HASL: an Expressive Language for Statistical Verification of Stochastic Models*

Paolo Ballarini
Université Paris Est
Créteil, France
paolo.ballarini@u-pec.fr

Hilal Djafri
LSV, ENS
Cachan, France
djafri@lsv.ens-cachan.fr

Marie Duflot
Université Paris Est
Créteil, France
duflot@u-pec.fr

Serge Haddad
LSV, ENS
Cachan, France
haddad@lsv.ens-cachan.fr

Nihal Pekergin
Université Paris Est
Créteil, France
nihal.pekergin@u-pec.fr

ABSTRACT

We introduce the Hybrid Automata Stochastic Logic (HASL), a new temporal logic formalism for the verification of discrete event stochastic processes (DESP). HASL employs Linear Hybrid Automata (LHA) as machineries to select prefixes of relevant execution paths of a DESP $\mathcal{D}$. The advantage with LHA is that rather elaborate information can be collected *on-the-fly* during path selection, providing the user with a powerful means to express sophisticated measures. A formula of HASL consists of an LHA $\mathcal{A}$ and an expression Z referring to moments of *path random variables*. A simulation-based statistical engine is employed to obtained a confidence-interval estimate of the expected value of Z. In essence HASL provide a unifying verification framework where sophisticated temporal reasoning is naturally blended with elaborate reward-based analysis. We illustrate the HASL approach by means of some examples and a discussion about its expressivity. We also provide empirical evidence obtained through COSMOS, a prototype software tool for HASL verification.

1. INTRODUCTION

From model checking to quantitative model checking. Since its introduction [EC80], 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 a decision procedure has been designed based on transient analysis of Markov chains [BHHK03].

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 sampling of distributions. Unfortunately both the formula associated with an elaborated property and the stochastic process associated with a real application make rare the possibility of such pattern. In these cases, statistical model checking [YS06] is thus an alternative to numerical techniques. Roughly speaking, statistical model checking consists in sampling executions of the system (possibly synchronized with some automata corresponding to the formula to be checked) and comparing the ratio of successful executions with a threshold specified by the formula. The advantage of the 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. However, 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 [ASSB00].

Our contribution. We introduce a new formalism called Hybrid Automaton Stochastic Logic (HASL) which provides a unified framework both for model checking and for performance and dependability evaluation. A HASL formula evaluates to a real number which is defined by the expectation of a path random variable conditioned by the success of the path. The concept of conditional expectation significantly enlarges the expressive power of the logic. The proposed temporal logic is indeed a quantitative logic permitting both to check if probability thresholds are met and to evaluate complex performability measures. 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's states. This automaton will synchronize with the DESP, precisely selecting ac-

*partially supported by French research project ANR-06-SETI-002.

cepting 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 like formalisms as CSL^{TA} [DHS09] and its extension to multi-clocks [CHKM09] 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 [BHHK00]) with a more precise selection of path executions, and the possibility to consider multiple rewards. Therefore HASL makes it possible to consider not only standard performability measures but also complex ones in a generic manner.

A statistical verification tool COSMOS has been developed for this logic. We have chosen generalized stochastic Petri nets (GSPN) as high level formalism for the description of the discrete event stochastic process since (1) it allows a flexible modeling w.r.t. the policies defining the process (choice, service and memory) and (2) due to the locality of net transitions and the simplicity of the firing rule it leads to efficient path generation.

Organization. In section 2 we describe the class of stochastic models we refer to (i.e. DESP). In section 3 we formally introduce the HASL logic and we provide an overview of the related work, where the expressiveness of HASL is compared with that of existing logics. In section 4, we recall the basic principles of statistical model checking, we detail COSMOS, a prototype software tool for HASL verification and report about experimental evidence. Finally, in section 5, we conclude and give some perspectives.

2. DISCRETE EVENT STOCHASTIC PROCESS

We describe the class of stochastic models that are suitable for HASL verification, namely Discrete Event Stochastic Processes (DESP). Such class includes Markov chain models, the (only) type of stochastic models targeted by existing stochastic logics. The definition of DESP we introduce in the following resembles that of generalized semi-Markov processes [Gly83] as well as that given by Alur, Coucourbetis and Dill [ACD91].

Syntax. DESPs are stochastic processes consisting of a (possibly infinite) set of states and whose dynamic is triggered by a set of discrete events. In order to be as general as possible, 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 Discrete Event Stochastic Process (DESP) is a tuple $\mathcal{D} = \langle S, \pi_0, E, Ind, enabled, target, delay, choice\rangle$ where:*

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

Before giving the semantics of a DESP, we informally describe its items. Given a state s, $enabled(s)$ is the set of events enabled in s. For an event $e \in enabled(s)$, $target(s, e)$ denotes the target state reached from s on occurrence of e and $delay(s, e)$ is the distribution of the delay between the enabling of e and its possible occurrence. Furthermore if we denote $E' \subseteq enabled(s)$ the set of events with same earliest delay in some configuration of the process with state s, $choice(s, E')$ describes how the conflict is randomly resolved: for all $e' \in E'$, $choice(s, E')(e')$ is the probability that e' will be selected among E'. 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 characterize the information on the DESP known by the LHA of 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.

Semantics. The following paragraphs give an intuition of the DESP semantics. The formal version can be found in appendix A. In order to define the semantics of this class of DESPs, we consider the following policies: choice is driven by the *race policy* (i.e. the event with the shortest delay occurs first), the service policy is *single server* (at most one instance per event may be scheduled) and the memory policy is the *enabled memory* one (i.e. a scheduled event remains so until executed or until it becomes disabled). Other policies could have been selected (resampling memory, age memory, etc.). We have stuck to the most usual policies for the sake of simplicity.

A *timed execution* of a DESP is an infinite sequence $\sigma = s_0 \xrightarrow{e_0,\tau_0} s_1 \xrightarrow{e_1,\tau_1} \cdots$ where for any i, $s_i \in S$ is the $(i+1)^{th}$ state of the sequence, $e_i \in E$ is the event which corresponds to the state change from s_i to s_{i+1} and $\tau_i \in \mathbb{R}_{\geq 0}$ is the occurrence time of event e_i.

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 $sched : E \to \mathbb{R}^+ \cup \{+\infty\}$ being the function that describes the occurrence time of each scheduled event ($+\infty$ meaning that the event is not yet scheduled). The semantics of DESP can be described as follows: the initial configuration of a DESP is $(s_0, 0, sched_0)$, with $sched_0(e) = +\infty$, for all $e \in E$ and s_0 chosen according to the initial distribution π_0. If $(s, \tau, sched)$ is the current configuration then the execution of a DESP is a succession of steps where each step consists of:

- For all $e \in enabled(s)$ such that $sched(e) = +\infty$ generate δ, a sample from distribution $delay(s, e)$ and set $sched(e) := \tau + \delta$.
- Determine the set $E' = \{e' \in E \mid \forall e \in E, sched(e') \leq sched(e)\}$ of enabled events with minimal schedule.

- Randomly choose in E' the next event e from distribution $choice(s, E')$.
- Execute event e which updates the current configuration of the DESP as follows: $s := target(s, e)$, $\tau := sched(e)$, $sched(e) := +\infty$ and $sched(e') := +\infty$ for all $e' \notin enabled(s)$.

Note that because of their definition the evolution of a DESP is naturally suitable for discrete event simulation. Furthermore observe that several higher-level formalisms commonly used for representing Markov chain models (e.g. Stochastic Petri Nets and/or Stochastic Process Algebras), can straightforwardly be adapted for representation 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).

In the following we describe an example of DESP expressed in terms of a Generalized Stochastic Petri Net (GSPN). Such model will be used in the remainder of the paper: in Section 3, for describing, through a couple of LHA examples, the intuition behind Hybrid Automata based verification; in Section 4 for discussing experimental evidence referred to a case study. Before describing the running example, we informally outline the basis of GSPN modeling (for a formal account we refer the reader to [AMBC$^+$95]), pointing out the differences between "original" GSPNs and GSPNs for representing DESPs (which we refer to as GSPN-DESP).

*GSPN models.*A GSPN model is a bi-partite graph consisting of two classes of nodes, *places* and *transitions*. Places may contain *tokens* (representing the state of the modeled system) while transitions indicate how tokens "flow" within the net (encoding the model's dynamics). The state of a GSPN consists of a *marking* indicating the distribution of tokens throughout the places (i.e. how many tokens each place contains). Roughly speaking a transition is enabled whenever all of its *input places* contains a number of tokens greater 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 all of its input places and producing tokens (in a number indicated by the multiplicity of the corresponding output arcs) in all of its output places. Transitions can be either *timed* (denoted by empty bars, see Figure 1) or *immediate* (denoted by filled-in bars, see Figure 1). Generally speaking transitions are characterized 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. Wtih the GSPN formalism [AMBC$^+$95] the delay of timed transitions is assumed *exponentially* distributed, whereas with GSPN-DESP it can be given by any finite-support distribution (as in the spirit of DESPs). Thus whether a GSPN timed-transition is characterized simply by its weight $t \equiv w$ ($w \in \mathbb{R}^+$ indicating an $Exp(w)$ distributed delay), a GSPN-DESP timed-transition is characterized by a triple: $t \equiv (\text{Dist-t}, \text{Dist-p}, w)$, where Dist-t indicates the type of distribution (e.g. Unif), dist-p indicates the parameters of the distribution (e.g $[\alpha, \beta]$) and $w \in \mathbb{R}^+$ is used to probabilistically choose between transitions occurring with equal delay[1]

[1]a possible condition in case of non-continuous delay distribution

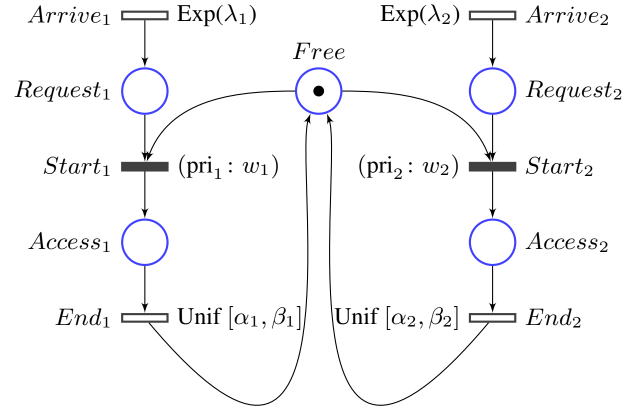

Figure 1: The SPN description of a shared memory system.

Running example. We consider the GSPN model of Figure 1 (inspired by [AMBC$^+$95]). It describes the behavior of an open system where two classes of clients (namely 1 and 2) compete to access a shared memory (resource). Class i-clients ($i \in \{1, 2\}$) enter the system according to a Poisson process with parameters λ_i (corresponding to the exponentially distributed timed transition $Arrive_i \sim Exp(\lambda_i)$). On arrival, clients cumulate in places $Request_i$ where they wait for the memory to be free (memory idle is represented by the presence of a token in place $Free$). The exclusive access to the shared memory 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 $Request_1$ and $Request_2$) a class i process wins the competition with a class $j = (i)mod(2)+1$ process with probability $Pr(win_i) = 1$ if pri$_i$ > pri$_j$, and with probability $Pr(win_i) = w_i/(w_i + w_j)$ if pri$_i$ = pri$_j$. The occupation time of the memory by a class i client is assumed to be Uniformly distributed within the interval $[\alpha_i, \beta_i]$ (corresponding to the Uniformly distributed timed transitions $End_i \sim Unif(\alpha_i, \beta_i)$). Thus on on firing of transition End_i the memory is released and a class i client leaves the system. Observe that all timed-transitions in this example have continuous distributions, so weights and priorities are not necessary for them since the probability that two transitions have identical schedule is null.

	Timed-transitions		immediate transitions	
	Arrival (Exponential)	End (Uniform)	Start priority	Start weight
class-1 Processes	λ_1	$[\alpha_1, \beta_1]$	π_1	w_1
class-2 Processes	λ_2	$[\alpha_2, \beta_2]$	π_2	w_2

Table 1: parameters of the shared-memory model

Table 1 summarizes the parameters of the GSPN model of Figure 1. These will be referred to in later sections while presenting a number of experiments performed on the shared memory model by means of the Cosmos tool.

3. HYBRID AUTOMATON STOCHASTIC LOGIC

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 synchronizes with an infinite timed execution of the considered DESP until some final state is reached or the synchronization fails. During this synchronization, some data variables evolve and also condition the evolution of this synchronization.
- The second component of a formula is an expression whose operands are mainly data variables and whose operators will be described formally later in this section. In order to express path indices, they include path operators such as min and max value along an execution, value at the end of a path, integral over time and the average value operator. Conditional expectations are applied to these indices in order to obtain the value of the formula.

3.1 Synchronized Linear Hybrid Automata

Syntax. The first component of a HASL formula is a restriction of hybrid automata [ACHH92], namely synchronized Linear Hybrid Automata (LHA). LHA extend the Deterministic Timed Automata (DTA) used to describe properties of Markov chain models [DHS09, CHKM09]. Simply speaking, LHA are automata whose set of *locations* is associated with a n-tuple X of real-valued variables (called data variables) and whose rate can vary.

In our context, LHA are used to synchronize with DESP paths. However they can evolve in an autonomous way: thus the symbol $\sharp$ denotes a pseudo-event that is not included in the event set E of the DESP associated with these autonomous changes. The values of the data variables $x_1, \ldots, x_n$ evolve with a linear rate depending on the location of the automaton and on the current state of the DESP. More precisely the function *flow* associates to each location a 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 ith 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 $\sum_{1 \le i \le n} \alpha_i x_i + c \prec 0$ where α_i and c are indicators (*i.e.* in Ind), $\prec$ stands for either $=, <, >, \le$ or $\ge$. The set of constraints is denoted by Const. Given a location and a state, an expression of the form $\sum_{1 \le i \le n} \alpha_i x_i + c$ evolves linearly 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. We denote by lConst the set of left closed constraints that are used for the "autonomous" edges (i.e. those labelled by $\sharp$). 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 a n-tuple of functions $u_1, ..., u_n$ where each u_k is of the form $x_k = \sum_{1 \le i \le n} \alpha_i x_i + c$ where $\alpha_i \in Ind$ and c are indicators. The set of updates is denoted by Up.

DEFINITION 2. *A* synchronized linear hybrid automaton *(LHA)* $\mathcal{A} = \langle E, L, \Lambda, \mathit{Init}, \mathit{Final}, X, \mathit{flow}, \rightarrow \rangle$ *comprises:*

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

The edges labelled with a set of events in 2^E are called synchronized *whereas those labelled with $\sharp$ are called* autonomous*. Furthermore $\mathcal{A}$ fulfills the following conditions.*

- **Initial determinism:** $\forall l \neq l' \in \mathit{Init}$, $\Lambda(l) \wedge \Lambda(l') \Leftrightarrow \texttt{false}$ [2].
- **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* $l'' \xrightarrow{\gamma, E_1, U} l$ *and* $l'' \xrightarrow{\gamma', E_2, U'} l'$ *are two distinct transitions, then either* $\Lambda(l) \wedge \Lambda(l') \Leftrightarrow \texttt{false}$ *or* $\gamma \wedge \gamma' \Leftrightarrow \texttt{false}$[2].
- **Determinism on $\sharp$:**[3] $\forall l, l', l'' \in L$, *if* $l'' \xrightarrow{\gamma, \sharp, U} l$ *and* $l'' \xrightarrow{\gamma', \sharp, U'} l'$ *are two distinct transitions, then either* $\Lambda(l) \wedge \Lambda(l') \Leftrightarrow \texttt{false}$ *or* $\gamma \wedge \gamma' \Leftrightarrow \texttt{false}$[2].
- **No $\sharp$-labelled loops:** *For all sequences* $l_0 \xrightarrow{\gamma_0, E_0, U_0} l_1 \xrightarrow{\gamma_1, E_1, U_1} \cdots \xrightarrow{\gamma_{n-1}, E_{n-1}, U_{n-1}} l_n$ *such that* $l_0 = l_n$, *there exists* $i \le n$ *such that* $E_i \neq \sharp$ [4].

Discussion. The motivation for the distinction between two types of edges in the LHA is that the transitions in the synchronized system (DESP + LHA) will be either autonomous, *i.e.* time-triggered (or rather variable-triggered) and take place as soon as a constraint is satisfied, or synchronized *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 behavior through synchronized transitions, but also take its own autonomous transitions in order to evaluate the desired property. In order to ensure that the first time instant at which a constraint is satisfied exists, we require for the constraints on autonomous transitions to be left closed. 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 or 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 [Can88], such an extension for autonomous transitions cannot be considered for obvious efficiency reasons.

The automata we consider are deterministic: given a path σ of a DESP, there is exactly one synchronization with the linear hybrid automaton. This constraint ensures the synchronized system is still

[2] These equivalences must hold whatever the interpretation of the indicators occurring in $\Lambda(l)$, $\Lambda(l')$, γ and γ'.

[3] Note that our two notions of determinism allow an autonomous and a synchronised edges to be simultaneously fireable

[4] This condition is sufficient to avoid an infinite behavior of $\mathcal{A}$ without synchronization.

a stochastic process. In the above definition, the first three conditions ensure the uniqueness of the synchronization. At last, the fourth disables "divergence" of the synchronization, i.e. the possibility of an infinity of consecutive autonomous events without synchronization.

Notations. A *valuation* ν maps every data variable to a real value. 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 synchronized LHA is, given an execution of a corresponding DESP, to first decide whether the execution is to be accepted or not, and also to maintain data values along the execution. Formal semantics of the synchronized stochastic process is given in appendix A. Here we describe how a timed path $\sigma = s_0 \xrightarrow{e_0,\tau_0} s_1 \xrightarrow{e_1,\tau_1} \ldots$ of a DESP $\mathcal{D}$ is synchronized with an LHA $\mathcal{A}$.

First there are two kinds of configurations for a synchronization:

- Non final configurations: (s_i, l, ν, τ) with $l \notin Final$, $s_i \models \Lambda(l)$ where τ $(\leq \tau_i)$ is the current time.
- Final configurations: (s_i, l, ν, τ) with $l \in Final$, $s_i \models \Lambda(l)$ and $\tau \leq \tau_i$ or the *implicit* final rejecting configuration $\perp$.

Given a non final configuration (s_i, l, ν, τ), some time elapses until an autonomous transition or a synchronized transition of $\mathcal{A}$ can be taken. So we describe the effect of a time step $0 \leq \delta \leq \tau_i - \tau$ on a configuration: the new configuration is $(s_i, l, \nu', \tau + \delta)$ with for every $1 \leq k \leq n$, $\nu'(x_k) = \nu(x_k) + \mathit{flow}_k(l)(s_i)\delta$.

An autonomous transition $l \xrightarrow{\gamma,\sharp,U} l'$ is fireable after letting elapse δ if $(s_i, \nu') \models \gamma$ and $s_i \models \Lambda(l')$. In this case the configuration potentially reached is $(s_i, l', U(s_i, \nu'), \tau + \delta)$. Since we forbid non determinism, for every potentially fireable autonomous transition, we determine its earliest firing (which exists since we allow only left closed constraints) and choose the autonomous transition that can fire the earliest. Due to the determinism on $\sharp$ only one autonomous transition is fireable at a time.

If no autonomous transition is fireable for any $\delta \in [0, \tau_i - \tau]$, we fix $\delta = \tau_i - \tau$ and we consider the synchronized transitions $l \xrightarrow{\gamma,E',U} l'$ such that $e_i \in E'$, $(s_i, \nu') \models \gamma$ and $s_{i+1} \models \Lambda(l')$. Due to the determinism of $\mathcal{A}$, there is at most one such transition. If there exists one, we reach the configuration $(s_{i+1}, l', U(s_i, \nu'), \tau_i)$. Otherwise we reach the final configuration $\perp$.

It remains to start the synchronization. If there is some $l_0 \in \mathit{Init}$ such that $s_0 \models \Lambda(l_0)$ the initial configuration is $(s_0, l_0, \nu_0, 0)$ where for all x_i, $\nu_0(x_i) = 0$. Note that, by initial determinism, there is at most one $l \in \mathit{Init}$ such that s_0 satisfies $\Lambda(l)$. Otherwise the synchronization starts and immediately ends up in state $\perp$. Observe that there are two possible behaviors for the synchronization. Either it ends up in some final state leading to a finite *synchronizing path* or the synchronization goes over all states of the path without ever reaching a final configuration. We discuss this point in the next section.

Example. The two LHA of figure 2 and 3 intend to illustrate the expressiveness of HASL's LHAs. They are meant to synchronize with the shared memory system of figure 1. The one in figure 2 uses two variables. x_0 has a null flow in every location and is used to count the number of memory accesses granted. x_1 expresses the difference of memory usage between processes of class 1 and 2. It has thus flow 1 (*resp.* -1) when the memory is used by class 1 (*resp.* 2) processes, and 0 when the memory is not used. As soon as k processes have been given a memory access, the system terminates in state l_3 or l_4 depending on which process type has used the memory for the longest cumulated period.

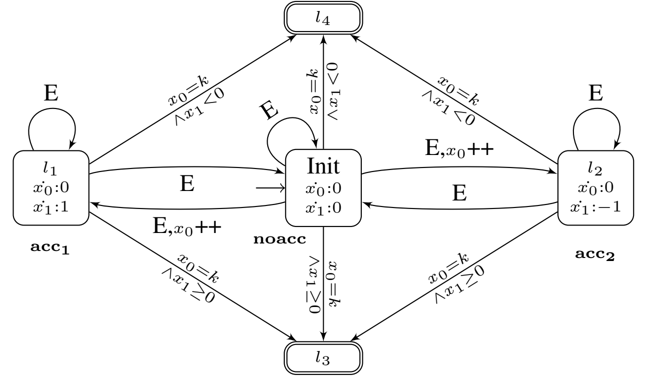

Figure 2: An LHA to compute the difference of memory usage

The example of figure 3 uses indicator dependent flows. x_1 counts the cumulated waiting time of processes of class 1 before k of them have been served. The flow $nbreq_1$ corresponds to the number of tokens in place $Request_1$ in the current marking whereas event $Serv_1$ corresponds to the firing of the bottom left transition of the SPN of figure 1. x_2 is the number of served processes of class 1 which is updated due to event $Serv_1$.

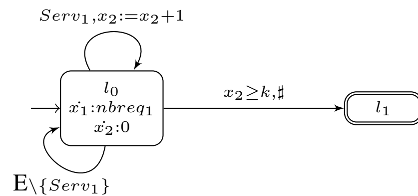

Figure 3: An LHA to compute the average waiting time

3.2 HASL expressions

The second component of a 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 is defined by the following grammar:

$$
\begin{aligned}
Z &::= E(Y) \mid Z + Z \mid Z \times Z \\
Y &::= c \mid Y + Y \mid Y \times Y \mid Y/Y \mid last(y) \mid min(y) \\
&\quad\ \mid max(y) \mid int(y) \mid avg(y) \\
y &::= c \mid x \mid y + y \mid y \times y \mid y/y
\end{aligned}
\tag{1}
$$

y is an arithmetic expression built on top of LHA data variables (x) and constants (c). Y is a path dependent expression built on top of basic path random variables such as $last(y)$ (i.e. the last value of y along a synchronizing path), $min(y)$/$max(y)$ (the minimum, resp. maximum, value of y along a synchronizing path), $int(y)$ (i.e. the integral over time along a path) and $avg(y)$ (the average value of y along a path). Finally Z, the actual target of HASL verification, is an arithmetic expression built on top of the first moment of Y ($E[Y]$), 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, $Var(Y) \equiv E[Y^2] - E[Y]^2$, $Covar(Y_1, Y_2) \equiv E[Y_1 \cdot Y_2] - E[Y_1] \cdot E[Y_2]$. Note that for efficiency reasons, in the implementation of the software tool, we have considered a restricted version of grammar (1), where products and quotients of data variables (e.g. $x_1 \cdot x_2$ and x_1/x_2) are allowed only within the scope of the $last$ operator (*i.e.* not with min, max, int or avg). This is because allowing products and quotients as arguments of path operators such as max, min requires the solution of a linear programming problem during the generation of a synchronized $\mathcal{D} \times \mathcal{A}$ path which, although feasible, would considerably affect the computation time.

Semantics. Given $\mathcal{D}$ a DESP and $(\mathcal{A}, Z)$ a HASL formula, we assume that with probability 1, the synchronizing path generated by a random execution path of $\mathcal{D}$ reaches a final state. 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 fulfills the target subformula of the `Until` operator. This (still open) issue is also addressed in [SVA05b, HJB$^+$10]. Due to this assumption, the random path variables are well defined and the expression Z associated with the formula may be evaluated with expectations defined w.r.t. the distribution of a random path *conditioned by acceptance of the path*. In other words, the LHA both calculates the relevant measures during the execution and selects the relevant executions for computing the expectations. This evaluation gives the result of the formula $(\mathcal{A}, Z)$ for $\mathcal{D}$.

Example. Referring to the LHA of figure 2, we can consider path random variables such as $Y = last(x_1)$ (the final difference of memory 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)), $E(last(x_0))$ will give the probability to reach l_3. With the LHA of figure 3, we can express (an overestimation of) the average waiting time by means of $Y = last(x_1/x_2)$. An underestimation can also be computed counting the overall number of requesting processes instead of the number of served processes.

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 offer to capture more complex examples and properties, and facilitates the expression and the computation of costs and rewards.

CSL. In [ASSB00] the logic 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 continuos path-quantifier $P_{\prec r}$ ($\prec \in \{<, \leq, \geq, >\}$, $r \in [0, 1]$). Thus a CSL formula $P_{\prec r}\varphi$ expresses that the probability of CTMC paths satisfying condition φ fulfills the bound $\prec r$, where φ is, typically, a time-bounded `Until` formula. In [BHHK03] it has been demonstrated that the evaluation of the probability measure of a (time-bounded) CSL specification corresponds to the transient analysis of a (modified) CTMC, for which efficient approximate numerical procedures exist.

CSRL. In the logic CSRL introduced by [BHHK00], CSL is extended to take into account Markov reward models, i.e. CTMCs with a single reward on states. The global reward of a path execution is then the integral of the instantaneous reward over time. In CSRL, the path operators `Until` and `Next` include also an interval specifying the allowed values for accumulated reward. Moreover new operators related to the expectation of rewards are defined. A numerical approach is still possible for approximating probability measures but its complexity is significantly increased. This formalism is also extended with rewards associated to actions [LKKP05]. CSRL is appropriate for standard performability measures but lacks expressiveness for more complex ones.

asCSL. In the logic asCSL introduced by [BCH$^+$07], the single interval time constrained `Until` of CSL is replaced by a regular expression with a time interval constraint. These path formulas can now express elaborated functional requirements as in CTL* but the timing requirements are still limited to a single interval globally constraining the path execution.

CSL^{TA}. In the logic CSL^{TA} introduced by [DHS09], the path formulas are defined by a single-clock deterministic time automaton. This clock can express timing requirements all along the path. From an expressiveness point of view, it has been shown that CSL^{TA} is strictly more expressive than CSL and that path formulas of CSL^{TA} are strictly more expressive than those of asCSL. Finally, the verification procedure is reduced to a reachability probability in a semi-Markovian process yielding an efficient numerical procedure.

DTA. In [CHKM09], 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. In order to exploit this theoretical result, a procedure for approximating this probability is designed based on a system of partial differential equations.

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 (even with DTA [DHS09], the most expressive among the logic for CTMC verification, properties of a model can be expressed only by means of clocks variables, while sophisticated measures corresponding to variables with real-valued rates cannot be considered). Furthermore observe that the evolution of stochastic logics seems to have followed two directions: one targeting *temporal reasoning* capability (evolutionary path: CSL $\rightarrow$ asCSL $\rightarrow$ CSL^{TA} $\rightarrow$ DTA), the other targeting *performance evaluation* capability (evolutionary path: CSRL $\rightarrow$ CSRL+impulse rewards). A unifying approach is currently not available, thus, for example, one can calculate the probability of 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: a unifying approach. 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 modeling point of view HASL allows for targeting of 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 LHA 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 both for model-checking and for performance and dependability studies. Note that the nature of the (real-valued) expression Z (1) (characterizing the outcome of a HASL formula) generalizes 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^{TA} and DTA a measure of probability). *Cost functions.* 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. It thus extends the possibilities of CSRL and its extensions [LKKP05] where only one reward function (on states and actions) is considered.

Finally we briefly discuss on the issue of nesting of probabilistic operators. Nesting of probabilistic operators, which is present in all stochastic logics discussed above, is meaningful only when an identification can be made between a state of the probabilistic system and a configuration (comprising the current time and the next scheduled events). Whereas this identification was natural for Markov chains, it is not possible with DESP and general distributions, and therefore this operation has not been considered in HASL. A similar problem arises for the steady state operator. The existence of a steady state distribution raises theoretical problems, except for finite Markov chains, but with HASL we allow for not only infinite state systems but also non Markovian behaviors. However, when the DESP has a regeneration point, various steady state properties can be computed by defining the regeneration point as a final state. For the expressiveness of HASL, we can state that when we omit nesting and steady state properties, HASL is at least as expressive as CSRL and DTA : *Every non nested transient CSRL or DTA formula can be expressed with HASL.*

4. SOFTWARE SUPPORT

We have been developing a prototype software tool for HASL verification, named COSMOS. COSMOS employs a (simulation-based) statistical approach to estimate measures of interest of the considered DESP model. We first give a brief summarize of the basic principles behind statistical model checking approaches, outlining how the approach supported by COSMOS compares with that of existing statistical model checkers. Then we report about an empirical experience regarding the application of COSMOS to the analysis of the shared memory model previously introduced (Figure 1). This includes a comparative evaluation of the performances, where the execution time of COSMOS is confronted with that of a popular probabilistic model checker, namely PRISM [PRI].

4.1 Statistical model checking

Model checking by statistical techniques [YS06, SVA05a, HLP06] constitute an (increasingly spreading) alternative to common probabilistic model checking based on numerical methods [ASSB00, HCH+02, BCH+07, DHS09, CHKM09]. The basic principle of statistical verification of probabilistic models is one of employing discrete-event simulation combined with statistical techniques in order to *reason-about/estimate* the likelihood of relevant properties. Hypothesis testing combined with stochastic simulation can be used whenever the considered problem concerns deciding whether the probability of a property ϕ fulfills a certain bound $Pr(\phi) \sim p$ ($p \in [0,1]$, $\sim \in \{<, \leq, \geq, >\}$) [YS06, SVA05a]; on the other hand whenever the focus is on estimation of $Pr(\phi)$ simulation is combined with confidence intervals methods to obtain arbitrarily accurate estimates of $Pr(\phi)$ [KZ09, HLP06, PRI]. The main appeal of statistical approaches is that they allow one to assess very large models which cannot be treated otherwise (i.e. through, memory demanding, numerical methods). The downside lies in the execution time which blows up with the accuracy chosen for the estimated outcome. Notably with existing model checkers supporting statistical verification (e.g. [SVA05b, You05, KZ09, HLP06, PRI]) the focus is (exclusively) on the estimation of measures of probability (which is to say, estimating the first moment of a Binomial random variable corresponding to the number of simulated paths satisfying ϕ). To the best of our knowledge estimation of more general quantities, involving moments of generic random variables, it is not supported by existing tools featuring statistical verification of stochastic models.

4.2 The COSMOS tool

COSMOS extends existing statistical model checking approaches in two respects: it provides the modeler with the ability to consider generic random variables (including binomial ones) as the quantity the estimation depends upon; secondly, it allows to target the estimation of functions of (generic) moments of random variables and not only the first moment of a Binomial variable. Thus, whereas existing statistical model checkers are concerned with estimating $E[Y]$ (where $Y \sim Binomial$ and depends on ϕ), COSMOS is concerned with estimating generic expressions such as, for example, $E[Y]$, $Var(Y) \equiv E[Y^2] - E[Y]^2$, $Covar(Y_1, Y_2) \equiv E[Y_1 \cdot Y_2] - E[Y_1] \cdot E[Y_2]$ (where Y, Y_1 and Y_2, are generic random variables depending on ϕ). In practice COSMOS takes three inputs: a DESP, expressed in terms of an (extended) GSPN model $\mathcal{N}$, an LHA $\mathcal{A}$, representing the random variables of interest and an expression Z (obtained through the grammar (1)) based on moments of the random variable represented by $\mathcal{A}$. The tool outcome is an estimation of the value of Z obtained by repeated sampling of the considered random variables where each sample is obtained by execution of an independent simulation run of the $\mathcal{N} \times \mathcal{A}$ product process. At present no graphical user interface is supported thus both the model $\mathcal{N}$ and the automaton $\mathcal{A}$ are described in a formatted textual form.

Implementation details. COSMOS is implemented in C++ and uses the BOOST libraries for generating the random numbers necessary for stochastic simulation. Furthermore COSMOS code compilation is optimized through the LLVM compiler infrastructure [llv].
Events are generated according to the corresponding delay distribution and maintained in a time-ordered fashion in the *event queue* (EQ) which is stored through a binary-min-heap structure. Using a binary-heap structure for maintaining the EQ guarantees a $O(\log(n))$ worst-case cost for insertion/deletion operations, how-

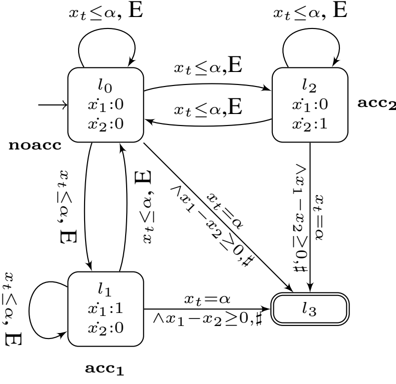

Figure 4: an LHA for measures based on the utilization time divergence between shared memory system of Figure 1

ever it also poses some non-trivial issues when it comes with handling of concurrent events. Since delays are governed by not necessarily continuous distributions (e.g. deterministic delay are allowed) then simultaneous events are possible. To disambiguate simultaneous events while adopting a binary-min-heap for maintaining the EQ we have developed the following solution: when a transition $t : (dist_t, pri_t, weight_t)$ (where $dist_t$ is the type of delay-distribution, while $pri_t \in \mathbb{N} \cup \{\infty\}$ and $weight_t \in \mathbb{R}_{>0}$ are the priority, respectively, the weight of t) becomes enabled the corresponding event $e_t = (d_t, pri_t, w_t)$ is created where $d_t, w_t \in \mathbb{R}^+$ are generated[5] according to $d_t \sim dist_t$ and $w_t \sim NegExp(weight_t)$ and representing, respectively, the firing time of t and a disambiguating value (stochastically) proportional to $weight_t$. The value w_t serves for ordering equally delayed events as they are inserted in the (binary-heap) EQ. In particular, on insertion of e_t in the EQ the following ordering schema is adopted when e_t is compared with an event $e_{t'} = (d_{t'}, pri_{t'}, w_{t'})$ already present in the EQ: $e_t < e_{t'}$ (meaning that e_t will occur before $e_{t'}$) iff $(d_t < d_{t'})$ or $(d_t = d_{t'} \wedge pri_t > pri_{t'})$ or $(d_t = d_{t'} \wedge pri_t = pri_{t'} \wedge w_t < w_{t'})$. With respect to the efficiency the cost for supporting non-continuously distributed delays, while keeping the low maintenance cost of a binary-heap EQ, is paid in terms of an extra random number generation operation (the generation of w_t) which is performed on creation of each event. Note that this extra cost can be avoided for the subclass of DESP such that all delays are described by continuous random variables. Finally note that COSMOS features *on-the-fly* calculation at two levels: the value of each path random variable Y_i contained in the expression $Z = f(Y_i)$ associated to an experiment, is updated *on-the-fly* on generation of a simulated trajectory; instead the value of the whole expression Z is updated, *on-the-fly*, at the end of each simulation run.

4.3 Experiments

We report about experience of application of the COSMOS tool for assessing measures of quantities referred to a DESP model of the shared memory system illustrated, in Petri Net form, in Figure 1. In particular we consider two type of experiments: the first one targeted to the estimation of a *measure of probability* (Experiment 1); the second one concerned with the estimation of a *real-valued quantity* (Experiment 2). An extended list of HASL expressions examples (referred to the shared memory model) are reported in Table 4.

The LHA for the experiments. Both types of experiment are referred to the LHA of Figure 4 (a variant of the example presented in Figure 2), which allows for recognizing executions such that , *"the memory has been occupied the longer (or as long as) by class 1 processes than by class 2's"* (within time-bound $[\alpha, \alpha]$, $\alpha \in \mathbb{R}^+$). Such automata uses 3 clock variables x_t storing the simulation time, x_1 and x_2 counting the occupation time by class 1, respectively class 2, processes. The initial location l_0 corresponds to states such that the memory is free, thus both x_1 and x_2 flows are null. Location $l_1(l_2)$ corresponds to states such that the memory is allocated to a class 1(class 2) process, thus the timer $x_1(x_2)$ is "on" : i.e $flow(x_1) = 1(flow(x_2) = 1)$. The accepting location (l_3) is reached through autonomous edges from any other location as soon as simulation time reaches the bound $x_t = \alpha$ and on condition that occupation time difference is $x_1 - x_2 \ge 0$. An extra (boolean) variable, referred to as x_{succ}, is used (even if not indicated in Figure 4) for storing the outcome of each simulated trajectory in the $\mathcal{N} \times \mathcal{A}$ process. (i.e x_{succ} is assigned with $x_{succ} := 1$ on entering an accepting location of the automaton, or with $x_{succ} := 0$ if simulation blocks before).

Experiment settings. The shared memory model depends on the parameters listed in Table 1. For our experiments we have considered to analyze the behavior of the model in function of the clients' arrival rate. In particular we have chosen to study configurations such that the arrival rate of class 1 processes is fixed to $\lambda_1 = 1$ (i.e. one arrival per time unit, on average) while those of class 2 processes' is varied in the set $\lambda_2 \in \{8/10, 9/10, 1, 11/10, 12/10\}$ (i.e. average arrival frequency varying between an arrival every eight tenths and twelve tenths of a time unit). The occupation of the memory is set to a uniform distribution over the interval $[\alpha, \beta] = [0.4, 0.5]$ for both classes of clients (i.e. each client occupies the memory for 0.45 time units, on average). Note that under these settings the system is stable as the utilization $\rho = (\lambda_1 + \lambda_2)(\alpha + \beta)/2 < 1$, $\forall \lambda_2 \in \{8/10, 9/10, 1, 11/10, 12/10\}$ (i.e. $\rho_{max} = 0.99 \leftrightarrow \lambda_2 = 12/10$, $\rho_{min} = 0.81 \leftrightarrow \lambda_2 = 8/10$). Finally, priorities and weight of immediate transitions are all set to 1 (i.e. competing clients are equally likely to grant access to the memory).

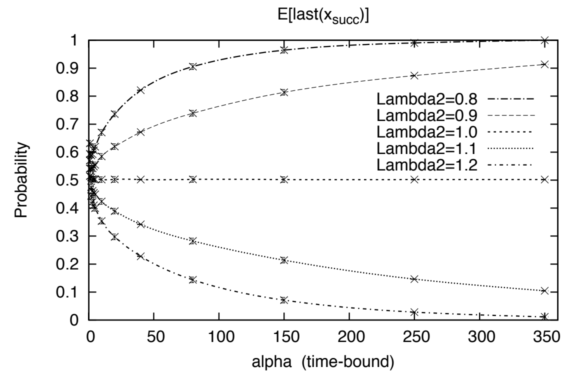

Figure 5: Probability that the memory is used longer by class 1 processes

Experiment 1 ($Z \equiv E[last(x_{succ})]$): the goal of this experiment is to measure *"the probability that the memory has been occupied the longer (or as long as) by class 1 processes than by class 2's"* (in the time-window $[0, \alpha]$, $\alpha \in \mathbb{R}^+$). This is achieved by considering the HASL Expression $Z \equiv E[last(x_{succ})]$ (recall that x_{succ} is an LHA boolean variable set to 1 on reaching of an accepting location and to 0 if the synchronization blocks).

Results: Figure 5 reports about the estimation of a 99%confidence-interval of $Z \equiv E[last(x_{succ})]$. The Y-axis represents (estimated)

[5] if t is immediate (i.e. $dist_t \equiv$ 0-Dirac), then only w_t is generated.

	T	probability measure			generated paths		exec-time		path avg exec-time		ratio
		PRISM num	PRISM stat	COSMOS	PRISM	COSMOS	PRISM	COSMOS	PRISM	COSMOS	COSMOS/PRISM
$C=5$	20	0.33574	0.33711	0.33564	92042	59000	20.27000	30.91000	0.00022	0.00052	2.38
	40	0.56931	0.56992	0.57057	92042	64800	32.91000	53.12000	0.00036	0.00082	2.29
	60	0.72075	0.71870	0.71985	92042	53300	40.38000	53.59000	0.00044	0.00101	2.29
	80	0.82229	0.81878	0.82165	92042	38800	45.87000	44.75000	0.00050	0.00115	2.31
	100	0.88261	0.88365	0.88528	92042	26900	49.41000	32.39000	0.00054	0.00120	2.24
	200	0.98655	0.98671	0.98694	92042	3600	55.63000	4.92000	0.00060	0.00137	2.26
$C=7$	20	0.07611	0.07655	0.07040	92042	17300	23.86000	10.53000	0.00026	0.00061	2.35
	40	0.15320	0.15205	0.14512	92042	32800	44.89000	36.23000	0.00049	0.00110	2.26
	60	0.22386	0.22419	0.21442	92042	44600	63.76000	69.57000	0.00069	0.00156	2.25
	80	0.28863	0.28917	0.27793	92042	53100	81.83000	105.33000	0.00089	0.00198	2.23
	100	0.34799	0.34987	0.33609	92042	59000	95.74000	138.90000	0.00104	0.00235	2.26
	200	0.57825	0.57859	0.56151	92042	65100	157.71000	250.47000	0.00171	0.00385	2.25

Table 2: comparing COSMOS vs PRISM statistical estimation wrt the tandem queueing system

probability values, while the X-axis represents the value of the time bound α. The five plots in Figure 5 correspond to the five possible value of the arrival rate $\lambda_2 \in \{8/10, 9/10, 1, 11/10, 12/10\}$ ($\lambda_1 = 1$ is kept constant), thus each curve correspond to one amongst the following combinations $(\lambda_1, \lambda_2) \in \{(1, 0.8), (1, 0.9), (1, 1), (1, 1.1), (1, 1.2)\}$. All curves converge asymptotically to a constant point corresponding to reaching the of the *steady-state* of the model. With the fully symmetric configuration (i.e. $\lambda_2 = 1$) processes of both class are equally likely to occupy the memory for the majority of time (plot $\lambda_2 = 1$ tending to 0.5); furthermore observe that, in symmetric conditions, the probability converges pretty quickly (i.e. within 10 time units). If class 1 processes arrive at a faster rate than class 2's (i.e. configurations $\lambda_2 = 0.8$ and $\lambda_2 = 0.9$) they are more likely to occupy the resource the longer and they will do so with probability 1 at times greater than 300 time units (curves $\lambda_2 = 0.8$ and $\lambda_2 = 0.9$ tending to 1).

The dual is true if class 2 processes arrive at a faster rate then class 1's (curves $\lambda_2 = 1.1$ and $\lambda_2 = 1.2$ tending to 0). Finally it can be observed that biased configurations converge later than the fully symmetric one: curves corresponding to unbalanced settings stabilize at times greater than (roughly) 300 time units.

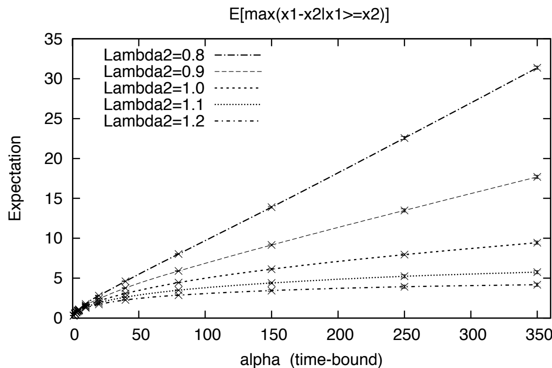

Figure 6: Probability that the memory is used longer by class 1 processes

Experiment 2 - ($Z \equiv E[max(x_1 - x_2)]$): the goal of this experiment is to measure *"the maximum of the difference between the occupation time given that the memory has been occupied the longer (or as long as) by class 1 processes than by class 2's"* (in the time-window $[0, \alpha]$, $\alpha \in \mathbb{R}^+$). This is achieved by considering the HASL Expression $Z \equiv E[max(x_1 - x_2)]$ (recall that x_1 and x_2 are timers counting the occupation of the shared memory by class 1, respectively class 2, processes).

Results: Figure 6 reports about the estimation of a 99%confidence-interval of $Z \equiv E[max(x_1 - x_2)]$. Note that, in this case, estimated values fall well beyond the probability interval $[0, 1]$, which is possible whenever the estimated quantity is not a probability measure. Curves in Figure 6 indicate that the maximum of the occupation time (non-negative) difference (i.e. $max(x_1 - x_2)|x_1 \geq x_2$) is an increasing function of time which tends towards linearity as the arrival rate of class 2 processes decreases.

Evaluating the performance of COSMOS. In order to assess the performance of COSMOS we have realized a comparative study with a popular probabilistic model checker, namely PRISM. We have considered two well-known CTMC benchmark models: a model of a tandem queueing system (TQS), which we used for the estimation of a measure of probability, and a model of a cyclic server polling system (CSPS), which we used for the estimation of a real-valued quantity[6]. The TQS consists of a M/Cox2/1-queue sequentially composed with a M/M/1-queue. Both queues are assumed to have the same capacity given by the model's parameter $C > 0$ (note that the dimension of the resulting CTMC depends on C). For the TQS model we have considered the estimation of the probability that *"the first queue becomes full within time T"*, which can straightforwardly be encoded both in CSL, by the time-bounded Until formula $\phi \equiv (\text{true}\ U^{[0,T]}\ Q_1\text{_full})$, and in HASL, by means of a simple LHA with 2 locations. The CSPS represents a system where a *server* cyclically poll N *clients* to provide them with the service they demand for; each client switches between an idle-state and a waiting-state where it attends to be polled by the server. For the CSPS model we have considered the estimation of the *"Expected time that station 1 is waiting to be served"* (within a given time-bound $T > 0$) which can be represented both in the reward-extended version of CSL supported by PRISM and by means of a simple LHA consisting of 3 locations in COSMOS. Table 2 and Table 3 report about experiments carried out on the TQS, respectively CSPS, model[7]. Data in Table 2 (Table 3) are grouped in two blocks corresponding to different configurations of the models (i.e. queues capacity $C = 5$ and $C = 7$ for the TQS, number

[6]both models can be found under the case-studies section at [PRI].
[7]Experiments run on a laptop computer with: CPU Intel Core 2 Duo T9400, 2.53GHz, 6MB L2 Cache, 4GB RAM, OS Linux(fedora 13) 2.6.34.7.

		waiting-time measure			generated paths		exec-time		path avg exec-time		ratio
	T	PRISM num	PRISM stat	COSMOS	PRISM	COSMOS	PRISM	COSMOS	PRISM	COSMOS	COSMOS/PRISM
$N=4$	5	0.40453	0.40359	0.40257	92042	146800	138.31000	691.32000	0.00150	0.00471	3.13390
	10	1.08484	1.10262	1.08681	92042	20400	233.53000	163.44000	0.00254	0.00801	3.15770
	20	2.49446	2.55786	2.51050	92042	13200	427.15000	191.96000	0.00464	0.01454	3.13359
	30	3.90634	4.00278	3.88790	92042	20500	619.07000	431.44000	0.00673	0.02105	3.12905
	40	5.31823	5.46646	5.31336	92042	28300	824.25000	780.49000	0.00896	0.02758	3.07970
	50	6.73012	6.93534	6.72050	92042	36400	1040.10000	1244.69000	0.01130	0.03419	3.02601
$N=8$	5	0.29667	0.30152	0.29783	92042	125900	205.11000	787.06000	0.00223	0.00625	2.80531
	10	0.90120	0.92978	0.90576	92042	21900	350.69000	220.38000	0.00381	0.01006	2.64114
	20	2.29753	2.36920	2.31629	92042	17100	581.42000	290.61000	0.00632	0.01699	2.69036
	30	3.73196	3.86155	3.76046	92042	30200	827.01000	717.62000	0.00899	0.02376	2.64462
	40	5.16951	5.34971	5.20608	92042	43100	1073.46000	1324.42000	0.01166	0.03073	2.63481
	50	6.60731	6.83793	6.65669	92042	55800	1294.79000	2082.87000	0.01407	0.03733	2.65347

Table 3: comparing COSMOS vs PRISM wrt the cyclic polling system

id	estimated quantity	description
f_1	$E[last(x_{succ})\|(x_1 \geq x_2)]$	Expected probability that Res has been used longer by class-1 processes within $T=\alpha$
f_2	$E[last(x_1)\|(x_1 \geq x_2)]$	Expected occupation time of Res by class-1 processes when Res has been used longer by class-1 processes within $T=\alpha$
f_3	$E[max(x_1-x_2)\|(x_1 \geq x_2)]$	Expected maximum occupation time difference when Res has been used longer by class-1 processes within $T=\alpha$
f_4	$E[avg(x_1-x_2)\|(x_1 \geq x_2)]$	Expected average of occupation time difference when Res has been used longer by class-1 processes within $T=\alpha$
f_5	$E[var(x_1-x_2)\|(x_1 \geq x_2)]$	Expected variability of occupation time difference when Res has been used longer by class-1 processes within $T=\alpha$

Table 4: Examples of HASL formulae referred to the open shared-memory system with 2 classes of processes

of clients $N = 4$ and $N = 8$ for the CSPS). Rows of each block contains data referred to different values of the time-bound T. The three columns grouped as *probability measure* in Table 2 (and as *waiting-time measure* in Table 3) report values calculated with 1) the numerical engine of PRISM 2) the (CSL) statistical method supported by PRISM and 3) the (HASL) statistical method supported by COSMOS[8]. By comparing the results of columns PRISM *num* with those in columns PRISM *stat* and COSMOS we observe that: 1) estimates of a probability measure obtained with both (statistical) PRISM and COSMOS are comparably accurate (see Table 2); 2) estimates of the average waiting time (i.e. a non-probability measure) obtained by COSMOS are significantly more accurate than those obtained by PRISM(see Table 3);. Although we do not know the implementation details of PRISM statistical engine, we believe a possible explanation for such a lower accuracy is that PRISM is based on an error-bounding method (i.e. the *Chernoff-Hoeffding* bound one), rather than a confidence-interval approach, as the criteria to drive the estimate of a probability measure and this may lead to a non-optimal accuracy when the estimated quantity is not a measure of probability. Columns *generated paths* compare the number of paths generated by PRISM statistical engine vs COSMOS': note that, while with PRISM the number of generated paths is constant (and depends on the chosen level of confidence/accuracy), with COSMOS the paths required to meet the desired confidence/accuracy is established *on-the-fly* (i.e. path generation stops as soon as the required level of accuracy is met). Observe that in all our experiments the number of paths generated by COSMOS is well below that required by PRISM. Finally data regarding the performance comparison of the two tools are reported in the remaining columns: column *exec-time* compares the total time needed by the statistical engines to output the estimate; column *path avg exec-time* compares the average time to generate a single trajectory (i.e. *exec-time/generated paths*); column *ratio* report the ratio of the the path average execution times. This indicates that COSMOS is (on average) about 2.3 times slower than PRISM' statistical engine, with respect to the TQS model, and about 2.9 times slower, with respect to the CSPS model. The lower speed of COSMOS is not surprising considering that simulation of the synchronized $\mathcal{D}\times\mathcal{A}$ process, as per HASL, is inherently more (computationally) demanding than the simulation of a CTMC, as per CSL.

[8]Statistical estimates have been obtained with the following settings: confidence-level $\epsilon=0.99$, approximation-level $\delta=0.01$.

5. CONCLUSION

We have presented a new logic for expressing elaborated properties related to stochastic processes. Contrary to previous approaches, a formula of HASL returns a conditional expectation whose condition is based on acceptance by a linear hybrid automaton. Such a logic can be employed both for probabilistic validation of functional properties or for elaborated performance analysis as we have illustrated with examples. We have developed a tool to experimentally validate the feasibility of the statistical based approach. While the first results are promising, we aim at overcoming the limitations of this approach namely (1) accelerating the path generation when faced to difficult acceptance condition *via* the rare event approach and (2) analyze the structure of the DESP in order to circumvent the constraint that almost surely a path is accepted or rejected by the LHA. With respect to performance improvement of the tool we are currently working on a *Model Driven Architecture* version of COSMOS where (optimized) source code for both the considered model and the considered LHA is automatically generated and integrated in the tool architecture, which is then fed in to LLVM to

get an efficient executable.

6. REFERENCES

[ACD91] R. Alur, C. Courcoubetis, and D. Dill. Model-checking for probabilistic real-time systems. In *18th Int. Coll. on Automata, Languages and Programming (ICALP'91)*, volume 510 of *LNCS*, pages 115–126. Springer, 1991.

[ACHH92] R. Alur, C. Courcoubetis, T. A. Henzinger, and P.-H. Ho. Hybrid automata: An algorithmic approach to the specification and verification of hybrid systems. In *Hybrid Systems*, volume 736 of *LNCS*, pages 209–229. Springer, 1992.

[AMBC+95] M. Ajmone Marsan, G. Balbo, G. Conte, S. Donatelli, and G. Franceschinis. *Modelling with Generalized Stochastic Petri Nets*. John Wiley & Sons, 1995.

[ASSB00] A. Aziz, K. Sanwal, V. Singhal, and R. Brayton. Model-checking continuous time Markov chains. *ACM Transactions on Computational Logic*, 1(1):162–170, 2000.

[BCH+07] C. Baier, L. Cloth, B. Haverkort, M. Kuntz, and M. Siegle. Model checking action- and state-labelled Markov chains. *IEEE Transactions on Software Engineering*, 33(4):209–224, 2007.

[BHHK00] C. Baier, B. R. Haverkort, H. Hermanns, and J.-P. Katoen. On the logical characterisation of performability properties. In *27th International Colloquium, ICALP 2000*, volume 1853 of *LNCS*, pages 780–792. Springer, 2000.

[BHHK03] C. Baier, B. Haverkort, H. Hermanns, and J.-P. Katoen. Model-checking algorithms for continuous-time Markov chains. *IEEE Transactions on Software Engineering*, 29(6):524–541, 2003.

[Can88] J. Canny. Some algebraic and geometric computations in pspace. In *20th ACM Symposium on Theory of Computing (STOC'88)*, pages 460–467. ACM, 1988.

[CHKM09] T. Chen, T. Han, J.-P. Katoen, and A. Mereacre. Quantitative model checking of continuous-time Markov chains against timed automata specifications. In *Proc. of the 24th Annual IEEE Symposium on Logic in Computer Science (LICS)*, pages 309–318. IEEE, 2009.

[DHS09] S. Donatelli, S. Haddad, and J. Sproston. Model checking timed and stochastic properties with CSL^{TA}. *IEEE Transactions on Software Engineering*, 35:224–240, 2009.

[EC80] E. A. Emerson and E. M. Clarke. Characterizing correctness properties of parallel programs using fixpoints. In *Proceedings of the 7th Colloquium on Automata, Languages and Programming*, volume 85 of *LNCS*, pages 169–181, 1980.

[Gly83] P. W. Glynn. On the role of generalized semi-Markov processes in simulation output analysis. In *Proceedings of the 15th conference on Winter simulation*, volume 1, pages 38–42, 1983.

[HCH+02] B.R. Haverkort, L. Cloth, H. Hermanns, J-P. Katoen, and C. Baier. Model checking performability properties. In *DSN '02: Proceedings of the 2002 International Conference on Dependable Systems and Networks*, pages 103–112. IEEE, 2002.

[HJB+10] R. He, P. Jennings, S. Basu, A. P. Ghosh, and H. Wu. A Bounded Statistical Approach for Model Checking of Unbounded Until Properties. In *Int. Conf. on Automated Software Engineering (ASE'10)*, pages 225–234. IEEE/ACM, 2010.

[HLP06] T. Herault, R. Lassaigne, and S. Peyronnet. APMC 3.0: Approximate verification of discrete and continuous time Markov chains. In *Proc. QEST'06*, pages 129–130, 2006.

[KZ09] J. P. Katoen and I. S. Zapreev. Simulation-based ctmc model checking: an emprical evaluation. In *Proc. QEST'09*, pages 31–40. IEEE, 2009.

[LKKP05] L.Cloth, J.-P. Katoen, M. Khattri, and R. Pulungan. Model checking markov reward models with impulse rewards. In *Int. Conference on Dependable Systems and Networks (DSN 2005)*, pages 722–731, 2005.

[llv] Llvm home page. http://llvm.org/.

[PRI] Prism home page. http://www.prismmodelchecker.org.

[SVA05a] K. Sen, M. Viswanathan, and G. Agha. On statistical model checking of stochastic systems. In *Computer Aided Verification*, pages 266–280, 2005.

[SVA05b] K. Sen, M. Viswanathan, and G. Agha. VESTA: A statistical model-checker and analyzer for probabilistic systems. In *Proc. QEST'05*, pages 251–252, 2005.

[You05] H.L.S. Younes. Ymer: A statistical model checker. In *Computer Aided Verification (CAV)*, volume 3576, pages 429–433. Springer, 2005.

[YS06] H.L.S. Younes and R.G. Simmons. Statistical probabilistic model checking with a focus on time-bounded properties. *Inf. Comput.*, 204(9):1368–1409, 2006.

APPENDIX

A. FORMAL SEMANTICS

A.1 Formal semantics of a DESP

Given a discrete event system, its execution is characterized 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. Formally, the stochastic behaviour of a DESP is defined by two families of random variables :

- $S_0, \ldots, S_n, \ldots$ 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 after the nth 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 nth event.

We start from the syntactical definition of a DESP given in section 2 and show how we obtain the two families of random variables $\{S_n\}_{n\in\mathbb{N}}$ and $\{\tau_n\}_{n\in\mathbb{N}}$. This definition is inductive w.r.t. n and includes some auxiliary families. The family $\{sched(e)_n\}_{n\in\mathbb{N}}$ whose range is $\mathbb{R}^+ \cup \{+\infty\}$ denotes whether the event e is already scheduled on reaching the nth state and what is the current schedule.

The family $\{sched^+(e)_n\}_{n\in\mathbb{N}}$ is the completion of scheduling:

- For every $e \in E$,
  $Pr(sched^+(e)_n = \infty \mid e \notin enabled(S_n)) = 1$
- For every $e \in E$,
  $Pr(sched^+(e)_n = sched(e)_n \mid$
  $e \in enabled(S_n) \wedge sched(e)_n \neq \infty) = 1$
- For every $e \in E$ and $d \in \mathbb{R}^+$,
  $Pr(\tau_n \leq sched^+(e)_n \leq \tau_n + d \mid$
  $e \in enabled(S_n) \wedge sched(e)_n = \infty) = delay(S_n, e)(d)$

Observation. In the whole section, when we write an expression like $Pr(sched^+(e)_n = \infty \mid e \notin enabled(S_n))$, we also mean that this conditional probability is independent from any event Ev that could be defined using the previously defined variables:
$Pr(sched^+(e)_n = \infty \mid e \notin enabled(S_n))$
$= Pr(sched^+(e)_n = \infty \mid e \notin enabled(S_n) \wedge Ev)$

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: $E'_n = \{e \in E \mid \forall e' \in E, sched^+(e)_n \leq sched^+(e')_n\}$.

The family $\{e_n\}_{n\in\mathbb{N}^*}$ denotes the family of triggered events.
$Pr(e_{n+1} = e \mid e \in E'_n) = choice(S_n, E'_n)(e)$

Now $S_{n+1} = target(S_n, e_{n+1})$ and:

- For every $e \in E$,
  $Pr(sched(e)_{n+1} = \infty \mid e = e_n) = 1$
- For every $e \in E$,
  $Pr(sched(e)_{n+1} = \infty \mid e \notin enabled(S_{n+1}) = 1$
- For every $e \in E$,
  $Pr(sched(e)_{n+1} = sched^+(e)_n \mid$
  $e \in enabled(S_{n+1}) \wedge e \neq e_n) = 1$

We start the induction by $Pr(\tau_0 = 0) = 1$, $Pr(S_0 = s) = \pi_0(s)$ and for every $e \in E$, $Pr(sched(e)_0 = \infty) = 1$.

A.2 Formal semantics of the synchronization

Here we formally define the DESP $\mathcal{D}'$ associated with the synchronization of a DESP $\mathcal{D}$ and a LHA $\mathcal{A}$. As in the previous section, it relies on the inductive definition of families of random variables.

The family of random state variables $\{S_n\}_{n\in\mathbb{N}}$ has range in $S \times L \uplus \{\bot\}$. The family of occurrence times $\{\tau_n\}_{n\in\mathbb{N}}$ has range in $\mathbb{R}^+$. The family $\{\nu(x_k)_n\}_{n\in\mathbb{N}}$ denotes the value of variable x_k on reaching the nth state. ν_n is the valuation associated with the vector of valuations $\nu(x_k)_n$. The family of triggered events $\{e_n\}_{n\in\mathbb{N}^*}$ has range in $E \uplus \{\sharp\}$.

We first describe what happens when reaching a final state. We consider that once a final state is reached, this state is infinitely repeated due to some implicit autonomous triggering.

$$Pr(S_{n+1} = S_n \wedge \tau_{n+1} = \tau_n \wedge \nu_{n+1} = \nu_n \wedge e_{n+1} = \sharp$$

$$\mid S_n = \bot \vee \exists s \in S\ \exists l \in Final\ S_n = (s, l)) = 1$$

The family $\{sched(e)_n\}_{n\in\mathbb{N}}$ whose range is $\mathbb{R}^+ \cup \{+\infty\}$ denotes whether the event e is already scheduled on reaching the nth state and what is the current schedule. The family $\{sched^+(e)_n\}_{n\in\mathbb{N}}$ is the completion of scheduling:

- For every $e \in E$,
  $Pr(sched^+(e)_n = \infty$
  $\mid S_n = \bot \vee \exists s \in S\ \exists l \in Final\ S_n = (s, l)) = 1$
- $Pr(sched^+(\sharp)_n = \tau_n$
  $\mid S_n = \bot \vee \exists s \in S\ \exists l \in Final\ S_n = (s, l)) = 1$
- For every $s \in S, l \in L \setminus Final$ and $e \notin enabled(s)$,
  $Pr(sched^+(e)_n = \infty \mid S_n = (s, l)) = 1$
- For every $s \in S, l \in L \setminus Final$ and $e \in enabled(s)$,
  $Pr(sched^+(e)_n = sched(e)_n \mid$
  $S_n = (s, l) \wedge sched(e)_n \neq \infty) = 1$
- For every $s \in S, l \in L \setminus Final$, $e \in enabled(s)$ and $\delta \in \mathbb{R}^+$,
  $Pr(\tau_n \leq sched^+(e)_n \leq \tau_n + \delta \mid$
  $S_n = (s, l) \wedge sched(e)_n = \infty) = delay(S_n, e)(\delta)$

Given a valuation of clocks ν, a state s of the DESP and a non final location l we define the effect of time elapsing by:
$Elapse(\nu, s, l, \delta) = \nu'$ where $\forall k\ \nu'(x_k) = \nu(x_k) + flow_k(l)(s)\delta$
We also introduce the autonomous delay $Delay(\nu, l)$ by:

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

$$\wedge (s, Elapse(\nu, s, l, \delta)) \models \gamma)$$

Observe that $Delay(\nu, s, l) \in \mathbb{R}^+ \cup \{\infty\}$.
Assume $Delay(\nu, s, l) < \infty$ and denote $l \xrightarrow{\gamma,\sharp,U} l'$ the corresponding transition (i.e. with $(s, Elapse(\nu, s, l, Delay(\nu, s, l))) \models \gamma$ and $s \models \Lambda(l)$).

- $Update_{\sharp}(\nu, s, l)$ is the clock valuation defined by: $U(s, Elapse(\nu, s, l, Delay(\nu, s, l)))$
- $starget(s, \sharp, l, \nu)$ is (s, l')

Now τ_{n+1} is defined by:
$\tau_{n+1} = \min(\min(sched^{+}(e)_n \mid e \in E), \tau_n + Delay(\nu, s, l))$

We are now in position to define inductive definition. We start with the case of an autonomous transition. For every $s \in S$ and $l \in L \setminus Final$,

- $Pr(e_{n+1} = \sharp \mid \tau_{n+1} = \tau_n + Delay(\nu_n, s, l)) = 1$
- $Pr(S_{n+1} = starget(s, \sharp, l, \nu_n) \mid S_n = (s, l) \wedge e_{n+1} = \sharp) = 1$
- $Pr(\nu_{n+1} = Update_{\sharp}(\nu_n, s, l) \mid e_{n+1} = \sharp) = 1$

Similarly we introduce the functions $Update_e(\nu, s, l)$ and $starget(s, e, l, \nu)$.

- If there exists $l \xrightarrow{\gamma, E', U} l'$ such that $e \in E'$, $(s, \nu) \models \gamma$ and $target(s, e) \models \Lambda(l')$ then $starget(s, e, l, \nu) = (target(s, e), l')$ and $Update_e(\nu, s, l) = U(s, \nu)$.
- Otherwise $starget(s, e, l, \nu) = \bot$ and $Update_e(\nu, s, l)$ is not defined.

The (random) family $\{E'_n\}_{n \in \mathbb{N}}$ denotes the set of events with minimal schedule: $E'_n = \{e \in E \mid \forall e' \in E, sched^{+}(e)_n \leq sched^{+}(e')_n\}$.

- For every $e \in E$, $s \in S$ and $l \in L \setminus Final$, $Pr(e_{n+1} = e \mid S_n = (s, l) \wedge e \in E'_n \wedge \tau_{n+1} < \tau_n + Delay(\nu_n, s, l)) = choice(s, E'_n)(e)$
- For every $e \in E$, $s \in S$ and $l \in L \setminus Final$, $Pr(S_{n+1} = starget(s, e, l, Elapse(\nu_n, \tau_{n+1} - \tau_n)) \mid S_n = (s, l) \wedge e_{n+1} = e) = 1$
- For every $e \in E$, $s \in S$ and $l \in L \setminus Final$, $Pr(\nu_{n+1} = Update_e(Elapse(\nu, s, l, \tau_{n+1} - \tau_n), s, l) \mid S_n = (s, l) \wedge e_{n+1} = e \wedge S_{n+1} \neq \bot) = 1$

It remains to define the new schedules.

- For every $e \in E$, $Pr(sched(e)_{n+1} = \infty \mid e = e_{n+1}) = 1$
- For every $e \in E$, $s \in S$ and $l \in l$, $Pr(sched(e)_{n+1} = \infty \mid S_{n+1} = (s, l) \wedge e \notin enabled(s) = 1$
- For every $e \in E$, $s \in S$ and $l \in l$, $Pr(sched(e)_{n+1} = sched^{+}(e)_n \mid S_{n+1} = (s, l) \wedge e \in enabled(s) \wedge e \neq e_n) = 1$

We start the induction as follows. $Pr(\tau_0 = 0) = 1$. For every $s \in S$ such that there exists $l \in Init$ with $s \models \Lambda(l)$, $Pr(S_0 = (s, l)) = \pi_0(s)$; for $l' \neq l$, $Pr(S_0 = (s, l')) = 0$. Let us denote $S' = \{s \in S \mid \forall l \in \ s \not\models l\}$, then $Pr(S_0 = \bot) = \sum_{s \in S'} \pi_0(s)$. For every $e \in E$, $Pr(sched(e)_0 = \infty) = 1$. For every clock x_k, $Pr(\nu(x_k)_0 = 0) = 1$.