HASL: a New Approach for Performance Evaluation and Model Checking from Concepts to Experimentation

Paolo Ballarini[a], Benoît Barbot[b], Marie Duflot[c], Serge Haddad[b], Nihal Pekergin[d]

[a]*Ecole Centrale de Paris, France*
[b]*LSV, ENS Cachan & CNRS & INRIA, France*
[c]*Université de Lorraine- LORIA, Nancy, France*
[d]*LACL, Université Paris-Est Créteil, France*

Abstract

We introduce the Hybrid Automata Stochastic Language (HASL), a new temporal logic formalism for the verification of Discrete Event Stochastic Processes (DESP). HASL employs a Linear Hybrid Automaton (LHA) to select prefixes of relevant execution paths of a DESP. LHA allows rather elaborate information to be collected *on-the-fly* during path selection, providing the user with powerful means to express sophisticated measures. A formula of HASL consists of an LHA and an expression Z referring to moments of *path random variables*. A simulation-based statistical engine is employed to obtain a confidence interval estimate of the expected value of Z. In essence, HASL provides a unifying verification framework where temporal reasoning is naturally blended with elaborate reward-based analysis. Moreover, we have implemented a tool, named COSMOS, for performing analysis of HASL formula for DESP modelled by Petri nets. Using this tool we have developed two detailed case studies: a flexible manufacturing system and a genetic oscillator.

Keywords: Discrete Event Stochastic Process, Statistical Model Checking, Performance Evaluation.

1. Introduction

From model checking to quantitative model checking. Since its introduction [29], model checking has quickly become a prominent technique for verification of discrete-event systems. Its success is mainly due to three factors: (1) the ability to express specific properties by formulas of an appropriate logic, (2) the firm mathematical foundations based on automata theory and (3) the simplicity of the verification algorithms which has led to the development of numerous tools. While the study of systems requires both functional, performance and dependability analysis, originally the techniques associated with these kinds of analysis were different. However, in the mid nineties, classical temporal logics were adapted to express properties of Markov chains and verification procedures have been designed based on transient analysis of Markov chains [9].

From numerical model checking to statistical model checking. The numerical techniques for quantitative model checking are rather efficient when a memoryless property

can be exhibited (or recovered by a finite-state memory), limiting the combinatory explosion due to the necessity to keep track of the history. Unfortunately both the formula associated with an elaborated property and the stochastic process associated with a real application make rare the possibility of such a pattern. In these cases, statistical model checking [48] is an alternative to numerical techniques. Roughly speaking, statistical model checking consists in sampling executions of the system (possibly synchronised with some automaton corresponding to the formula to be checked) and comparing the ratio of successful executions with a threshold specified by the formula. The advantage of statistical model checking is the small memory requirement while its drawback is its inability to generate samples for execution paths of potentially unbounded length.

Limitations of existing logics. A topic that has not been investigated is the suitability of the temporal logic to express (non necessarily boolean) quantities defined by path operators (minimum, integration, etc.) applied on instantaneous indicators. Such quantities naturally occur in standard performance evaluation. For instance, the average length of a waiting queue during a busy period or the mean waiting time of a client are typical measures that cannot be expressed by the quantitative logics based on the concept of successful execution probability like CSL [7].

Our contribution. Following the idea to relieve these limitations, we introduce a new language called Hybrid Automaton Stochastic Language (HASL) which provides a unified framework both for model checking and for performance and dependability evaluation [1]. Evaluating a system using HASL allows both the probability computation of a complex set of paths as well as elaborate performability measures. The use of conditional expectation over these subsets of paths significantly enlarges the expressive power of the language.

A formula of HASL consists of an automaton and an expression. The automaton is a Linear Hybrid Automaton (LHA), i.e. an automaton with clocks, called in this context *data variables*, where the dynamic of each variable (i.e. the variable's evolution) depends on the model states. This automaton is synchronised with the model of the system, precisely selecting accepting paths while maintaining detailed information on the path through data variables. The expression is based on moments of path random variables associated to path executions. These variables are obtained by operators like time integration on data variables.

HASL extends the expressiveness of automaton-based CSL as CSL^{TA} [28] and its extension to multi-clocks [22] with state and action rewards and sophisticated update functions especially useful for performance and dependability evaluation. On the other hand it extends reward enriched versions of CSL, (CSRL [10]) with a more elaborate selection of path executions, and the possibility to consider multiple rewards. Therefore HASL makes possible to consider not only standard performability measures but also complex ones in a generic manner.

A statistical verification tool, named COSMOS, has been developed for this language. We have chosen generalised stochastic Petri nets (GSPN) as high level formalism for the description of the discrete event stochastic process since (1) it allows a flexible modelling w.r.t. the policies defining the process (choice, service and memory)

[1] A preliminary version of this language has been presented in [12].

and (2) due to the locality of net transitions and the simplicity of the firing rule it leads to efficient path generation. This tool has been presented in [11].

We have developed in this paper two detailed case studies relative to flexible manufacturing systems (FMS) and to gene expression. The choice of FMS was guided by the fact that they raise interesting analysis issues: productivity, flexibility, fault tolerance, etc. While analysis of such systems is usually based on standard performance indices, we demonstrate the usefulness of HASL through elaborate transient formulas. Similarly, biological measures often concern the shape of trajectories which can only be expressed by complex formulas. In addition, our experiments demonstrate the time efficiency of the COSMOS tool.

Organisation. In section 2, we describe the class of stochastic models we refer to (i.e. DESP). In section 3, we formally introduce the HASL language and we provide an overview of the related work, where the expressiveness of HASL is compared with that of existing logics. Section 4 is dedicated to the presentation of the basic principles of statistical model checking as well as the COSMOS software tool for HASL verification. In section 5, we present two full case studies: a flexible manufacturing system and a genetic oscillator. Finally, in section 6, we conclude and give some perspectives.

2. Discrete Event Stochastic Process

We describe in this section a large class of stochastic models that are suitable for HASL verification, namely Discrete Event Stochastic Processes (DESP). Such a class includes in particular the main type of stochastic models targeted by existing stochastic logics, namely Markov chains. The definition of DESP we introduce is similar to that of generalised semi-Markov processes [30] as well as that given in [2].

Syntax. DESPs are stochastic processes consisting of a (possibly infinite) set of states and whose dynamic is triggered by a set of discrete events. We do not consider any restriction on the nature of the distribution associated with events. In the sequel $dist(A)$ denotes the set of distributions whose support is A.

Definition 1. *A DESP is a tuple*
$\mathcal{D} = \langle S, \pi_0, E, Ind, enabled, delay, choice, target \rangle$ *where*

- S *is a (possibly infinite) set of states,*
- $\pi_0 \in dist(S)$ *is the initial distribution on states,*
- E *is a finite set of events,*
- Ind *is a set of functions from* S *to* $\mathbb{R}$ *called state indicators (including the constant functions),*
- $enabled : S \to 2^E$ *are the enabled events in each state with for all* $s \in S$, $enabled(s) \neq \emptyset$.
- $delay \colon S \times E \to dist(\mathbb{R}^+)$ *is a partial function defined for pairs* (s, e) *such that* $s \in S$ *and* $e \in enabled(s)$.

- *choice* : $S \times 2^E \times \mathbb{R}^+ \rightarrow dist(E)$ *is a partial function defined for tuples* (s, E', d) *such that* $E' \subseteq enabled(s)$ *and such that the possible outcomes of the corresponding distribution are restricted to* $e \in E'$.
- $target$: $S \times E \times \mathbb{R}^+ \rightarrow S$ *is a partial function describing state changes through events defined for tuples* (s, e, d) *such that* $e \in enabled(s)$.

From syntax to semantics. Given a state s, $enabled(s)$ is the set of events enabled in s. For an event $e \in enabled(s)$, $delay(s, e)$ is the distribution of the delay between the enabling of e and its possible occurrence. Furthermore, if we denote d the earliest delay in some configuration of the process with state s, and $E' \subseteq enabled(s)$ the set of events with minimal delay, $choice(s, E', d)$ describes how the conflict is randomly resolved: for all $e' \in E'$, $choice(s, E', d)(e')$ is the probability that e' will be selected among E' after waiting for the delay d. The function $target(s, e, d)$ denotes the target state reached from s on occurrence of e after waiting for d time units.

Let us detail these explanations. A *configuration* of a DESP is described as a triple $(s\ ,\ \tau\ ,\ sched)$ with s being the current state, $\tau \in \mathbb{R}^+$ the current time and $sched$: $E \rightarrow \mathbb{R}^+ \cup \{+\infty\}$ being the function that describes the occurrence time of each enabled event ($+\infty$ if the event is not enabled). Starting from a given configuration $(s\ ,\ \tau\ ,\ sched)$ of a DESP, we can now informally define the dynamics of a DESP. It is an infinite loop, where each iteration consists in the following steps. First the function $sched$ provides E', the set of enabled events with minimal delay, i.e. $E' = \{e \in enabled(s) \mid \forall e' \in enabled(s),\ sched(e) \leq sched(e')\}$. The corresponding delay d is equal to $sched(s) - \tau$ for any $s \in E'$. Then the probability distribution $choice(s, E', d)$ randomly specifies which event $e \in E'$ will be sampled. The next state s' is then defined by $target(s, e, d)$. Finally the function $sched$ is updated as follows. First for all events e' different from e that are still enabled, $sched(e')$ is maintained while for all other enabled event e' in state s', a new delay d' is sampled according to the distribution $delay(s', e')$ and $sched(e')$ is set to $\tau + d + d'$. When e' is disabled, $sched(e')$ is set to $+\infty$. The three elements of a configuration being updated, the system can now start a new iteration. Let us now give a more formal definition of $sched$ and the families of random variables corresponding to the successive states, events and time instants of a trace of the DESP.

Operational semantics of a DESP. Given a discrete event system, its execution is characterised by a (possibly infinite) sequence of events $\{e_1, e_2, \ldots\}$ and occurrence time of these events. Only the events can change the state of the system. In the stochastic framework, the behaviour of a DESP is defined by three families of random variables:

- $e_1, \ldots, e_n, \ldots$ defined over the set of events E denoting the sequence of events occurring in the system;
- $s_0, \ldots, s_n, \ldots$ defined over the (discrete) state space of the system, denoted as S. s_0 is the system initial state and s_n for $n > 0$ is the state reached after the n^{th} event. The occurrence of an event does not necessarily modify the state of the system, and therefore s_{n+1} may be equal to s_n;

- $\tau_0 \leq \tau_1 \leq \cdots \leq \tau_n \leq \cdots$ defined over $\mathbb{R}^+$, where τ_0 is the initial instant and τ_n for $n > 0$ is the instant of the occurrence of the n^{th} event.

We start from the syntactical definition of a DESP and show how we obtain the three families of random variables $\{s_n\}_{n\in\mathbb{N}}$, $\{e_n\}_{n\in\mathbb{N}^*}$ and $\{\tau_n\}_{n\in\mathbb{N}}$. This definition is inductive w.r.t. n and includes some auxiliary families.

Notation. In the whole section, when we write an expression like $Pr(e_{n+1} = e \mid e \in E'_n)$, we also mean that this conditional probability is independent from any random event Ev that could be defined using the previously defined variables:
$Pr(e_{n+1} = e \mid e \in E'_n) = Pr(e_{n+1} = e \mid e \in E'_n \wedge Ev)$

The family $\{sched(e)_n\}_{n\in\mathbb{N}}$ whose range is $\mathbb{R}^+ \cup \{+\infty\}$ denotes the current schedule for event e in state s_n (with value ∞ if e is not scheduled). Now τ_{n+1} is defined by $\tau_{n+1} = \min(sched(e)_n \mid e \in E)$ and the family $\{E'_n\}_{n\in\mathbb{N}}$ denotes the set of events with minimal schedule: $E'_n = \{e \in E \mid \forall e' \in E, sched(e)_n \leq sched(e')_n\}$.
From this family we obtain the conditional distribution of e_{n+1}:
$Pr(e_{n+1} = e \mid e \in E'_n \wedge \tau_{n+1} - \tau_n = d) = choice(s_n, E'_n, d)(e)$.

Now $s_{n+1} = target(s_n, e_{n+1}, \tau_{n+1} - \tau_n)$ and:
• For every $e \in E$, $Pr(sched(e)_{n+1} = \infty \mid e \notin enabled(s_{n+1})) = 1$
• For every $e \in E$,
$Pr(sched(e)_{n+1} = sched(e)_n \mid e \in enabled(s_{n+1}) \cap enabled(s_n) \wedge e \neq e_{n+1}) = 1$
• For every $e \in E$ and $d \in \mathbb{R}^+$,
$Pr(\tau_{n+1} \leq sched(e)_{n+1} \leq \tau_{n+1} + d \mid e \in enabled(s_{n+1}) \wedge (e \notin enabled(s_n) \vee e = e_{n+1})) = delay(s_{n+1}, e)(d)$. Notice that $delay$ is defined by its cumulative distribution function.

We start the induction by:
• $Pr(\tau_0 = 0) = 1$, $Pr(s_0 = s) = \pi_0(s)$
• For every $e \in E$, $Pr(sched(e)_0 = \infty \mid e \notin enabled(s_0)) = 1$
• For every $e \in E$ and $d \in \mathbb{R}^+$, $Pr(sched(e)_0 \leq d \mid e \in enabled(s_0)) = delay(s_0, e)(d)$.

We define the subset $Prop \subseteq Ind$ of *state propositions* taking values in $\{0, 1\}$. The sets Ind and $Prop$ will be used in the sequel to characterise the information on the DESP known by the automaton (LHA) corresponding to a formula. In fact the LHA does not have direct access to the current state of the DESP but only through the values of the state indicators and state propositions.

Note that by definition the evolution of a DESP is naturally suitable for discrete event simulation. However, while it can model almost all interesting stochastic processes, it is a low level representation since the set of states is explicitly described. A solution for a higher level modelling is to choose one of the formalisms commonly used for representing Markov chains (e.g. Stochastic Petri Nets [1] or Stochastic Process Algebras [31]), that can straightforwardly be adapted for representation of a large class of DESPs. It suffices that the original formalisms are provided with formal means to represent the type of delay distribution of each transition/action (function $delay$ of Definition 1) as well as means to encode the probabilistic choice between concurrent events (i.e. function $choice$ of Definition 1).

Our approach is based on Generalised Stochastic Petri Net (GSPN), a formalism particularly well suited for concurrent and distributed systems which yields to highly efficient verification of properties. Below we informally outline the basis of GSPN

specification (for a formal account we refer the reader to [1]) pointing out the differences between "original" GSPNs and our variant.

GSPN models. A GSPN model is a bipartite graph consisting of two classes of nodes: *places* and *transitions*. Places may contain *tokens* (representing the state of the modelled system) while transitions indicate how tokens "flow" within the net (encoding the model dynamics). The state of a GSPN consists of a *marking* indicating the distribution of tokens among the places (i.e. how many tokens each place contains). Roughly speaking a transition t is enabled whenever every *input place* of t contains a number of tokens greater than or equal to the multiplicity of the corresponding (input) arc. An enabled transition may *fire*, consuming tokens (in a number indicated by the multiplicity of the corresponding input arcs) from its input places, and producing tokens (in a number indicated by the multiplicity of the corresponding output arcs) in its *output places*. Transitions can be either *timed* (denoted by empty bars) or *immediate* (denoted by filled-in bars, see Figure 1). Transitions are characterised by: (1) a *distribution* which randomly determines the delay before firing it; (2) a *priority* which deterministically selects among the transitions scheduled the soonest, the one to be fired; (3) a *weight*, that is used in the random choice between transitions scheduled the soonest with the same highest priority. With the original GSPN formalism [1] the delay of timed transitions is assumed *exponentially* distributed, whereas with our GSPN it can be given by any distribution. Thus a GSPN timed-transition is characterised by a tuple: $t \equiv (type, par, pri, w)$, where $type$ indicates the type of distribution (e.g. uniform), par indicates the parameters of the distribution (e.g $[\alpha, \beta]$), $pri \in \mathbb{R}^+$ is a priority assigned to the transition and $w \in \mathbb{R}^+$ is used to probabilistically choose between transitions occurring with equal delay and equal priority. Observe that the information associated with a transition (i.e. $type, par, pri, w$) is exploited in different manners depending on the type of transition. For example for a transition with a continuous distribution the priority (pri) and weight (w) records are superfluous (hence ignored) since the probability that the schedule of the corresponding event is equal to the schedule of the event corresponding to another transition is null. Similarly, for an immediate transition (denoted by a filled-in bar) the specification of the distribution type (i.e. $type$) and associated parameters (par) is irrelevant (hence also ignored). Therefore these unnecessary informations are omitted in Figure 1.

Running example. This model will be used in Section 3 for describing, through a couple of LHA examples, the intuition behind hybrid automata based verification. We consider the GSPN model of Figure 1 (inspired by [1]). It describes the behaviour of an open system where two classes of clients (processes) (namely 1 and 2) compete to access a shared resource (memory). Class i-clients ($i \in \{1, 2\}$) enter the system according to a Poisson process with parameter λ_i (corresponding to the exponentially distributed timed transition Arr_i with rate λ_i). On arrival, clients cumulate in places Req_i waiting for the resource to be free (a token in place $Free$ witnessing that the resource is available). The exclusive access to the shared resource is regulated either deterministically or probabilistically by the *priority* (pri_i) and the *weight* (w_i) of immediate transitions $Start_1$ and $Start_2$. Thus in presence of a competition (i.e. one or more tokens in both Req_1 and Req_2) a class i client wins the competition with a class

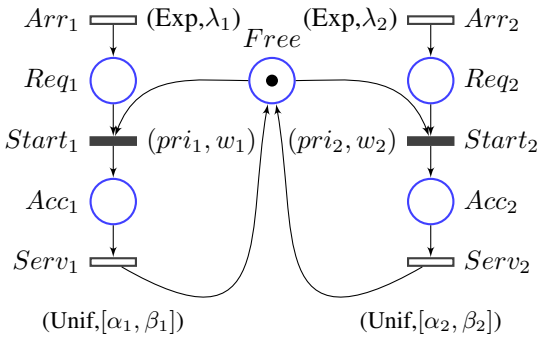


Figure 1: The GSPN description of a shared resource system.

$j \neq i$ client with probability 1 if $pri_i > pri_j$, and with probability $w_i/(w_i + w_j)$ if $pri_i = pri_j$. The occupation time of the resource by a class i client is assumed to be uniformly distributed within the interval $[\alpha_i, \beta_i]$ (corresponding to transitions $Serv_i$). Thus on firing transition $Serv_i$ the resource is released and a class i client leaves the system.

3. HASL

The use of statistical methods instead of numerical ones gives us the possibility to relieve the limitations that were inherent to numerical methods, in terms of model and properties. When numerical model checking was focusing on Markovian models, statistical methods permit to use a very wide range of distributions, and to *synchronise* such a model with an automaton that includes linearly evolving variables, complex updates and constraints. We are no more limited to the probability with which a property is satisfied, we can also compute the expected value of performability parameters such as waiting time, number of clients in a system, production cost of an item. In this section, we present the Hybrid Automata Stochastic Language, first introduced in [12], and illustrate its expressiveness on examples. We intuitively describe the syntax and semantics of HASL before formally defining them in the next subsections. A formula of HASL consists of two parts:

- The first component of a formula is a hybrid automaton that synchronises with an infinite timed execution of the considered DESP until some final location is reached (i.e. the execution is successful) or the synchronisation fails. This automaton uses data variables evolving along the path. They both enable to select the subset of successful executions and maintain detailed information on the path.

- The second component of a formula is an expression based on the data variables that expresses the quantity to be evaluated. In order to express path indices, they

include path operators such as min and max values along an execution, value at the end of a path, integral over time and the time average value operator. Conditional expectations over the successful paths are applied to these indices in order to obtain the value of the formula[2].

In order to illustrate our formalism, we consider the automaton of Figure 2. Its purpose is to compute the lower and upper bounds on the waiting time for clients of type 1 in the DESP of Figure 1. It has three data variables, whose evolution rate is indicated in location l_0. Variable x_2 (resp. x_3) is a counter to record the number of completed (resp. started) accesses of type 1 clients to the shared resource. Variable x_1 counts the cumulated waiting time for all clients of type 1, thus the rate of x_1 in location l_0 is equal to the number of type 1 clients waiting for the resource (Req_1). The transitions can either be synchronised and triggered by the actions of the DESP (the self loops) where $Serv_1$ (resp. $Start_1$) denotes the end (resp. beginning) of type 1 service, or autonomous (denoted by $\sharp$) and triggered by the evolution of the variables' values (transition to the final location l_1).

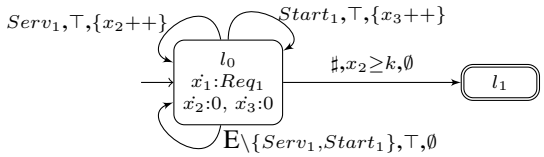


Figure 2: An LHA to compute the bounds on the waiting time

An execution of this automaton terminates in final location l_1 after k type 1 clients have completed their access to the shared resource. The formulas used to compute the aforementioned bounds are presented in page 15. Let us explain the behaviour of the synchronised product of the LHA and the DESP, based on the following execution with $k = 1$. Initially in the DESP two transitions (Arr_1 and Arr_2) are enabled and so (randomly) scheduled respectively at times 3 and 17). The LHA starts in location l_0 with all variables initialised to 0. Since the rate of x_2 in l_0 is null, the autonomous transition labelled by $\sharp$ and guarded by $x_2 \geq 1$ cannot be eventually fired. So the initial state of this execution is: $((Free, 0, (Arr_1{:}3, Arr_2{:}17)), (l_0, (0, 0, 0))$ where the first component of this couple is the marking of the GSPN, the current time, and the time schedules of its enabled transitions while the second component is the location of the LHA and the values of its data variables. Then transition Arr_1 is fired synchronising with the loop below l_0. $Start_1$ is now enabled and is scheduled immediately (due to its Dirac distribution). In addition, Arr_1 is again enabled and scheduled (at time 12). This leads to: $((Req_1 + Free, 3, (Arr_1{:}12, Arr_2{:}17, Start_1{:}3)), (l_0, (0, 0, 0)))$. The firing of $Start_1$ is synchronised with the upper right loop incrementing x_3. This

[2]As seen later on, the existence of expectations requires syntactical or semantical restrictions on the formula and/or on the DESP.

leads to: $((Acc_1, 3, (Arr_1{:}12, Arr_2{:}17, Serv_1{:}15)), (l_0, (0, 0, 1)))$. A new client of type 1 arrives at time 12, with the scheduling of a new client at time 20: $(Req_1 + Acc_1, 12, (Arr_1 : 20, Arr_2 : 17, Serv_1 : 15)), (l_0, (0, 0, 1)))$. When the first client is served at time 15, this leads to a synchronisation with the upper left loop and x_2 is incremented. In addition the rate of x_1 is 1, and so at the time of the firing x_1 is incremented by 3: $(Req_1 + Free, 15, (Arr_1{:}20, Arr_2{:}17, Start_1{:}15, \sharp{:}15)), (l_0, (3, 1, 1)))$. Observe that the autonomous transition and a synchronised one are scheduled at the same time (15). Since an autonomous transition has higher priority, it is fired. The resulting location is a final one and so the reached state is an absorbing state of the product: $(Req_1 + Free, 15, (l_1, (3, 1, 1)))$.

We will now proceed to a more formal definition of the automata and expressions used in HASL.

3.1. Synchronised Linear Hybrid Automata

Syntax. The first component of a HASL formula is a restriction of a hybrid automaton [3], namely a synchronised Linear Hybrid Automaton (LHA). Such automata extend the Deterministic Timed Automata (DTA) used to describe properties of Markov chain models [28, 22]. Simply speaking, LHA are automata whose set of *locations* is associated with a n-tuple of real-valued variables (called data variables) whose rate can vary.

In our context, the LHA is used to synchronise with DESP paths. However, it can evolve in an autonomous way. The symbol $\sharp$, associated with these autonomous changes, is thus used to denote a pseudo-event that is not included in the event set E of the DESP. The transitions in the synchronised system (DESP + LHA) are either autonomous, *i.e.* time-triggered (or rather variable-triggered) and take place as soon as a constraint is satisfied, or synchronised *i.e.* triggered by the DESP and take place when an event occurs in the DESP. The LHA will thus take into account the system behaviour through synchronised transitions, but also take its own autonomous transitions in order to evaluate the desired property.

The values of the data variables $x_1, \ldots, x_n$ evolve with a linear rate depending both on the location of the automaton and on the current state of the DESP. More precisely, the function *flow* associates with each location of the automaton an n-tuple of indicators (one for each variable) and, given a state s of a DESP and a location l, the flow of variable x_i in (s, l) is $\mathit{flow}_i(l)(s)$ (where $\mathit{flow}_i(l)$ is the i^{th} component of $\mathit{flow}(l)$). Our model also uses *constraints*, which describe the conditions for an edge to be traversed, and *updates*, which describe the actions taken on the data variables on traversing an edge. A *constraint* of an LHA edge is a boolean combination of inequalities of the form $\sum_{1 \leq i \leq n} \alpha_i x_i + c \prec 0$ where $\alpha_i, c \in Ind$ are indicators, and $\prec \in \{=, <, >, \leq, \geq\}$. The set of constraints is denoted by Const. Given a location and a state, an expression of the form $\sum_{1 \leq i \leq n} \alpha_i x_i + c$ linearly evolves with time. An inequality thus gives an interval of time during which the constraint is satisfied. We say that a constraint is left closed if, whatever the current state s of the DESP (defining the values of indicators), the time at which the constraint is satisfied is a union of left closed intervals. These special constraints are used for the "autonomous" edges, to ensure that the first time instant at which they are satisfied exists. We denote by lConst the set of left closed constraints.

An *update* is more general than the reset of timed automata. Here each data variable can be set to a linear function of the variables' values. An update U is then an n-tuple of functions $u_1, ..., u_n$ where each u_k is of the form $x_k = \sum_{1 \leq i \leq n} \alpha_i x_i + c$ where the α_i and c are indicators. The set of updates is denoted by Up.

Definition 2. *A* synchronised linear hybrid automaton *(LHA) is defined by a tuple* $\mathcal{A} = \langle E, L, \Lambda, Init, Final, X, flow, \rightarrow \rangle$ *where:*

- E *is a finite alphabet of events;*
- L *is a finite set of locations;*
- $\Lambda : L \rightarrow Prop$ *is a location labelling function;*
- $Init$ *is a subset of* L *called the initial locations;*
- $Final$ *is a subset of* L *called the final locations;*
- $X = (x_1, ...x_n)$ *is a* n*-tuple of data variables;*
- $flow : L \mapsto Ind^n$ *is a function which associates with each location one indicator per data variable representing the evolution rate of the variable in this location.* $flow_i$ *denotes the projection of flow on its* i^{th} *component.*
- *The transition relation* $\rightarrow \subseteq L \times \big((2^E \times \mathsf{Const}) \uplus (\{\sharp\} \times \mathsf{IConst})\big) \times \mathsf{Up} \times L$ *is a set of edges, where the notation* $l \xrightarrow{E', \gamma, U} l'$ *means that* $(l, E', \gamma, U, l') \in \rightarrow$ *and* $\uplus$ *denotes the disjoint union.*

Furthermore $\mathcal{A}$ *fulfils the following conditions.*

- **Initial determinism:** $\forall l \neq l' \in Init$, $\Lambda(l) \wedge \Lambda(l') \Leftrightarrow \texttt{false}$. *This must hold whatever the interpretation of the indicators occurring in* $\Lambda(l)$ *and* $\Lambda(l')$.
- **Determinism on events:** $\forall E_1, E_2 \subseteq E \, s.t. \, E_1 \cap E_2 \neq \emptyset$, $\forall l, l', l'' \in L$, *if* $l'' \xrightarrow{E_1, \gamma, U} l$ *and* $l'' \xrightarrow{E_2, \gamma', U'} l'$ *are two distinct transitions, then either* $\Lambda(l) \wedge \Lambda(l') \Leftrightarrow \texttt{false}$ *or* $\gamma \wedge \gamma' \Leftrightarrow \texttt{false}$. *Again this equivalence must hold whatever the interpretation of the indicators occurring in* $\Lambda(l)$, $\Lambda(l')$, γ *and* γ'.
- **Determinism on** $\sharp$**:** $\forall l, l', l'' \in L$, *if* $l'' \xrightarrow{\sharp, \gamma, U} l$ *and* $l'' \xrightarrow{\sharp, \gamma', U'} l'$ *are two distinct transitions, then either* $\Lambda(l) \wedge \Lambda(l') \Leftrightarrow \texttt{false}$ *or* $\gamma \wedge \gamma' \Leftrightarrow \texttt{false}$.
- **No** $\sharp$**-labelled loops:** *For all sequences* $l_0 \xrightarrow{E_0, \gamma_0, U_0} l_1 \xrightarrow{E_1, \gamma_1, U_1} \cdots \xrightarrow{E_{n-1}, \gamma_{n-1}, U_{n-1}} l_n$ *such that* $l_0 = l_n$, *there exists* $i \leq n$ *such that* $E_i \neq \sharp$.

Illustration. In order to make this definition more concrete, we get back to the automaton of Figure 2. On this automaton the set of events E is Arr_i, $Start_i$ and $Serv_i$ with $i \in \{1, 2\}$ corresponding to the events of the associated GSPN of Figure 1. The set L of locations consists of l_0 and l_1. Location l_0 is initial and location l_1 is final. There are no labelling functions (such as **acc1**, **noacc** and **acc2** which can be found in Figure 3). The set of data variables is $X = \{x_0, x_1, x_2\}$. Concerning the flow, the last two variables have rate 0 in l_0 whereas x_0 evolves with a rate that is equal to the marking of place Req_1 in Figure 1. The rate of variables in state l_1 is irrelevant as a path ends as soon as this state is reached. Concerning the transitions the top left loop on l_0 is triggered as soon as action $Serv_1$ is fired in the GSPN, it has trivial constraint $\top$ (True) and its update consists in incrementing x_2. The transition from l_0 to l_1 is not synchronised (label $\sharp$), has a left closed constraint $x_2 \geq k$ and no update.

Discussion. The automata we consider are deterministic in the following (non usual) sense. Given a path σ of a DESP, there is at most one synchronisation with the linear hybrid automaton. The three first constraints ensure that the synchronised system is still a stochastic process. The fourth condition disables "divergence" of the synchronised product, i.e. the possibility of an infinity of consecutive autonomous events without synchronisation.

It should also be said that the restriction to linear equations in the constraints and to a linear evolution of data variables can be relaxed, as long as they are not involved in autonomous transitions. Polynomial evolution of constraints could easily be allowed for synchronised edges for which we would just need to evaluate the expression at a given time instant. Since the best algorithms solving polynomial equations operate in PSPACE [20], such an extension for autonomous transitions cannot be considered for obvious efficiency reasons.

Notations. A *valuation* $\nu : X \to \mathbb{R}$ maps every data variable to a real value. In the following, we use Val to denote the set of all possible valuations. The value of data variable x_i in ν is denoted $\nu(x_i)$. Let us fix a valuation ν and a state s. Given an expression $exp = \sum_{1 \leq i \leq n} \alpha_i x_i + c$ related to variables and indicators, its interpretation w.r.t. ν and s is defined by $exp(s, \nu) = \sum_{1 \leq i \leq n} \alpha_i(s)\nu(x_i) + c(s)$. Given an update $U = (u_1, \ldots, u_n)$, we denote by $U(s, \nu)$ the valuation defined by $U(s, \nu)(x_k) = u_k(s, \nu)$ for $1 \leq k \leq n$. Let $\gamma \equiv exp \prec 0$ be a constraint, we write $(s, \nu) \models \gamma$ if $exp(s, \nu) \prec 0$. Let φ be a state proposition. We write $s \models \varphi$ if $\varphi(s) = \mathbf{true}$.

Semantics. The role of a synchronised LHA is, given an execution of a corresponding DESP, to decide whether the execution is to be accepted or not, and also to maintain data values along the execution. Before defining the model associated with the synchronisation of a DESP $\mathcal{D}$ and an LHA $\mathcal{A}$, we need to introduce a few notations to characterise the evolution of a synchronised LHA. Given a state s of the DESP, a non final location l and a valuation ν of $\mathcal{A}$, we define the effect of time elapsing by: $Elapse(s, l, \nu, \delta) = \nu'$ where, for every variable x_k, $\nu'(x_k) = \nu(x_k) + \mathit{flow}_k(l)(s) \times \delta$. We also introduce the autonomous delay $Autdel(s, l, \nu)$:

$$Autdel(s, l, \nu) = \min(\delta \mid \exists l \xrightarrow{\sharp, \gamma, U} l' \wedge s \models \Lambda(l') \wedge (Elapse(s, l, \nu, \delta)) \models \gamma)$$

Whenever $Autdel(s,l,\nu)$ is finite, we know that there is at least one executable transition with minimal delay and, thanks to the "determinism on $\sharp$" of definition 2, we know that this transition is unique. In the following we denote $Next(s,l,\nu)$ the target location of this first transition and $Umin(s,l,\nu)$ its update. We now proceed to the formal definition of the DESP, $\mathcal{D}' = \langle S', \pi'_0, E', Ind', enabled', target', delay', choice' \rangle$ associated with the synchronisation of a DESP $\mathcal{D}$ and an LHA $\mathcal{A}$.

- $S' = (S \times L \times Val) \uplus \{\bot\}$ among which $(S \times Final \times Val) \uplus \{\bot\}$ are the absorbing states.
- $\pi'_0(s,l,\nu) = \begin{cases} \pi_0(s) & \text{if } (l \in Init \wedge s \models \Lambda(l) \wedge \nu = 0) \\ 0 & otherwise \end{cases}$
  and $\pi'_0(\bot) = 1 - \Sigma_{s \in S, l \in L, \nu \in Val} \pi'_0(s,l,\nu)$.
  Note that this definition gives a distribution since, due to "initial determinism" of definition 2, for every $s \in S$, there is at most one $l \in Init$ such that $s \models \Lambda(l)$.
- $E' = E \uplus \{\sharp\}$
- $Ind' = \emptyset$. In fact Ind' is useless since there is no more synchronisation to make.
- if $Autdel(s,l,\nu) \neq \infty$ then
  $enabled'(s,l,\nu) = enabled(s) \cup \{\sharp\}$
  else
  $enabled'(s,l,\nu) = enabled(s)$
- $delay'((s,l,\nu),e) = delay(s,e)$ for every $e \in enabled(s)$ and, whenever $\sharp \in enabled'(s,l,\nu)$, $delay'((s,l,\nu),\sharp)$ is a Dirac function with a spike for the value $Autdel(s,l,\nu)$.
- $choice'((s,l,\nu),E',d)(e) =$
  $\begin{cases} 1 \text{ if } (\sharp \in E' \wedge e = \sharp) \\ 0 \text{ if } (\sharp \in E' \wedge e \neq \sharp) \\ 0 \text{ if } e \notin E' \\ choice(s,E',d)(e) \text{ otherwise} \end{cases}$
  Again this is coherent since, as soon as $\sharp \notin E'$, then E' is a subset of $enabled(s)$ on which $choice$ is thus defined.
- For a synchronised event e,
  if $e \in enabled(s)$ and there exists $l \xrightarrow{E',\gamma,U} l'$ with $e \in E'$ such that $target(s,e,d) \models \Lambda(l')$ and $Elapse(s,l,\nu,d) \models \gamma$ then
  $target'((s,l,\nu),e,d) = (target(s,e,d), l', \nu')$
  with $\nu' = U(Elapse(s,l,\nu,d))$
  else
  $target'((s,l,\nu),e,d) = \bot$
  Due to the determinism, there is at most one such transition.
- For an autonomous event $\sharp \in enabled'(s,l,\nu)$,
  $target'((s,l,\nu),\sharp) = (s,l',\nu')$ with $l' = Next(s,l,\nu)$ and
  $\nu' = Umin(s,l,\nu)(Elapse(s,l,\nu,Autdel(s,l,\nu)))$

- In order to get a DESP, for any absorbing state one adds a special event **tick** only enabled in absorbing states with Dirac distribution on value 1 (so that time diverges in these states).

Roughly speaking, as long as the automaton is not in a final state, the product of a DESP and an LHA waits for the first transition to occur. If it is an autonomous one then only the location of the automaton and the valuation of variables change. If it is a synchronised event triggered by the DESP, either the LHA can take a corresponding transition and the system goes on with the next transition or the system goes to a dedicated rejecting state $\perp$ implying the immediate end of the synchronisation. In case of a conflict of two transitions, an autonomous and a synchronised one, the autonomous transition is taken first.

Note that, by initial determinism, for every $s \in S$ there is at most one $l \in Init$ such that s satisfies $\Lambda(l)$. In case there is no such l the synchronisation starts and immediately ends up in the additional state $\perp$. Determinism on events (*resp.* on $\sharp$) ensures that there is always at most one synchronised (*resp.* autonomous) transition fireable at a given instant.

Example. The three LHAs of Figure 2, 3 and 4 intend to illustrate the expressiveness of LHAs in the context of HASL. When the first two are meant to synchronise with the shared resource system of Figure 1, the last one expresses a property of the flexible manufacturing systems presented in Figure 8. Note that in the LHAs presented hereinafter, $\sharp$ labels for autonomous transitions are omitted; furthermore, label E is used to denote *universal* synchronisation (i.e. synchronisation with any event). The example of Figure 2 uses indicator dependent flows and has already been explained on page 8.

The LHA in Figure 3 illustrates the interest of associating state propositions with locations of the LHA (function Λ of Definition 2). In the figure, three such propositions are used, associated to non final locations: $\mathbf{acc_1}$ (there is a token in place Acc_1), $\mathbf{acc_2}$ (there is a token in place Acc_2) and $\mathbf{noacc}$ (there is no token neither in Acc_1 nor in Acc_2). The interest of such propositions is that the automaton can take a transition to location l_1 only if $\mathbf{acc_1}$ has value 1 in the corresponding state of the system. Hence, starting from location $Init$, no matter which precise event occurs in the system, the automaton will switch from $Init$ to l_1 and l_2 depending on which class of clients has access to the resource. The fact that for example three different transitions labelled with E without any constraint are available in location $Init$ does not induce non determinism as only one of these transitions is possible at a time thanks to the state propositions. The automaton uses variables x_0 that is used to count the number of granted resource accesses and x_1 that expresses the difference of resource usage between clients of class 1 and 2. To do so variable x_1 has flow 1 (*resp.* -1) when the resource is used by class 1 (*resp.* 2) clients, and 0 when the resource is not used. In figures, we denote the flow of variable x by $\dot{x}$. As soon as k clients have been given a resource access, the system terminates in location l_3 or l_4 depending on which client type has used the resource for the longest cumulated period.

The LHA of Figure 4 is meant to compute the probability to have at least K product completions (events corresponding to transitions $Serv_1$ and $Serv_2$ of the FMS Petri nets of section 5) in a time interval of duration D during horizon mD. The three

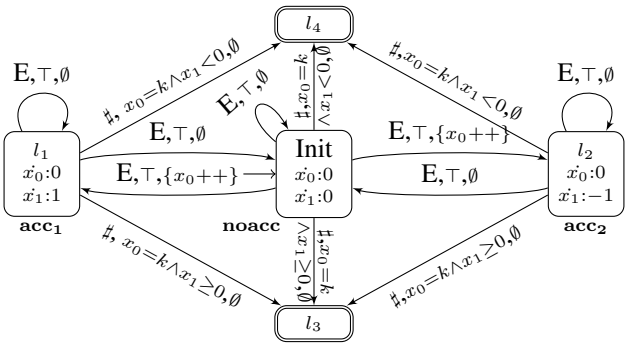


Figure 3: An LHA to compute the difference of shared resource usage.

variables represent respectively the total time (x_1), the time and the number of product completions since the beginning of the interval (x_2 and x_3). The automaton reaches a final state when the time horizon is reached (state l_1). The value of x_4 at the final location represents the proportion of successful K-completions.

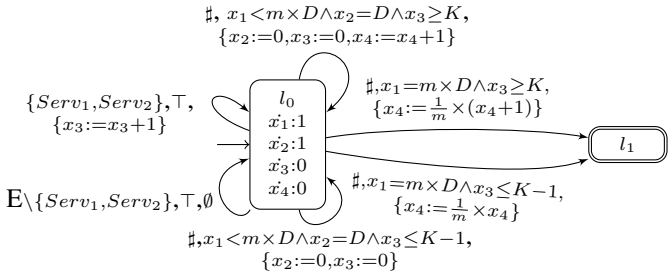


Figure 4: An LHA for experiment ϕ_3 of FMS case study (section 5.1)

3.2. HASL expressions

The second component of an HASL formula is an expression related to the automaton. Such an expression, denoted Z, is based on moments of a path random variable Y and defined by the grammar:

$$\begin{aligned} Z &::= c \mid P \mid \mathbb{E}[Y] \mid Z + Z \mid Z - Z \mid Z \times Z \mid Z/Z \\ Y &::= c \mid Y + Y \mid Y \times Y \mid Y/Y \mid LAST(y) \mid MIN(y) \\ &\quad \mid MAX(y) \mid INT(y) \mid AVG(y) \\ y &::= c \mid x \mid y + y \mid y \times y \mid y/y \end{aligned} \tag{1}$$

Preliminary assumption. Before explaining this syntax, we emphasise that given an infinite path of the synchronised product, the formula is evaluated on the finite prefix that ends when the path reaches an absorbing state. In the general case, it would be possible that an infinite path does not admit such a prefix. Here we assume that

given a DESP $\mathcal{D}$ and a HASL formula $(\mathcal{A}, Z)$, with probability 1, the synchronising path generated by a random execution path of $\mathcal{D}$ reaches an absorbing configuration. This semantical assumption can be ensured by structural properties of $\mathcal{A}$ and/or $\mathcal{D}$. For instance, the time bounded `Until` of CSL guarantees this property. As a second example, the time unbounded `Until` of CSL also guarantees this property when applied on finite CTMCs where all terminal strongly connected components of the chain include a state that either fulfils the target sub-formula of the `Until` operator or falsifies the continuing sub formula. This (still open) issue is also addressed in [44, 32].

The variable y is an arithmetic expression built on top of LHA data variables (x) and constants (c). A path variable Y is a path dependent expression built on top of basic path random variables such as $LAST(y)$ (i.e. the value of y when reaching the absorbing state), $MIN(y)$ (resp. $MAX(y)$), the minimum (resp. maximum), value of y along a synchronising path, $INT(y)$ (i.e. the integral over time along the finite prefix) and $AVG(y)$ (the average value of y along a path). Finally Z, the actual target of HASL verification, is either P, denoting the probability that a path is accepted by the LHA or an arithmetic expression. Such an expression is built on top of the first moment of Y ($\mathbb{E}[Y]$), and thus allowing for the consideration of diverse significant characteristics of Y (apart from its expectation) as the quantity to be estimated, including, for example, the variance $Var[Y] \equiv \mathbb{E}[Y^2] - \mathbb{E}[Y]^2$ and the covariance $Covar[Y_1, Y_2] \equiv \mathbb{E}[Y_1 Y_2] - \mathbb{E}[Y_1]\mathbb{E}[Y_2]$. Note that for efficiency reasons, in the implementation of the COSMOS software tool, we have considered a restricted version of grammar (1), where products and quotients of data variables (e.g. $x_1 \times x_2$ and x_1/x_2) are allowed only within the scope of the $LAST$ operator (*i.e.* not with MIN, MAX, INT or AVG). Indeed, allowing products and quotients as arguments of path operators such as MAX or MIN requires the solution of a linear programming problem during the generation of a synchronised $\mathcal{D} \times \mathcal{A}$ path which, although feasible, would considerably affect the computation time.

Example. With the LHA of Figure 2, we can express bounds for the average waiting time until the first k clients have been served. The upper (resp. lower) bound on the waiting time is computed by the final value of x_1 divided by k (resp. x_3). In our formalism it corresponds to the expressions $\mathbb{E}[LAST(x_1)/k]$ for the upper bound and to $\mathbb{E}[LAST(x_1)/LAST(x_3)]$ for the lower bound. Referring to the LHA of Figure 3, we can consider path random variables such as $Y = LAST(x_1)$ (the final difference of shared resource usage), or $Y = AVG(x_1)$ (the average along paths of such a difference). Furthermore, with a slight change of the automaton (setting x_0 to 0 (*resp.* 1) when reaching l_4 (*resp.* l_3)), $\mathbb{E}[LAST(x_0)]$ will give the probability to reach l_3. Finally, on the LHA of Figure 4, the ratio of time intervals of the form $[nD, (n+1)D[$ during which K product completions occur is evaluated through expression $\mathbb{E}[LAST(x_4)]$.

Semantics. We emphasise that the (conditional) expectation of a path random variable is not always defined. There are two obvious necessary conditions on the synchronised product: (1) almost surely the random execution ends either in a final state of the LHA or in the rejecting state, and (2) with positive probability the random execution ends in a final state of the LHA. However these conditions are not sufficient. Different restric-

tions on the path formula ensure the existence of expectations. For instance, when the formula only includes bounded data variables and the operator INT and the division are excluded, the expectation exists. Divisions may be allowed when the path expression is lower bounded by a positive value: in the example of Figure 2, $LAST(x_3)$ is lower bounded by $LAST(x_2)$ which is lower bounded by k. The existence of regeneration points of the synchronised product may also entail the existence of such expectations. We do not detail the numerous possible sufficient conditions but in all considered applications here, it can be proved that the expectations exist.

3.3. Expressiveness of HASL

In this subsection we first give an overview of related logics. Then we discuss the expressiveness of HASL and show how it improves the existing formalisms to capture more complex examples and properties, and facilitates the expression and the computation of costs and rewards.

CSL and its variants. In [7] Continuous Stochastic Logic (CSL) has been introduced and the decidability of the verification problem over a finite continuous-time Markov chain (CTMC) has been established. CSL extends the *branching time* reasoning of CTL to CTMC models by replacing the discrete CTL path-quantifiers `All` and `Exists` with a continuous path-quantifier that expresses that the probability of CTMC paths satisfying a given condition fulfils a given bound. Several variants have been introduced and studied such as CSRL [10], that extends CSL to take into account Markov reward models, i.e. CTMCs with a single reward on states or possibly on actions [40], that is used to specify an interval (on the accumulated reward) on the `Until` or `Next` operator. asCSL, introduced in [8] replaces the interval time constrained `Until` of CSL by a regular expression with a time interval constraint. These path formulas can express elaborated functional requirements as in CTL* but the timing requirements are still limited to a single interval globally constraining the path execution. In the logic CSL$^{\text{TA}}$ [28], path formulas are defined by a single-clock deterministic timed automaton. This logic has been shown strictly more expressive than CSL and also more expressive than asCSL when restricted to path formulas.

DTA. In [22], deterministic timed automata with multiple clocks are considered and the probability for random paths of a CTMC to satisfy a formula is shown to be the least solution of a system of integral equations. The cost of this more expressive model is both a jump in the complexity as it requires to solve a system of partial differential equations, and a loss in guaranty on the error bound.

M(I)TL. Several logics based on linear temporal logic (LTL) have been introduced to consider timed properties, including Metric (Interval) Temporal logic in which the `Until` operator is equipped with a time interval. Chen *et al.* [21] have designed procedures to approximately compute desired probabilities for time bounded verification, but with complexity issues. The question of stochastic model checking on (a sublogic of) M(I)TL properties, has also been tackled see e.g. [50].

Observe that all of the above mentioned logics have been designed so that numerical methods can be employed to decide about the probability measure of a formula.

This very constraint is at the basis of their limited expressive scope which has two aspects: first the targeted stochastic models are necessarily CTMCs; second the expressiveness of formulas is constrained by decidability/complexity issues. Furthermore the evolution of stochastic logics based on CTL seems to have followed two directions: one targeting *temporal reasoning* capability (in that respect the evolutionary pattern is: CSL $\rightarrow$ asCSL $\rightarrow$ CSL$^{\text{TA}}\rightarrow$ DTA), the other targeting *performance evaluation* capability (evolutionary path: CSL $\rightarrow$ CSRL $\rightarrow$ CSRL+impulse rewards). A unifying approach is currently not available, thus, for example, one can calculate the probability for a CTMC to satisfy a sophisticated temporal condition expressed with a DTA, but cannot, assess performance evaluation queries at the same time (i.e. with the same formalism).

HASL. As HASL is inherently based on simulation for assessing measures of a model, it naturally allows for releasing the constraints imposed by logics that rely on numerical solution of stochastic models. From a modelling point of view, HASL allows for studying a broad class of stochastic models (i.e. DESP), which includes, but is not limited to, CTMCs. From an expressiveness point of view, the use of LHAs allows for generic variables, which include, but are not limited to, clock variables (as per DTA). This means that sophisticated temporal conditions as well as elaborate performance measures of a model can be accounted for in a single HASL formula, rendering HASL a unified framework suitable for both model-checking and performance and dependability studies. Note that the nature of the (real-valued) expression Z, available in grammar (1) page 14, generalises the common approach of stochastic model checking where the outcome of verification is (an approximation of) the mean value of a certain measure (with CSL, asCSL, CSL$^{\text{TA}}$ and DTA a measure of probability).

It is also worth noting that the use of data variables and extended updates in the LHA enables to compute costs/rewards naturally. The rewards can be both on locations and on actions. First using an appropriate flow in each location of the LHA, possibly depending on the current state of the DESP we get "state rewards". Then, by considering the *update expressions* on the edges of the LHA, we can model sophisticated "action rewards" that can either be a constant, depend on the state of the DESP and/or depend on the values of the variables. Thus HASL extends the possibilities of CSRL (and extensions [40]). The extension does not only consist of the possibility to define multiple rewards (that can be handled, for example, through the reward-enriched version of CSL supported by the PRISM [42] tool) but rather of their use. First several rewards can be used in the same formula, and last but not least these rewards have a more active role, as they can not only be evaluated at the end of the path, but they can also take an important part in the selection of enabled transitions, hence of accepted paths. It is for example possible and easy to characterise the set of paths along which a reward reaches a given value and after that never goes below another value, a typical example that neither PRISM-CSL nor CSRL can handle.

Limitations of HASL. Finally, we briefly discuss two features that are available in the above mentioned stochastic logics but not in HASL. First HASL does not allow to properly model nesting of probabilistic operators. The key reason is that this nesting is meaningful only when an identification can be made between a state of the proba-

bilistic system and a configuration (comprising the current time and the next scheduled events). While this identification was natural for Markov chains, it is not possible with DESP and general distributions that have no memoryless property, and therefore this operation has not been considered in HASL. Furthermore, even for Markovian systems, the complexity of the statistical method on formulas with nesting is quite high [48] as the verification time per state along a path is no longer constant.

A similar problem arises for the steady state operator. The existence of a steady state distribution raises theoretical problems, except for finite Markov chains. With HASL we allow for not only infinite state systems but also non Markovian behaviours. However, when the DESP has a regeneration point, various steady state properties can be computed considering the sub-execution between regeneration points.

In conclusion it is worth noting that these limitations are rather due to the verification method (statistical in our case) and to the expressiveness of the model (allowing non Markovian systems) than to a particular tool. All this information given, particularly concerning nested and steady state formulae, we can now state and prove our claim about the respective expressiveness of HASL, CSRL and CSL$^{\text{TA}}$:

Proposition 1. *Given a non nested transient CSRL formula $P_{\bowtie q}\phi$ and a system described as a Markov Reward Model, it is possible to build an LHA to estimate the probability p for ϕ to hold, and then decide whether it fulfils the bound required (*i.e. *$p \bowtie q$ with $\bowtie \in \{<, >, \leq, \geq\}$).*

To prove this proposition, we first need to characterise what is a non nested transient CSRL formula. Following the grammar given in [10], such a formula is either a boolean combination of atomic propositions, or of the form $P_{\bowtie q}X^I_J\varphi$ or $P_{\bowtie q}\varphi\ \mathcal{U}^I_J\ \psi$ with φ and ψ being boolean combinations of atomic propositions. Roughly speaking, path formula $X^I_J\varphi$ means that the first transition in the system will occur after a time delay in interval I and accumulated reward within interval J, and that it will lead to a state satisfying φ. The second path formula $\varphi\ \mathcal{U}^I_J\ \psi$ means that the path reaches, with a delay in I and a cumulated reward in J a state satisfying ψ, and that all preceding states satisfy φ. The case of boolean formulas being rather trivial, we will focus on $P_{\bowtie q}X^I_J$ and $P_{\bowtie q}\mathcal{U}^I_J$

The kind of systems on which CSRL formulas are checked is called Markov Reward Model (MRM), which consists of a finite state labelled continuous time Markov chain plus two reward structures: one on actions and one on states. In our formalism, the MRM will be represented by both a GSPN and an LHA. The underlying labelled Markov chain will be represented by a GSPN. For each state s_i of the MRM we create one place P_i of the GSPN, and an indicator p_i that is true when a token is in place P_i, and for every couple of states (s_i, s_j) such that the rate in the MRM is $R(s_i, s_j) = r_{ij} > 0$, we add a transition t_{ij} with input place P_i, output place P_j and exponential distribution using rate r_{ij}. The atomic propositions of the MRM are simply encoded as indicators, now on places instead of states.

For a formula of the kind $P_{\bowtie q}X^I_J\varphi$ the LHA is quite simple. First the boolean formula φ is transformed straightforwardly into a state proposition p_φ on the GSPN. It has one initial location $Init_i$ per state s_i of the MRM, plus two final locations, one called f_1 labelled with p_φ, the other f_2 with $\neg p_\varphi$. It also has three data variables, t with rate 1 in every location for the global time, OK with rate 0 in every location to determine

whether an execution satisfies the property or not, and r to capture the reward structure. In order to ensure initial determinism (see Definition 2), each $Init_j$ is labelled with indicator p_j. The reward structure is captured by a data variable r whose rate in $Init_i$ is $\rho(s_i)$, the reward on state s_i in the MRM. Then, for every transition (s_i, s_j) with impulse reward $\iota(s_i, s_j)$ of the MRM, the LHA has three transitions starting from $Init_i$. The first one, taken if state s_j of the MRM does not satisfy φ, sets OK to 0 and leads to location f_2. The two other transitions lead to f_1 and set OK to 1 or 0 depending on whether the interval constraint $c = t \in I \wedge r + \iota(s_i, s_j) \in J$ is satisfied or not. The probability to satisfy $X^I_J\varphi$ is computed by the expression $\mathbb{E}[LAST(OK)]$ that can then be compared to bound q.

As for `Until` operator, it is also possible to build an LHA to compute the desired probability, but this automaton is more complicated. In order not to make this part too long, we just give here an idea of the LHA checking `Until` formulas. It is again built based on the corresponding MRM. First, if the value 0 is not both in I and J, for the states satisfying $\neg\varphi$ the synchronisation ends immediately setting OK to 0. If 0 belongs to both intervals then, starting in a state satisfying $\neg\varphi \wedge \psi$, the synchronisation immediately ends setting OK to 1. For all other states of the MRM we create a corresponding location in the LHA from which we will follow the evolution of the MRM (with rates on locations modelling the state rewards and updates modelling the impulse rewards). A further distinction has to be made between locations satisfying both φ and ψ and others. Indeed, in the first case, time can pass without further transition of the GSPN and the formula become true. This has to be detected. Otherwise, the transitions will then synchronise with those of the GSPN and check whether time and reward after the MRM transition are within the interval bounds. If yes then the execution will either end if ψ is satisfied or if neither φ nor ψ are, setting OK to 1 and 0 respectively, or go on until the above solution occurs. If time or reward passes over its interval, the synchronisation ends setting OK to 0. In all other cases the synchronisation goes on.

On the automata built according to the above described method, the desired probability can then be computed using again expression $\mathbb{E}[LAST(OK)]$ and compared to the desired bound

Proposition 2. *Given a non nested transient CSL^{TA} formula $P_{\bowtie q}\mathcal{A}(\phi_1, ..., \phi_n)$ and a system described as a Continuous Time Markov Chain, it is possible to build an LHA to estimate the probability p for an execution to be accepted by the DTA $\mathcal{A}(\phi_1, ..., \phi_n)$, and then decide whether it fulfills the bound required (*i.e. $p \bowtie q$ *with* $\bowtie \in \{<, >, \leq, \geq\}$*).*

Since DTA is a class of automata that is a strict subset of LHA , the construction is rather simple since the DTA can itself be used. It however has to be slightly modified. Indeed, the DTA was rejecting executions not satisfying the property. For the LHA we need to accept all executions and add a variable OK that is set to 0 or 1 depending whether the execution satisfies the desired property. Thus all transitions leading to an accepting location are enriched with the update OK:=1. Furthermore, for every location of the automaton we add transitions setting OK to 0 and leading to a new accepting location KO to ensure that every transition of the GSPN can synchronise with the LHA. For example in a location with a single outgoing transition labelled

$e, x \leq a, \emptyset$ we add two transitions to KO with label $E \setminus \{e\}, true, OK := 0$ and $e, x > a, OK := 0$.

4. Software Support

In order to provide a software support to the HASL formalism we developed COSMOS[3] [11], a prototype software platform for HASL-based verification. In this section, we describe the COSMOS tool including its architecture and providing as well a comparison with other platforms featuring statistical model checking functionalities: PRISM [42], UPPAAL-SMC [17], MARCIE [33] and PLASMA [36]. We start with a brief summary of *confidence interval* estimation, the statistical method that COSMOS relies on.

4.1. Confidence Interval Estimation.

In statistics, Confidence Interval (CI) estimation is a method for estimating an interval which is likely to contain the exact value that an (unknown) parameter θ of a certain population may assume. The peculiarity of CI estimation is twofold: i) it allows for specifying how reliable the estimate should be by choosing the desired *confidence level* $\alpha \in (0, 1)$); ii) it allows for specifying how accurate the estimate should be by choosing the desired admitted error bound (i.e. the width δ of the resulting interval). In other words, if we repeatedly estimate the interval for a given θ we are guaranteed that the (possibly different) resulting intervals will contain the actual value of θ in a proportion corresponding to $(1 - \alpha)$.

Static sample size estimation. Originally CI estimation works by collecting (through execution of experiments) a fixed number n of samples $X_1, \ldots X_n$ of the target parameter θ. Sampled values are then used to calculate the interval containing θ. The general form of the $100(1 - \alpha)\%$ confidence-interval for the expected value μ_θ of θ, denoted $CI^{\alpha}_{\mu_\theta}$, is:

$$CI^{\alpha}_{\mu_\theta} = (PE_{\mu_\theta}) \pm EB_\alpha$$

where PE_{μ_θ} is a Point Estimate of μ_θ and EB_α is the Error Bound, which corresponds to the semi-width of the CI interval, i.e. $EB_\alpha = \delta/2$. The sample mean $\overline{X} = \frac{\sum_{i=1}^{n} X_i}{n}$ is used as (*unbiased*) Point Estimator of μ_θ. On the other hand the expression for the EB depends on assumptions concerning the nature of target parameter θ (hence of samples $X_1, \ldots X_n$). COSMOS handles three kinds of variables:

- Indicator variables (Bernoulli variables) used for evaluation of unknown probabilities. In this case, we provide an error bound applying the Clopper-Pearson method [25].

[3]COSMOS is an acronym of the french sentence "*Concept et Outils Statistiques pour le MOdèles Stochastiques*" whose english translation would sound like: "Tools and Concepts for Statistical analysis of Stochastic MOdels".

- Bounded variables used for evaluation of proportions, ratios, mean number of clients in a system with finite capacity, etc. In this case, we provide an error bound applying the Chernoff-Hoeffding method [35].
- General variables without any additional knowledge. In this case using the central limit theorem, we provide an asymptotically correct error bound based on an approximation by a normal distribution with unknown mean and variance.

There are three parameters for the interval estimation: the size of the samples, the error bound and the confidence level. In the case of indicator or bounded variables, the user provides two of them and the third one is computed before the sampling. For general variables, the size of the sample, and one more parameter (the error bound or the confidence level) must be given, and the remaining one is computed after the sampling. Thus, the sample size is fixed before the sampling.

Dynamic sample size estimation. When the user wants to *a priori* provide the error bound and the confidence level for a general variable, the system can perform a dynamic number of samples depending on some stopping condition. While an exact condition cannot be achieved, Chow and Robbins [24] provide a stopping condition ensuring that when the error bound goes to 0, then the probability that the unknown expectation belongs to the interval associated with their method converges to the confidence level. COSMOS also offers this functionality. Compared to conservative methods like Clopper-Pearson and Chernoff-Hoeffding ones, the simulation time is significantly reduced, as illustrated in Figure 6.

Confidence interval for expressions. The previous paragraphs deal with the case of a single estimation. HASL expressions include an arbitrary number of estimations denoted by operator $\mathbb{E}[]$ (see Equation (1)). Assume that we have to perform n estimations for which we get CI $[a_i, b_i]$ and confidence level α_i for $i = 1 \dots n$. Then the confidence level of the whole expression is given by $\alpha = 1 - \sum_{i=1}^{n} 1 - \alpha_i$. The CI of the expression is obtained by applying operations on intervals like:
$[a, b] + [a', b'] = [a + a', b + b']$, $[a, b] - [a', b'] = [a - b', b - a']$,
$[a, b] \times [a', b'] = [\min(aa', ab', ba', bb'), \max(aa', ab', ba', bb')]$, etc.

4.2. Hypothesis testing

When one is not interested in the actual value of a statistic but rather wants to decide whether this value is above or below a threshold, hypothesis testing methods can be used. For a Bernoulli variable of unknown mean p, and given two probabilities $p_0 < p_1$ (the maximal probability of false positive and true negative results), the Sequential Probability Ratio Test (SPRT) [47] is an optimal sequential test for deciding whether $p \geq p_0$ or $p \leq p_1$ holds.

4.3. The COSMOS *Tool*

COSMOS *code generation scheme.* COSMOS is implemented in C++ and relies on the BOOST libraries for random number generation functionalities. The tool is designed according to a *model driven code generation* scheme (Figure 5): the inputs $\mathcal{D}$ and

$\mathcal{A}$ are parsed in order to generate a C++ code that implements the simulation of the synchronized product of the inputs. The code generator takes advantage of the structure of the GSPN and the LHA to produce an efficient code.

More precisely the code generation works as follows: From the GSPN given as input the tool produces C++ functions for firing transitions, checking whether a transition is enabled and computing the probability distributions. For any transition t, the set of transitions that might become enabled or disabled due to the firing of t are also generated. These informations are obtained by an analysis of the structure of the Petri net. They significantly increase the speed of simulation by testing only a small subset of transitions after each firing.

In order to perform an efficient synchronization of the LHA with the GSPN, the tool generates data that links synchronized transitions of the automaton with transitions of the net. Checking the firing of an autonomous transition of the LHA requires to generate a function that manages a system of linear equations associated with the transition. Due to the syntactical constraints for autonomous transitions, the function computes the exact firing time of such a transition. A timed integral occurring in a formula of HASL is also efficiently determined thanks to the linear constraint requirement.

The generated code is linked with a library containing parts of the simulator that are independent of the model and the formula. This library contains the main function that determines the next event to occur by means of an event heap and the generated code. In addition a pseudo random generator computes delays for the new events that are put into the heap.

Figure 5: COSMOS's model-driven code generation scheme

COSMOS launches several copies of the resulting executable code in parallel that repeatedly generate trajectories and send back the evaluation of the formulas on these trajectories. COSMOS aggregates these evaluations and stops the simulation depending on the selected statistical method (see above). The compilation time is generally negligible compared to the simulation time.

Input and Output files. COSMOS uses the XML-based file format of CosyVerif [26] named GrML both for the GSPN and the LHA. COSMOS can output results in different ways. By default the mean value and confidence interval of each HASL formula is written in a text file with some statistics about the simulation (number of generated paths, execution time, etc.). Several options allow to output intermediate results, traces of simulation and graphics.

Interface. COSMOS interface can be either a command line tool or a graphical user interface. The command line requires as parameters the path to the GSPN and LHA files and several options which set the statistical parameters and the output format. COSMOS is integrated into the CosyVerif [26] platform which provides a graphical interface.

HASL Implementation. COSMOS implements a slight extension of HASL. This extension does not enhance the expressive power of HASL but adds some macros allowing

more compact syntax. First, the operator VAR is provided as a macro for $\mathbb{E}[Y^2] - \mathbb{E}[Y]^2$. Second, two macros allow to compute PDF (Probability Density Function) and CDF (Cumulative Density Function). Their syntax is $PDF(Y, step, start, end)$ (respectively $CDF(Y, step, start, end)$) where Y is an expression defined like in (1). PDF (resp. CDF) is translated by COSMOS into several HASL formulas, each of them compute the probability for $Y \in [start + i \cdot step; start + (i + 1) \cdot step[$ (resp. $Y \geq start + i \cdot step$) with $i < (end - start)/step$. An example of their use is provided in the second case study.

4.4. Related Tools

Numerous tools are available for performing SMC, some of them also performing numerical model checking. Here is a non exhaustive list of tools freely available for universities: COSMOS [13], PLASMA [37], PRISM [39], UPPAAL [18], MARCIE [33], APMC [34], YMER [49], MRMC [38] and VESTA [43]. As APMC is partly integrated in PRISM, we have discarded it. Since 2011, MRMC has not been updated and the corresponding team seems to use UPPAAL. Finally, the link for downloading VESTA is not valid anymore. So we focus on the following tools: YMER, PRISM, UPPAAL, PLASMA, MARCIE and COSMOS.

YMER is a statistical model checker for CTMCs and generalised semi-Markov processes described using the PRISM language. Its property specification language is a fragment of CSL without the steady-state operator but including the unbounded `Until`.

PRISM is a tool for performing model checking on probabilistic models that has been used for numerous applications. The numerical part of PRISM can analyse discrete and continuous time Markov chains, Markov decision processes and probabilistic timed automata. The statistical part only deals with Markov chains as it cannot handle nondeterminism. The PRISM language defines a probabilistic system as a synchronised product between reactive modules, thus can describe large systems in a compact way. The verification procedures of PRISM take as input a wide variety of languages for the specification of properties. Most of them are based on CSL or PCTL.

UPPAAL is a verification tool including many formalisms: timed automata, timed games, priced timed automata, etc. It supports automata-based and game-based verification techniques and has shown its ability to analyse large scale applications. It has recently been enriched with a statistical model checker engine to verify timed systems with a stochastic semantics. The specification language is PLTL (i.e. an adaptation of LTL with path operators substituted for quantifiers) with bounded `Until`.

PLASMA is a platform dedicated for statistical model checking. It accepts the PRISM language but extended with more general distributions and a dedicated biological language for the models. The property specification language is a restricted version of PLTL with a single threshold operator. Furthermore, PLASMA is built with a plugin system allowing a developer to extend it, and it can be integrated in another software via a library.

MARCIE is a tool for qualitative and quantitative analysis of Generalised Stochastic Petri nets. It relies on Interval Decision Diagram (IDD) to represent symbolically the state space of the Petri net. The implementation of IDD is mostly parallel, taking advantages of multicore architectures. It has been recently extended with a simulation

engine for the model checking of PLTL formulas. Like COSMOS, MARCIE can deal with unbounded until properties as long as the user guarantees the termination. It has been developed for the study of chemical reaction networks and thus facilitates the modelling of such systems.

Discussion. The formalisms are characterised by different features: model specifications (dedicated languages in PRISM or standard formalisms in COSMOS, MARCIE and UPPAAL), expressiveness of the formalisms (supported distributions, presence of clocks, etc.) and dedicated application based languages (e.g. for biological systems in PLASMA and MARCIE). While the probabilistic extension of UPPAAL is similar to COSMOS, the differences come essentially from the initial motivations. As UPPAAL intends to attribute a stochastic semantics to originally non probabilistic timed automata, the available probability distributions are restricted (exponential for transitions having guards without upper time bound, and uniform otherwise). COSMOS which is constructed as a statistical model checker provides several well known distributions (exponential, normal, deterministic, uniform, gamma, etc.). Even if any general distribution can be approximated by combining several exponential distributions, the simulation cost increases with the required accuracy. MARCIE only supports exponential and immediate distributions.

The property specification is expressed by a formula for most of the tools but it is a combination of an expression and an automaton for COSMOS. The available operators in formulas often exclude unbounded `Until` and nesting of probabilistic operators (only possible for Markovian models). Moreover the evolution of the time and data variables are subject to some restrictions.

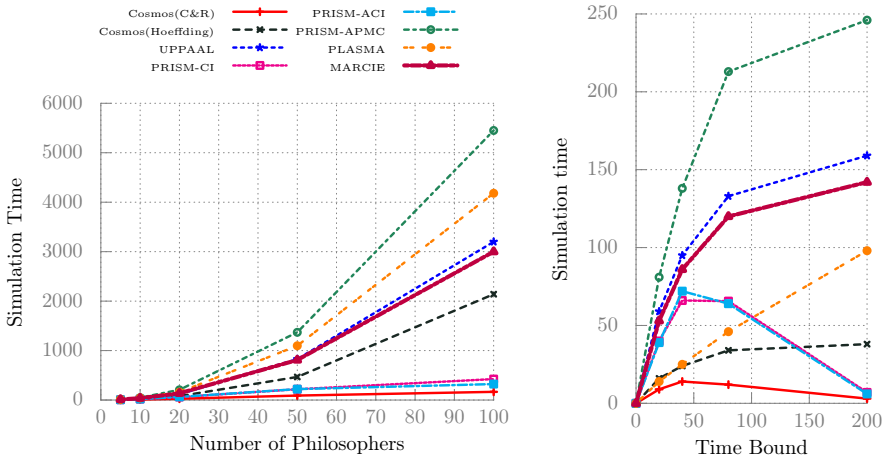
4.5. Tool Evaluation

We performed several experiments aimed at evaluating COSMOS both in terms of accuracy and runtime. For this purpose we consider two popular workbench models. The first one, a tandem queuing system (TQS), is available on the PRISM web page [42], the second one is a model of dining philosophers (DPM). We run experiments with COSMOS, PRISM (version 4.0.2), UPPAAL-SMC (version 4.1.13), PLASMA (version 1.1.4), YMER (version 3.1) and MARCIE (version 1178M). These are the stable version available from the corresponding web sites.

The TQS is a $M/Cox_2/1$ queue composed with a $M/M/1$ queue. For the experiments, we use queue capacities : $N = 5$, the arrival rate : $\lambda = 20$, the service rates of the first phase in the first queue : $\mu_1 = 0.2$, $\mu_1' = 1.8$ (for clients without second service phase), the service rate of the second phase in the first queue : $\mu_2 = 2$ and the service rate in the second queue : $\kappa = 4$ as in [39]. The DPM is a mutual exclusion problem where N philosophers are sitting around a table. Initially thinking, they can decide to eat by taking two forks shared with their right and left neighbours. However a contention problem may arise due to the sharing of resources (forks). For the experiments, the rate of all exponential distributions is chosen as 10. PRISM supports Chernoff-Hoeffding, SPRT methods and the sequential confidence interval computation using approximations to Student and Normal laws. UPPAAL and PLASMA support Chernoff-Hoeffding and SPRT methods while YMER only supports SPRT method. MARCIE uses a static sample size algorithm which is not described in the manual. So

for experiments with MARCIE, we set the number of samples to the one required by Chernoff-Hoeffding method. COSMOS provides Chernoff-Hoeffding, Chow-Robbins and Gaussian methods for the sequential confidence interval estimation and the SPRT method.

For the TQS model we consider the following time-bounded reachability measure: $\phi_{TQS} \equiv$ *The probability that the first queue in the tandem gets full within time* T. For the dining philosopher model we consider the following measure: $\phi_{DPM} \equiv$ *The probability to reach a deadlock state before* N *philosophers eat*. A deadlock occurs when all philosophers have taken one fork. Both properties can be straightforwardly encoded in CSL and HASL, hence equivalent verification experiments can be performed by all tools.



(a) The philosophers model

(b) The TQS model

Figure 6: Comparison of simulation time for probability measures

Experiment settings. We have set the following statistical parameters: the confidence level is 0.95 and the width of confidence interval is 0.005, the probability of error is 0.005 for the hypothesis testing and the width of the indifference region is 0.001. To fulfil these parameters, tools must generate large number of trajectories. The other parameters have been set to their default value. Most tools can take advantage of parallelisation but for simplicity we use only one processor[4] for the comparison.

DPM experiments. Figure 6(a) refers to the runtime for the DPM as a function of the number of philosophers. COSMOS is the fastest of the tools using Chernoff-Hoeffding bounds. For 100 philosophers, MARCIE is 1.4 times slower, UPPAAL is 1.5 times slower, PLASMA is 1.9 times slower, and PRISM-APMC is 2.5 times slower. Among the tools us-

[4]These experiments have been executed on a MacBook Pro, with processor 2.4 GHz Intel Core 2 Duo.

ing sequential procedures, the two versions of PRISM have similar runtime and COSMOS is up to 1.9 times faster.

TQS experiments with confidence intervals. Results about the runtime comparison with different time bounds T are reported in Figure 6(b). There are two kinds of behaviours for tools depending on the applied statistic method. For the first one that corresponds to the Chernoff-Hoeffding method, the simulation time is increasing with the time bound T. About 295000 trajectories are required to obtain the specified confidence interval. For the second one that corresponds to sequential confidence interval methods, the required number of samplings decreases when the time bound goes to infinity. This phenomenon is a consequence of the evolution of the satisfaction probability of ϕ_{TQS} that goes to 1 when T goes to infinity.

COSMOS is again the fastest of the tools using Chernoff-Hoeffding method. When the time bound is 200, PLASMA is approximatively 2.6 times slower, MARCIE is 3.7 times slower, UPPAAL is 4.2 times slower and PRISM-APMC is 6.5 times slower. Among the tools using sequential procedures, when the time bound is 40 the two versions of PRISM have similar runtime and COSMOS is up to 2.8 times faster.

4.5.1. TQS experiments with sequential hypothesis testing.

Results on hypothesis testing are reported in Table 1. Each value is the mean over 100 experiments. The threshold value for the hypothesis is always very close the numerical value in order to increase the number of trajectories performed by tools. Results confirm that hypothesis testing methods are faster than confidence interval based methods. The number of trajectories generated by all tools are similar to each other. In most cases COSMOS is the fastest, YMER being up to 1.8 times slower, UPPAAL 2.8 times slower, and PRISM 6 times slower.

Figure 7: Runtime for a probability measure of the TQS model for Sequential Testing

Time	NumValue	$p \geq ?$	UPPAAL	PRISM	YMER	COSMOS
10	0.17505	0.17	4.78	12.02	3.29	3.36
20	0.33574	0.33	11.54	23.78	7.48	5.08
40	0.56931	0.564	21.23	46.47	14.00	7.78
80	0.81894	0.814	20.10	43.46	13.01	7.60
200	0.98655	0.981	2.81	8.10	1.92	2.74

Table 1: Runtime comparison for the TQS for Sequential Testing

Accuracy comparison. To assess the accuracy of COSMOS we compared its output with the one produced by PRISM via both its *numerical* engine (PRISM-n) and its *statistical* engine (PRISM-s)[5]: results indicate that COSMOS and PRISM-s are comparably accurate with the estimated intervals always containing the (actual) value obtained with

[5]Experiments were executed with confidence level=99.99%, interval width=0.01.

the numerical engine of PRISM. Using the value computed by the numerical engine of PRISM, we perform a coverage test of COSMOS: we compute the ratio of simulations that return a confidence interval which contains the real value. This ratio is always close to the confidence level.

5. Case Studies

In this section we demonstrate the potential of HASL model checking through two different applications, namely an industrial modelling and a biological modelling application. In the first case study we develop a complete (GSPN) model of a production process, which includes the representation of different routing policies and we analyse them through dedicated HASL formulae. In the second case study we focus on a model of biological oscillator (i.e. the circadian-clock) for which we show how we can perform a detailed analysis of its oscillatory characteristics (i.e. period analysis) by means of HASL.

5.1. Flexible Manufacturing System

Flexible Manufacturing Systems (FMS) design has been introduced to optimise manufacturing systems according to different criteria. The goal is to evaluate and compare several manufacturing system architectures before selecting the suitable one according to a selection of specific criteria. This process implies resorting to formal models and evaluation methods. Although the vast majority of FMS stochastic modelling studies have been focused on the analysis of *steady-state*-based measures such as *throughput*, *productivity*, *makespan* [19], the relevance of *transient-analysis* of FMS models has been extensively demonstrated [41]. In [14], we have proposed a compositional framework targeted to modelling FMSs by means of the HASL verification approach. Here we focus on the transient analysis and more precisely with fixed horizon: thus it can be straightforwardly shown that all expectations related to these transient properties exist. Furthermore, it is well known that for systems presenting regenerative points (like idle states), every steady-state measure can be obtained by averaging the corresponding transient measure between two occurrences of a regenerative point.

5.1.1. FMS model

We now proceed to the description of our FMS case study. We first state the architecture of the system, composed of two machines served by two conveyor belts. Then we present two possible routing policies, and finally we describe the experiments we performed on this system. The numerical results as well as their interpretation are given in section 5.1.2.

Architecture. The raw material arrives to an unbounded buffer which is represented by place *Products* in GSPN models (see Figure 8). The inter-arrival times are distributed according to a uniform distribution within interval $[a1, a2]$. A uniform distribution is chosen, rather than a deterministic one, in order to represent small variabilities due to logistic problems. The raw materials are oriented to one of the two conveyor belts according to a strategy that will be explained later. Each conveyor belt $i \in \{1, 2\}$

consists of an unbounded input buffer, represented by place *Buffer*$_i$ in the GSPN model, and a set of equally distanced positions, represented by places *Pos*$_i^j$. Here we consider 4 positions, $j \in \{1, ..., 4\}$. The transitions *Mv*$_i^j$ denote the movements on the conveyor i, from *Buffer*$_i$ all the way to the input buffer Q_i of machine i. Transitions *Mv*$_i^j$ with $j > 1$ and *Start*$_i$ are deterministic with parameter T_{unit} which represents transport time of one pallet between two successive positions. Transition *Mv*$_i^1$ is immediate. Priorities denoted π_k in the figure are introduced to ensure that all tokens progress simultaneously (without them, a token could move from *Pos*$_i^4$ to Q_i, blocking the conveyor belt before all tokens have moved forward). They satisfy the condition $\pi_k < \pi_l$ whenever $k < l$.

Conveyor belt i drives materials to place Q_i that represents both the bounded input buffer of machine i and the output buffer of conveyor i. When it is full (i.e. two tokens in Q_i), the conveyor belt i is blocked (as ensured by inhibitor arcs of weight 2). The service in machine i is represented by transition *Serve*$_i$ in the GSPN model. The service duration in machine i follows a lognormal distribution, $Ln\mathcal{N}(\mu, \sigma^2)$;[6]

We consider two models. In the first one we suppose that there is no machine failure so the service is always available, we refer to this model as M_1. In the second one that we refer as M_2, we suppose a failure/repair model for each machine. The time between two successive failures is distributed according to an Erlang distribution. The reparation time is uniformly distributed in the interval $[r1, r2]$.

Routing policies. In manufacturing systems, the routing policy to send material to one among several machines doing the same job is an important question as a bad policy can have a significant impact on the efficiency of the whole system. Here we consider two policies S_1 and S_2 to choose the transport unit (conveyor belt) to drive raw material from the input buffer to one of the machines.

Policy S_1.
If one of the conveyor belts is blocked and the other is available, then put the material to the available one (transitions $\mathbf{In_i^1}$). If both conveyor belts are available and if the number of material in conveyor i is greater than a given threshold l_i while for the other conveyor is less than its own threshold l_j, put the material in the conveyor which does not exceed its threshold ($\mathbf{In_i^2}$). Otherwise, one of the conveyor is randomly chosen with probability 0.5 ($\mathbf{In_i^2}$ or $\mathbf{In_i^3}$).

Policy S_2.
Choose the conveyor belt with the smallest number of occupied positions. If these numbers are equal, choose randomly a conveyor with probability 0.5 ($\mathbf{In_i}$ with inhibitor arcs).

The top GSPN model of Figure 8 represents a FMS with policy $S1$ and non-failing machines (model M_1) while the bottom one represents a FMS with policy $S2$ and failing/repair machines (model M_2). The two other options (policy $S1$, model M_2 and policy $S2$, model M_1) can easily be deduced from the figures given by adding/removing fail and repair transitions.

[6] with scale parameter μ and shape parameter σ^2. The expectation is given by $e^{\mu+\sigma^2/2}$ and the variance by $(e^{\sigma^2} - 1)e^{2\mu+\sigma^2}$.

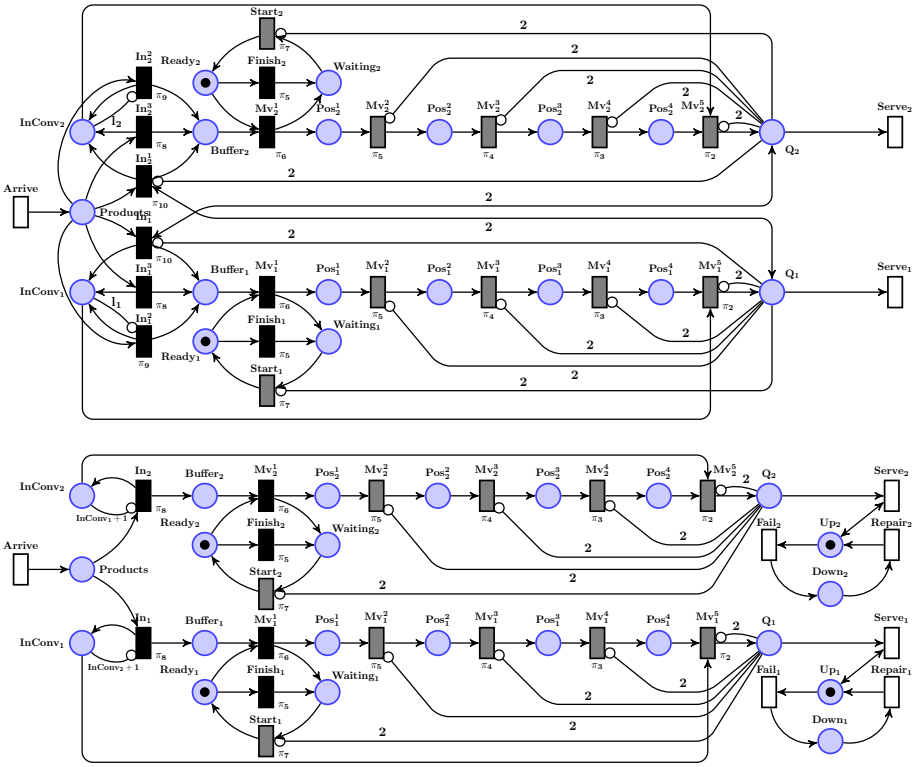


Figure 8: GSPNs for policies S_1 (above) and S_2 (below).

Evaluating the system. We have chosen several meaningful properties in order to assess the quality of the FMS design w.r.t. different model assumptions: routing policies, and presence of failures. We also study the behaviour of COSMOS and in particular its runtime and the accuracy of its results (witnessed by the width of the confidence intervals).

ϕ_1 : First we want to characterise the bottlenecks of the architecture. More precisely, a large ratio of blocking time for the conveyor may indicate that the buffer of the server should be enlarged. Furthermore if some cost is associated with the restarting of the conveyors, decreasing this ratio can induce significant savings. So ϕ_1 denotes the ratio of blocking time for conveyor 1. Since we study this formula in a symmetric framework the choice of the conveyor is irrelevant.

ϕ_2 : In order to support additional load due to client requests, it is important to estimate the average completion time for a product. Using Little formula (on the long run), it is equivalent to compute the expected number of products in the system which is denoted by ϕ_2. We estimate this value depending on the relative rate of the two machines.

ϕ_3 : For logistic issues, the time is divided in periodic intervals. It is required to guarantee a certain number of production on each of these cycles. This can be charac-

terised by some thresholds relative to the number of items produced inside an interval. Failing to meet this threshold may have dramatic consequences for the company. So ϕ_3 is the probability to produce at least K products (the threshold) in a time interval of the form $[iD, (i+1)D[$ for $0 \leq i < m$ during horizon $T = mD$ (see Figure 4 for the LHA expressing this formula).

5.1.2. Numerical results

Unless specified otherwise, all numerical results have been obtained with a confidence interval level 0.99 and with a confidence interval width 0.01. The transport time of pallets between two successive positions is $Tunit = 0.5$ time unit. In the following tables, T denotes the simulation horizon, S.T. denotes the simulation time in seconds, N.P. denotes the number of paths for the required accuracy of the estimation, and C.I. denotes the confidence interval width. In the experiences, the inter-arrival distribution of raw material is *Unif*$[0.45, 0.55]$. We first study model M_1 without machine failures. In table 2, we consider property ϕ_1 with respect to horizon T under policy S_1 with thresholds $l_1 = 3$, $l_2 = 3$ and under policy S_2. The service rates in machines are symmetric and the service distribution is $Ln\mathcal{N}(-0.683046, 0.83255)$ with expectation $1/1.4$ and variance $(1/1.4)^2$.

T	S_1			S_2		
	ϕ_1	S.T.	N.P.	ϕ_1	S.T.	N.P.
10	0.113	0.07	3060	0.098	0.03	2680
50	0.221	0.19	2070	0.218	0.16	2060
100	0.243	0.25	1390	0.243	0.22	1480
400	0.255	0.48	730	0.259	0.49	730
1600	0.259	1.37	560	0.263	1.19	570
6400	0.260	4.66	530	0.265	4.32	530
25600	0.260	18.66	520	0.265	16.61	520
102400	0.261	75.34	520	0.265	65.83	520

Table 2: The ratio of blocking time of conveyor 1 for model M_1 under policies S_1 and S_2.

We observe that the values seem to become stable when $T > 6400$. For short horizons, the sample variance is high, necessitating a large number of generated paths. The sample variance then decreases when the simulation horizon increases, and becomes constant when the situation seems to stabilise. The blocking probability for conveyor 1 is slightly smaller under policy S_1 than under S_2.

In Table 3, we present the expected number of products in the system with the thresholds $l_1 = 3, l_2 = 3$. We note that the expected number of products in the system is greater under policy S_2.Thus the mean production time for a product is better in the long term under policy S_1.

In the next experiments, we study the impact of increasing the machine service rates. Indeed, when the response time constraints are not met for a given FMS architecture, one solution may be the replacement of one or both machines with more efficient ones. For this purpose, we evaluate different configurations in order to determine the most suitable one. In particular, we consider configurations with a fixed total service rate which is 1.5 times the original total service rate (used in table 3). In the asymmetric case, i.e. ϕ_2(asy), the service rate of the second machine is set to twice the rate of the first machine while the service rate is not modified for the first machine. In

T	S_1			S_2		
	ϕ_2	S.T.	N.P.	ϕ_2	S.T.	N.P.
10	5.299	1.51	109120	5.354	1.35	113040
50	7.601	34.29	559050	8.478	44.41	842390
100	8.085	67.85	553750	9.185	93.19	875700
400	8.471	116.24	236130	9.756	166.78	388840
1600	8.569	133.71	67770	9.905	187.82	109350
6400	8.596	139.13	17550	9.943	199.72	29020
25600	8.599	136.18	4290	9.950	196.08	7090
102400	8.602	152.78	1190	9.951	208.64	1870

Table 3: The expected number of products in the system for model M_1 under policies S_1 and S_2.

the symmetric case, i.e. ϕ_2(sym), service rates of both machines are the same and they are increased by 1.5 times the original value. In table 4 we give results of both policies for ϕ_2(asy) with different threshold values (l_1, l_2) for S_1, and for ϕ_2(sym) with $l_1 = 3$, $l_2 = 3$ for S_1.

T	$\phi_2(asy)$								$\phi_2(sym)$	
	S_2	S_1							S_2	S_1
		(3,5)	(3,4)	(3,3)	(2,3)	(1,3)	(1,4)	(1,5)		(3,3)
10	4.889	4.748	4.764	4.785	4.689	4.624	4.590	4.611	4.149	4.110
50	6.626	5.817	5.835	5.928	5.762	5.695	5.587	5.576	5.011	4.762
100	6.892	5.964	5.994	6.108	5.888	5.827	5.723	5.722	5.198	4.855
400	7.087	6.074	6.107	6.226	5.998	5.958	5.831	5.841	5.415	4.925
1600	7.133	6.079	6.124	6.236	6.030	5.977	5.848	5.863	5.503	4.950
6400	7.141	6.101	6.115	6.239	6.050	5.983	5.861	5.860	5.525	4.956
25600	7.153	6.103	6.133	6.251	6.044	5.982	5.859	5.860	5.527	4.955
102400	7.154	6.098	6.129	6.259	6.042	5.987	5.861	5.864	5.518	4.958

Table 4: The expected number of products in the system for model M_1 with asymmetric service rates $\phi_2(asy)$, different thresholds for policy S_1, and symmetric service rates.

By comparing results in Tables 3, and 4, we observe that increasing the service rate reduces significantly the mean number of products in the system. We observe that symmetric increase gives better results than asymmetric increase for both policies. Furthermore, policy S_1 is better than policy S_2 for all experiences for the expected number of products in the system. The best performance is reached by symmetric increase under policy S_1 with threshold $(l_1, l_2) = (3, 3)$. For asymmetric configuration of the $S1$ policy, asymmetric thresholds $(l_1, l_2) = (1, 4)$ and $(l_1, l_2) = (1, 5)$ seem to provide better results. Such results may be useful during the cost-contribution analysis of FMS designing. In the case we consider here, for example, the designer knows that investing on a single twice-faster machine (i.e. asymmetric configuration) is, performance-wise, less convenient as investing on a pair of 50% faster machines (i.e. symmetric configuration). Thus he/she can opt for either possibility based on machine costs.

In remaining tables, we compare models M_1 and M_2. Failures occur according to an *Erlang* distribution of 4 stages of exponential distribution with mean value 1000, *Erlang*$(4, 1000)$, while repair time follows a uniform distribution *Unif*$(30, 50)$. Thus the mean time to failure is 4000 while mean repair time is 40 time units. First we report again in Table 5 the experiences of Tables 3 for model M_1 and we give the results for model M_2. In short horizons, models M_1 and M_2 have similar behaviours.

As expected, in large horizons, the number of products in the system significantly increases due to failures under both policies. However policy S_1 performs better than policy S_2 in model M_2.

T	M_1 (no faults)				M_2 (faults)			
	S_1		S_2		S_1		S_2	
	ϕ_2	S.T.	ϕ_2	S.T.	ϕ_2	S.T.	ϕ_2	S.T.
10	5.299	1.51	5.354	1.35	5.317	0.02	5.350	0.02
50	7.601	34.29	8.478	44.41	7.620	0.44	8.499	0.49
100	8.085	67.85	9.185	93.19	8.655	2.39	9.885	2.99
400	8.471	116.24	9.756	166.78	21.616	106.91	26.929	118.43
1600	8.569	133.71	9.905	187.82	33.936	738.59	42.118	748.87
6400	8.596	139.13	9.943	199.72	38.691	1724.47	47.796	1737.40
25600	8.599	136.18	9.950	196.08	39.950	2026.30	49.279	2104.75
102400	8.602	152.78	9.951	208.64	40.286	2193.67	49.701	2205.40

Table 5: The expected number of products in the system for models M_1 and M_2 with failures/repairs under policies S_1 and S_2.

In Table 6, we present results for property ϕ_3 with the required number of productions $K = 95$ during each time interval $D = 50$, and with a confidence interval width 0.005 for model M_1 and model M_2 with failures. We observe that in the case when the FMS is subject to failures, the probability of having at least 95 productions during time interval $D = 50$ significantly decreases. Moreover for both models M_1 and M_2, the policies S_1 and S_2 provide similar performances for this property contrasting with the different expected number of products in the system (ϕ_2) under these policies.

T	M_1 (no faults)				M_2 (faults)			
	S_1		S_2		S_1		S_2	
	ϕ_3	S.T.	ϕ_3	S.T.	ϕ_3	S.T.	ϕ_3	S.T.
50	0.046	0.841	0.020	0.321	0.047	0.893	0.021	0.367
100	0.479	1.108	0.449	1.035	0.423	1.869	0.395	1.522
400	0.807	1.203	0.771	1.361	0.578	1.999	0.553	2.034
1600	0.886	2.029	0.852	1.970	0.633	2.133	0.622	2.091
6400	0.908	5.239	0.872	5.078	0.651	5.289	0.641	4.858
25600	0.912	19.340	0.877	17.508	0.654	18.280	0.646	16.446
102400	0.914	77.016	0.878	69.205	0.655	72.776	0.648	65.585

Table 6: The probability to complete at least $K = 95$ productions during a time interval $D = 50$ under both policies, for model M_1 and M_2.

5.2. Analysis of oscillations in a model of the circadian clock

Many real-life systems are characterised by oscillatory dynamics, i.e. their evolution follows a periodic trend. Given a stochastic oscillator, a relevant issue is to be able to assess its basic oscillatory characteristics such as for example how long (on average) a period lasts for, or also how *regular* the exhibited periods are. Here we consider a model of biological oscillator known as circadian-clock, and we demonstrate how HASL can effectively be used to estimate its periodic character. Model checking based analysis of stochastic oscillators has been considered using different types of temporal logic formalisms and tools [6, 15, 45, 27]. Based on Spieler's approach [45] in which timed-automata *monitors* are used to assess the period duration of a CTMC oscillator,

here we show how, through LHA *monitors*, we can extend the capability of analysing oscillator by not only measuring the period duration but also its fluctuation.

5.2.1. A stochastic model of the circadian clock

Circadian clocks are biological mechanisms responsible for keeping track of daily cycles of light and darkness. The cycling behaviour involves a network of biochemical species which exhibit a periodic *signal*. Here we consider a simple model (presented in [46]) of a circadian network consisting of chemical equations (2). The model involves 2 genes, geneA and geneR, expressing the *activator* protein A, respectively the *repressor* protein R. Both genes can be in either of two (mutually exclusive) states: 1) having the promoter region free (represented by species D_A, respectively D_R); 2) having a molecule of activator A attached to the promoter region (represented by species D'_A, respectively D'_R). The model accounts for *protein expression*, i.e. a two steps process in which first the gene *transcribes* an mRNA molecule, M_A (resp. M_R), which is then *translated* into the target protein, A (resp. R).

Network reactions. mRNA *transcription* is modelled by reactions R_5, R_6 (for geneA) and R_7, R_8 (for geneR). Notice that transcription happens at different speed depending on the state of the gene. If a gene's promoter is *free* then transcription (i.e R_6, R_8) is slower (see rates in Table 7), if a molecule of activator A is attached to a gene's promoter through R_1 (resp. R_3) transcription (i.e R_5, R_7) is faster. The remaining reactions are as follows: mRNA *translation* corresponds to R_9 (resp. R_{10}); the *complexation* of protein A and R is modelled by R_{11}, while R_{12} models its "asymmetric" reverse, i.e. the degradation of A while complexed and consequent release of R (only); simple *degradation* of each species is modelled by R_{13}, R_{14}, R_{15} and R_{16}. The (continuous) kinetic rate constants [7] (taken from [46]) are given in Table 7.

$$
\begin{array}{lll}
R_1 : A + D_A \xrightarrow{\gamma_A} D'_A & R_2 : D'_A \xrightarrow{\theta_A} A + D_A & R_3 : A + D_R \xrightarrow{\gamma_R} D'_R \\
R_4 : D'_R \xrightarrow{\theta_R} D_R + A & R_5 : D'_A \xrightarrow{\alpha'_A} M_A + D'_A & R_6 : D_A \xrightarrow{\alpha_A} M_A + D_A \\
R_7 : D'_R \xrightarrow{\alpha'_R} M_R + D'_R & R_8 : D_R \xrightarrow{\alpha_R} M_R + D_R & R_9 : M_A \xrightarrow{\beta_A} M_A + A \\
R_{10} : M_R \xrightarrow{\beta_R} M_R + R & R_{11} : A + R \xrightarrow{\gamma_C} C & R_{12} : C \xrightarrow{\delta_A} R \\
R_{13} : A \xrightarrow{\delta_A} \emptyset & R_{14} : R \xrightarrow{\delta_R} \emptyset & R_{15} : M_A \xrightarrow{\delta_{M_A}} \emptyset \\
R_{16} : M_R \xrightarrow{\delta_{M_R}} \emptyset & &
\end{array}
\tag{2}
$$

Stochastic model. Chemical equations (2) can give rise to either a system of ODEs (in the continuous-deterministic semantics) or to a stochastic process (in the discrete-stochastic semantics). Here we focus on the discrete-stochastic semantics: Figure 9

[7] Notice that rates of first order reactions are expressed in h^{-1} units whereas rates of second order reactions are expressed in $mol^{-1}h^{-1}$

α_A	$50\ h^{-1}$	α_R	$0.01\ h^{-1}$	δ_A	$1\ h^{-1}$	δ_R	$0.2\ h^{-1}$
$\alpha_{A'}$	$500\ h^{-1}$	$\alpha_{R'}$	$50\ h^{-1}$	$\gamma_A = \gamma_R$	$1\ mol^{-1}h^{-1}$	γ_C	$2\ mol^{-1}h^{-1}$
β_A	$50\ h^{-1}$	β_R	$5\ h^{-1}$	θ_A	$50\ h^{-1}$	θ_R	$100\ h^{-1}$
δ_{MA}	$10\ h^{-1}$	δ_{MR}	$0.5\ h^{-1}$				

Table 7: Reactions' rates for the circadian oscillator

shows the GSPN encoding of equations (2) developed with COSMOS. The configuration of the GSPN requires setting the initial population and the rates of each transition. Following [46], we consider a model with a single copy of each gene, and we further assume that the promoter region of both genes is initially free. Thus initially we set $D_A = D_R = 1$ and $D'_A = D'_R = 0$. Observe that a gene can only be in either of two states (e.g. D_A or D'_A) hence the following invariant conditions must hold: $D_A + D'_A = 1$ and $D_R + D'_R = 1$. The remaining species are initially supposed to be inexistent, hence they are initialised to 0. Concerning the conversion of the reaction rate constants, for simplicity we assume a unitary volume of the system under consideration, hence all continuous rates in Table 7 can be used straightforwardly as rates of the corresponding discrete-stochastic reactions. In this case we assume all reactions following a negative exponential law thus yielding a CTMC model. Furthermore, we assume all reactions following the *mass action law* (i.e. the kinetic rate of a reaction is given by the product of the rate constant with the population of each reactant), which in GSPN terms means all transitions in Figure 9 are associated with (single-server) marking-dependent exponential distributions whose rate constants correspond to those in Table 7.

We stress that the resulting CTMC has nine dimensions with four bounded dimensions, i.e., $D_A, D'_A, D_R, D'_R \in \{0, 1\}$, and five unbounded dimensions $M_A, A, M_R, R, C \in \mathbb{N}$, therefore its state space is $S \subseteq \{0, 1\}^4 \times \mathbb{N}^5$. Furthermore the CTMC can be proved to be ergodic, that is, each of its states has a finite mean recurrence time, which means that a drift towards infinity is not possible. As a consequence the HASL specified random variables we consider later on for assessing the oscillatory characteristic of the circadian clock CTMC have a well defined mean value.
.

The oscillatory dynamics of the circadian clock network can be observed by looking at some trajectories. Figure 10 depicts examples of realisations of the activator A (red-plot) and repressor R (blue-plot) dynamics along a simulated trace of the GSPN model of Figure 9. The left plots correspond to the original rates (i.e. $\delta_R = 0.2$) for which the exhibited period duration is approximatively 24, while the right plots correspond to a 10 times speed-up of the repressor degradation (i.e. $\delta_R = 2$) which induces a more than halved period duration of approximatively 11. In the remainder we formally assess various measures related to the period of oscillations, including the mean value of the period duration.

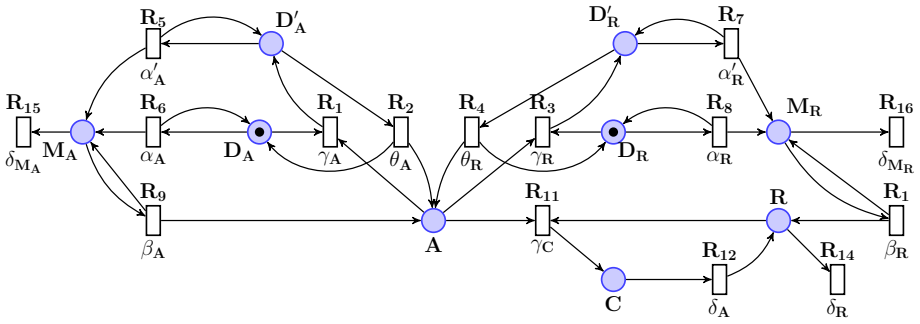


Figure 9: GSPN encoding of the system (2) of chemical equations corresponding to the circadian-clock.

Figure 10: Single trajectory showing the oscillatory character of activator A and repressor B dynamics with normal repressor's degradation rate $\delta_R = 0.2$ (left) and with $10\times$ speed-up, i.e. $\delta_R = 2$ (right).

5.2.2. Measuring the noisy-period of stochastic oscillations with HASL

HASL-based analysis of oscillations relies on the concept of *noisy periodicity* [45], i.e. a weaker characterisation of periodicity of a function. A function $f : \mathbb{R}^+ \to \mathbb{R}$ is said noisy periodic with respect to given thresholds $L < H \in \mathbb{R}^+$ (inducing the partition of $\mathbb{R}^+$ into $low : [0, L)$, $mid : [L, H)$, $high : [H, +\infty)$) if f perpetually switches from low to $high$ and back to low. Such a generic characterisation extends naturally to traces of a DESP as illustrated in Figure 11).

Figure 11: Example of *noisy periodic* trace corresponding to a given ($L < H$) partition of a DESP state space.

A (noisy) period corresponds to the time interval between two consecutive crossing points corresponding to entering in one extreme of the partition, interleaved by at least one crossing point into the opposite extreme. For example in Figure 11 the first period p_1 corresponds to the duration between the first mid-to-low crossing and the successive mid-to-low crossing interleaved by a mid-to-$high$ crossing. Notice that the first complete period (p_1) might be preceded by a spurious period (i.e. p_0) which should be discarded as there's no guarantee that $T = 0$ corresponds with an actual mid-to-low crossing, hence with the actual beginning of p_0.

The $\mathcal{A}_{per}$ automaton. We introduce an LHA (Figure 12), denoted $\mathcal{A}_{per}$, designed for detecting realisations of noisy periods of a given observed species (in this case A). More specifically, $\mathcal{A}_{per}$ is designed for assessing two characteristics of an oscillatory trace: the mean period duration (denoted $\bar{t}_p$) and the period fluctuation (denoted $s^2_{t_p}$) over the first N periods detected along a trace. Period fluctuation represents how much (on average) a single period realisation deviates from the mean duration computed

name	domain	update definition
t	$\mathbb{R}_{\geq 0}$	*reset*
n	$\mathbb{N}$	*increment*
top	bool	*complement*
t_p	$\mathbb{R}_{\geq 0}$	*reset*
$\bar{t}_p$	$\mathbb{R}_{\geq 0}$	$f(\bar{t}_p, t_p, n) = \frac{1}{n+1}(\bar{t}_{p_n} \cdot n + t_p)$
$s^2_{t_p}$	$\mathbb{R}_{\geq 0}$	$g(s^2_{t_p}, \bar{t}_p, t_p, n) = \frac{1}{n}[(n-1)s^2_{t_p} + (t_p - \bar{t}_p)(t_p - f(\bar{t}_p, t_p, n+1))]$

Table 8: The data variables of automata $\mathcal{A}_{per}$ of Figure 12 for measures of noisy-periodicity

over the periods observed along a trace. From the point of view of analysis the period fluctuation is a useful indication of the regularity of the observed oscillator. $\mathcal{A}_{per}$ consists of an initial *transient filter* (locations l_0, l'_0) plus three main locations **low**, **mid** and **high** (corresponding to the partition of A's domain induced by thresholds $L < H$). Initially the simulated trajectory unfolds for a given duration ($initT$) letting $\mathcal{A}_{per}$ within l_0, l'_0 without doing anything.[8] After $initT$ time units $\mathcal{A}_{per}$ enters **low**[9] where the actual oscillation analysis begins.



Figure 12: An LHA for selecting noisy periodic traces wrt. partition low=$(-\infty, L]$, mid=(L, H) and high=$[H - \infty)$.

From **low** the automaton follows the profile of A hence moving to **mid** as soon as $L < A < H$ holds, and then to **high** as soon as $A \geq H$. With $\mathcal{A}_{per}$ a period starting point is associated with the first mid-to-low crossing that follows a mid-to-$high$ crossing [10]. Hence the first detected period (crossing from low to $high$ and back to low) is discarded as its duration may be spurious. $\mathcal{A}_{per}$ uses six variables (Table 8): t is a timer that keeps track of simulation time; n counts the number of detected noisy-periods while top is a boolean flag used for distinguishing the **mid** to **low** crossing points that correspond to the closure of one period (i.e. when a traversal from mid-to-low has been preceded by a traversal from $high$-to-mid) from those who do not. Notice that, in order to ignore the first potentially spurious period (p_0 in Figure 11), n is initially set to $n := -1$ on entering **low** from the initial transient filter, and the simulation time t is then reset on detection of the closure of the first spurious detected (i.e. on entering **low** from **mid** when $n = -1$). Furthermore t_p stores the duration of the last detected period (t_p), while $\overline{t_p}$ maintain the mean duration of all (so far) detected periods and $s^2_{t_p}$ stores the fluctuation of the period duration for all (already) detected periods. Notice that the

[8] This is useful for eliminating the effect of the *initial transient* from long run measures as already discussed in [4].

[9] the choice of beginning measuring in **low** is arbitrary, equivalently $\mathcal{A}_{per}$ can be defined so that analysis starts in either **mid** or **high**.

[10] again equivalent versions of $\mathcal{A}_{per}$ can be easily obtained which detect periods by considering different starting points, e.g period that starts with a crossing from mid to $high$.

fluctuation (i.e. $s^2_{t_p}$) is computed *on the fly* (see Table 8) by adaptation of the so-called *online algorithm* for computing the variance out of a sample of observations. Finally the analysis of simulated trajectories stops (by entering the accepting location **end**) as soon as the N^{th} period has been detected. For $\mathcal{A}_{per}$ we consider the following target measures:

- $Z_1 \equiv \mathbb{E}[LAST(t)/N]$: corresponding to the mean value of the period duration for the first N detected periods.
- $Z_2 \equiv PDF(LAST(t)/N, s, l, h)$: corresponding to the PDF of the period duration over the first N detected periods, where $[l, h]$ represents the considered support of the estimated PDF, and $[l, h]$ is discretised into uniform subintervals of width s
- $Z_3 \equiv \mathbb{E}[LAST(s^2_{t_p})]$: corresponding to the fluctuation of the noisy-period duration.

Results. We run a number of experiments for assessing the influence of the repressor degradation rate (δ_R) on the period of the circadian oscillator. Figure 13(a) shows three plots representing the PDF of the period obtained through evaluation of formula Z_2 for three values of δ_R (i.e. 0.1, 0.2 and 2). With $\delta_R = 0.2$ (i.e. the original value as in [46]) the PDF is centered at $t = 24.9$, i.e. slightly more than the expected 24 hours duration for a normally functioning circadian clock. Speeding up by a factor 10 the repressor degradation (i.e. $\delta_R = 2$) yields a slightly more than halved oscillation period (i.e. PDF centered at $T = 10.8$). Finally slowing down the degradation rate of a half (i.e. $\delta_R = 0.1$) yields a less than doubled oscillation period (i.e. PDF centered at $T = 40.7$).

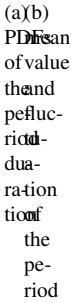


Figure 13: PDFs (left) and mean value v. fluctuation (right) of the period of the circadian clock model in function of the repressor degradation rate.

Figure 13(b) compares the measured mean value of the period (red plot, computed with Z_1) with the period fluctuation (blue plot, computed with Z_3). Such plots are in agreement with the PDF plots of Figure 13(a) and confirm the general outcome of this analysis which is: slowing down the degradation of the repressor results in increasing the length of the period as well as its irregularity (i.e. the period fluctuation also increases). In fact the period fluctuation (blue plot Figure 13(b)) decreases

with the increase of δ_R which means that a slower degradation of the repressor corresponds to an increase in the irregularity of the periods. This is in agreement with the PDF plots of Figure 13(a) as the width of the PDF bell-shaped curves increases with the increase of δ_R). All plots in Figure 13 result from sampling of finite trajectories consisting of $N = 100$ periods, where periods have been detected using $L = 1$ and $H = 1000$ as partition thresholds, and target estimates have been computed with 99% confidence and confidence-interval width of 0.01. The PDF plots in Figure 13(a) have been computed using a discretisation of the period support interval $[0, 50]$ into subintervals of width 0.1. Finally, in order to assess the effect that the initial transient of the circadian clock model may have on the outcome of measuring, we repeated all of the above discussed experiments with different values of the $initT$ parameter (e.g. $initT \in \{1, 10, 50, 100, 500, 1000, \ldots\}$) which determines the starting measuring point for $\mathcal{A}_{per}$. The outcomes of repeated experiments turned out to be essentially the same and hence independent of the chosen $initT$ value, indicating that initialisation period for the circadian clock is quite short.

6. Conclusion

We have presented a new framework for expressing elaborate properties related to stochastic processes. A formula of HASL returns either a probability (as the previous approaches do) or a conditional expectation whose condition is based on acceptance by a linear hybrid automaton. Such a framework can be used both for probabilistic validation of functional properties or for elaborate performance analysis. We have developed a tool COSMOS and we have experimentally validated it on Flexible Manufacturing Systems and biological case studies, thus illustrating the feasibility of this statistical based approach.

While the empirical efficiency has been established, we aim at overcoming the well-known limitations of the statistical approach. In a recent work [16], an importance sampling method has been designed and implemented in COSMOS accelerating the path generation when faced to difficult acceptance condition related to a rare event. Another research direction consists in analysing the structure of the DESP in order to circumvent the constraint that almost surely a path is accepted or rejected by the LHA.

References

[1] Ajmone Marsan, M., Balbo, G., Conte, G., Donatelli, S., Franceschinis, G.: Modelling with Generalized Stochastic Petri Nets. John Wiley & Sons (1995)

[2] Alur, R., Courcoubetis, C., Dill, D.: Model-checking for probabilistic real-time systems. In: ICALP'91, LNCS 510, pp. 115–126 (1991)

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

[4] Amparore, E.G., Ballarini, P., Beccuti, M., Donatelli, S., Franceschinis, G.: Expressing and computing passage time measures of gspn models with hasl. In: J.M. Colom, J. Desel (eds.) Petri Nets, *Lecture Notes in Computer Science*, vol. 7927, pp. 110–129. Springer (2013)

[5] Amparore, E.G., Barbot, B., Beccuti, M., Donatelli, S., Franceschinis, G.: Simulation-based verification of hybrid automata stochastic logic formulas for stochastic symmetric nets. In: G.A. Wainer (ed.) Proceedings of the 1st ACM SIGSIM Conference on Principles of Advanced Discrete Simulation (PADS'13), pp. 253–264. ACM Press, Montreal, Canada (2013)

[6] Andrei, O., Calder, M.: Trend-based analysis of a population model of the akap scaffold protein. T. Comp. Sys. Biology **14**, 1–25 (2012)

[7] Aziz, A., Sanwal, K., Singhal, V., Brayton, R.: Model-checking CTMCs. ACM Trans. on Computational Logic **1**(1), 162–170 (2000)

[8] Baier, C., Cloth, L., Haverkort, B., Kuntz, M., Siegle, M.: Model checking action- and state-labelled Markov chains. IEEE Trans. on Software Eng. **33**(4), 701–710 (2007)

[9] Baier, C., Haverkort, B., Hermanns, H., Katoen, J.P.: Model-checking algorithms for CTMCs. IEEE Trans. on Software Eng. **29**(6), 524–541 (2003)

[10] Baier, C., Haverkort, B.R., Hermanns, H., Katoen, J.P.: On the logical characterisation of performability properties. In: ICALP'00, LNCS 1853, pp. 780–792 (2000)

[11] Ballarini, P., Djafri, H., Duflot, M., Haddad, S., Pekergin, N.: COSMOS: a statistical model checker for the hybrid automata stochastic logic. In: Proc. QEST'11, pp. 143–144. IEEE Computer Society Press (2011)

[12] Ballarini, P., Djafri, H., Duflot, M., Haddad, S., Pekergin, N.: HASL : an expressive language for statistical verification of stochastic models. In: Proc. Valuetools'2011, pp. 306–315 (2011)

[13] Ballarini, P., Djafri, H., Duflot, M., Haddad, S., Pekergin, N.: HASL: An expressive language for statistical verification of stochastic models. In: P.H. Samson Lasaulce Dieter Fiems, L. Vandendorpe (eds.) Proceedings of the 5th International Conference on Performance Evaluation Methodologies and Tools (VALUETOOLS'11), pp. 306–315. Cachan, France (2011)

[14] Ballarini, P., Djafri, H., Duflot, M., Haddad, S., Pekergin, N.: Petri nets compositional modeling and verification of flexible manufacturing systems. In: CASE, pp. 588–593. IEEE (2011)

[15] Ballarini, P., Guerriero, M.: Query-based verification of qualitative trends and oscillations in biochemical systems. Theoretical Computer Science **411**(20), 2019 – 2036 (2010). DOI 10.1016/j.tcs.2010.02.010. URL http://www.sciencedirect.com/science/article/pii/S0304397510001052

[16] Barbot, B., Haddad, S., Picaronny, C.: Coupling and importance sampling for statistical model checking. In: C. Flanagan, B. König (eds.) Proc. TACAS'12, *LNCS*, vol. 7214, pp. 331–346. Springer, Tallinn, Estonia (2012)

[17] Bulychev, P.E., David, A., Larsen, K.G., Mikucionis, M., Poulsen, D.B., Legay, A., Wang, Z.: Uppaal-smc: Statistical model checking for priced timed automata. In: Proceedings 10th Workshop on Quantitative Aspects of Programming Languages and Systems, *EPTCS*, vol. 85, pp. 1–16 (2012)

[18] Bulychev, P.E., David, A., Larsen, K.G., Mikucionis, M., Poulsen, D.B., Legay, A., Wang, Z.: Uppaal-smc: Statistical model checking for priced timed automata. In: H. Wiklicky, M. Massink (eds.) QAPL, vol. 85, pp. 1–16 (2012)

[19] Buzacoott, J.A., Shantikumar, J.G.: Stochastic Models of Manufacturing Systems. Prentice-Hall (1993)

[20] Canny, J.: Some algebraic and geometric computations in PSPACE. In: Proc. STOC'88, pp. 460–467 (1988)

[21] Chen, T., Diciolla Marco an Kwiatkowska, M., Mereacre, A.: Time-bounded verification of ctmcs against real-time specifications. In: 9th International Conference, FORMATS 2011, Lecture Notes in Computer Science, pp. 26–42. Springer (2011)

[22] Chen, T., Han, T., Katoen, J.P., Mereacre, A.: Quantitative model checking of CTMC against timed automata specifications. In: Proc. LICS'09, pp. 309–318 (2009)

[23] Chiola, G., Dutheillet, C., Franceschinis, G., Haddad, S.: Stochastic well-formed colored nets and symmetric modeling applications. IEEE Transactions on Computers **42**(11), 1343–1360 (1993). URL http://www.lsv.ens-cachan.fr/Publis/PAPERS/PS/CDFH-toc93.ps

[24] Chow, Y.S., Robbins, H.: On the asymptotic theory of fixed-width sequential confidence intervals for the mean. Annals of Mathematical Statistics **36**(2), 457–462 (1965)

[25] Clopper, C.J., Pearson, E.S.: The use of confidence or fiducial limits illustrated in the case of the binomial. Biometrika **26**(26), 404–413 (1934)

[26] CosyVerif home page. http://www.cosyverif.org

[27] David, A., Larsen, K.G., Legay, A., Mikucionis, M., Poulsen, D.B., Sedwards, S.: Runtime verification of biological systems. In: ISoLA (1), pp. 388–404 (2012)

[28] Donatelli, S., Haddad, S., Sproston, J.: Model checking timed and stochastic properties with CSL^{TA}. IEEE Trans. on Software Eng. **35**, 224–240 (2009)

[29] Emerson, E.A., Clarke, E.M.: Characterizing correctness properties of parallel programs using fixpoints. In: Proc. of ICALP'80, LNCS 85, pp. 169–181 (1980)

[30] Glynn, P.W.: On the role of generalized semi-Markov processes in simulation output analysis. In: Proc. Conf. Winter simulation, pp. 38–42 (1983)

[31] Gorrieri, R., Herzog, U., Hillston, J.: Unified specification and performance evaluation using stochastic process algebras. Perform. Eval. **50**(2/3), 79–82 (2002)

[32] He, R., Jennings, P., Basu, S., Ghosh, A.P., Wu, H.: A bounded statistical approach for model checking of unbounded until properties. In: Proc. ASE'10, pp. 225–234 (2010)

[33] Heiner, M., Rohr, C., Schwarick, M.: MARCIE - Model checking And Reachability analysis done effiCIEntly. In: J. Colom, J. Desel (eds.) Proc. PETRI NETS 2013, *LNCS*, vol. 7927, pp. 389–399. Springer (2013)

[34] Herault, T., Lassaigne, R., Peyronnet, S.: APMC 3.0: Approximate verification of discrete and continuous time Markov chains. In: Proc. QEST'06, pp. 129–130 (2006)

[35] Hoeffding, W.: Probability inequalities for sums of bounded random variables. Journal of the American Statistical Association **58**(301), pp. 13–30 (1963)

[36] Jégourel, C., Legay, A., Sedwards, S.: A platform for high performance statistical model checking - plasma. In: TACAS, *Lecture Notes in Computer Science*, vol. 7214, pp. 498–503 (2012)

[37] Jégourel, C., Legay, A., Sedwards, S.: A platform for high performance statistical model checking - plasma. In: C. Flanagan, B. König (eds.) TACAS, vol. 7214, pp. 498–503. Springer (2012)

[38] Katoen, J.P., Zapreev, I.S., Hahn, E.M., Hermanns, H., Jansen, D.N.: The ins and outs of the probabilistic model checker MRMC. QEST pp. 167–176 (2009)

[39] Kwiatkowska, M., Norman, G., Parker, D.: PRISM 4.0: Verification of probabilistic real-time systems. In: G. Gopalakrishnan, S. Qadeer (eds.) Proc. 23rd International Conference on Computer Aided Verification (CAV'11), *LNCS*, vol. 6806, pp. 585–591. Springer (2011)

[40] L.Cloth, Katoen, J.P., Khattri, M., Pulungan, R.: Model checking Markov reward models with impulse rewards. In: Proc. DSN'05 (2005)

[41] Narahari, Y., Viswanadham, N.: Transient analysis of manufacturing systems performance. IEEE Trans. on Robotics and Automation **10 (2)**, 230–244 (1994)

[42] PRISM home page. http://www.prismmodelchecker.org

[43] Sen, K., Viswanathan, M., Agha, G.: On statistical model checking of stochastic systems. In: CAV, *Lecture Notes in Computer Science*, vol. 3576, pp. 266–280. Springer (2005)

[44] Sen, K., Viswanathan, M., Agha, G.: VESTA: A statistical model-checker and analyzer for probabilistic systems. In: Proc. QEST'05, pp. 245–246 (2005)

[45] Spieler, D.: Characterizing oscillatory and noisy periodic behavior in markov population models. In: Proc. QEST'13 (2013)

[46] Vilar, J., Kueh, H.Y., Barkai, N., Leibler, S.: Mechanisms of noise-resistance in genetic oscillators. Proc. National Academy of Sciences of the United States of America **99**(9), 5988–5992 (2002)

[47] Wald, A.: Sequential tests of statistical hypotheses. The Annals of Mathematical Statistics **16**(2), 117–186 (1945)

[48] Younes, H., Simmons, R.: Statistical probabilistic model checking with a focus on time-bounded properties. Inf. Comput. **204**(9), 1368–1409 (2006)

[49] Younes, H.L.: Ymer: A statistical model checker. In: Computer Aided Verification, pp. 429–433. Springer (2005)

[50] Zuliani Paolo an Platzer, A., Clarke, E.M.: Bayesian statistical model checking with application to stateflow/simulink verification. Formal Methods in System Design **43**(2), 338–367 (2013)