finally, we developed a Smart Grid case study where even a preliminary study revealed important cost-benefit relations relevant to full-scale deployment.

References

1. Alur, R., Courcoubetis, C., Henzinger, T.A., Ho, P.H.: Hybrid automata: An algorithmic approach to the specification and verification of hybrid systems. In: Grossman, R.L., Ravn, A.P., Rischel, H., Nerode, A. (eds.) HS 1991 and HS 1992. LNCS, vol. 736, Springer, Heidelberg (1993)
2. Demongodin, I., Koussoulas, N.: Differential Petri nets: representing continuous systems in a discrete-event world. IEEE Transactions on Automatic Control 43(4), 573–579 (1998)
3. Henzinger, T.A.: The theory of hybrid automata. In: LICS (1996)
4. Legay, A., Delahaye, B., Bensalem, S.: Statistical model checking: An overview. In: Barringer, H., Falcone, Y., Finkbeiner, B., Havelund, K., Lee, I., Pace, G., Roşu, G., Sokolsky, O., Tillmann, N. (eds.) RV 2010. LNCS, vol. 6418, pp. 122–135. Springer, Heidelberg (2010)
5. Lynch, N.A.: Input/Output automata: Basic, timed, hybrid, probabilistic, dynamic,.. In: Amadio, R.M., Lugiez, D. (eds.) CONCUR 2003. LNCS, vol. 2761, pp. 187–188. Springer, Heidelberg (2003)
6. Lynch, N.A., Segala, R., Vaandrager, F.W., Weinberg, H.B.: Hybrid I/O automata. In: Alur, R., Sontag, E.D., Henzinger, T.A. (eds.) HS 1995. LNCS, vol. 1066, Springer, Heidelberg (1996)
7. Martins, J., Platzer, A., Leite, J.: Statistical model checking for distributed probabilistic-control hybrid automata in the smart grid. Tech. Rep. CMU-CS-11-119, Computer Science Department, Carnegie Mellon University (2011)
8. Meseguer, J., Sharykin, R.: Specification and analysis of distributed object-based stochastic hybrid systems. In: Hespanha, J.P., Tiwari, A. (eds.) HSCC 2006. LNCS, vol. 3927, pp. 460–475. Springer, Heidelberg (2006)
9. Platzer, A.: Differential dynamic logic for hybrid systems. J. Autom. Reas. 41(2), 143–189 (2008)
10. Platzer, A.: Quantified differential dynamic logic for distributed hybrid systems. In: Dawar, A., Veith, H. (eds.) CSL 2010. LNCS, vol. 6247, pp. 469–483. Springer, Heidelberg (2010)
11. Platzer, A.: Stochastic differential dynamic logic for stochastic hybrid programs. In: Bjørner, N., Sofronie-Stokkermans, V. (eds.) CADE 2011. LNCS, vol. 6803, pp. 431–445. Springer, Heidelberg (2011)
12. Trivedi, K.S., Kulkarni, V.G.: FSPNs: Fluid stochastic Petri nets. In: Ajmone Marsan, M. (ed.) ICATPN 1993. LNCS, vol. 691, pp. 24–31. Springer, Heidelberg (1993)
13. Yahav, E., Reps, T., Sagiv, M.: LTL model checking for systems with unbounded number of dynamically created threads and objects. Tech. Rep. TR-1424, Computer Sciences Department, University of Wisconsin (2001)
14. Younes, H.L.S., Simmons, R.G.: Statistical probabilistic model checking with a focus on time-bounded properties. Inf. Comput. 204(9), 1368–1409 (2006)
15. Zuliani, P., Platzer, A., Clarke, E.M.: Bayesian statistical model checking with application to Simulink/Stateflow verification. In: Johansson, K.H., Yi, W. (eds.) HSCC, pp. 243–252. ACM, New York (2010)