CSL\textsuperscript{LHA}: an Expressive Language for Statistical Verification of Stochastic Models

Paolo Ballarini  Hilal Djafri  Marie Duflot  Serge Haddad  Nihal Pekergin

Octobre 2010
TR–LACL–2010–8
Laboratory of Algorithmics, Complexity and Logic (LACL)  
University Paris 12 (Paris Est)  

P. BALLARINI, H. DJAFRI, M. DUFLOT, S. HADDAD, N. PEKERGIN, 

CSL$^LHA$: an Expressive Language for Statistical Verification of Stochastic Models

© Paolo Ballarini, Hilal Djafri, Marie Duflot, Serge Haddad, Nihal PEKERGIN Octobre 2010.
CSL\textsuperscript{LHA}: an Expressive Language for Statistical Verification of Stochastic Models \footnote{This work is supported by a french research project CHECKBOUND, ANR-06-SETI-002}

Paolo Ballarini\textsuperscript{1}  Hilal Djafri\textsuperscript{2}  Marie Duflot\textsuperscript{1}  Nihal Pekergin\textsuperscript{1}
\textsuperscript{1} LACL, University of Paris-Est (Paris 12),
61 avenue Général de Gaulle 94010, Créteil, France
\textsuperscript{2} LSV, ENS Cachan,
61, av. de Président Wilson, 94235 CACHAN, France
paolo.ballarini@u-pec.fr, djafri@lsv.ens-cachan.fr, marie.duflot@u-pec.fr
haddad@lsv.ens-cachan.fr, nihal.pekergin@u-pec.fr

Abstract
We introduce a new temporal logic formalism, named CSL\textsuperscript{LHA}, for the verification of discrete event stochastic processes (DESP). Being inspired by existing stochastic logic, such as, the Continuous Stochastic Logic (CSL) and its following action-state (asCSL) and timed automata (CSL\textsuperscript{TA}) evolutions, CSL\textsuperscript{LHA} extends them in two respects: firstly it targets a much broader class of stochastic models (i.e. DESP rather than Markov chains); secondly it increases the expressiveness of verification by allowing to capture sophisticated dynamics expressed in terms of multivariate conditions. A formula of CSL\textsuperscript{LHA} consists of a Linear Hybrid Automaton (LHA) and an expression. The LHA selects prefixes of relevant execution paths of a DESP and collects information along the paths. The result of the formula verification is the evaluation of an expression of the considered path random variable. A statistical engine is then employed for the confidence-interval estimation of the expected value of the expression associated to an LHA. We illustrate the CSL\textsuperscript{LHA} approach by means of some examples and experimental results obtained through a prototype software tool for CSL\textsuperscript{LHA} 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 really different. However, in the mid of the nineties, classical temporal logics have been adapted to express properties of Markov chains and 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 combinatorial 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 [HR06] 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. Here we introduce a new formalism called CSL_{LHA} where a formula evaluates as a real number and 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 logics as shown later in the examples. From a syntactical point of view, CSL_{LHA} is a CSL like formalism and more precisely is based on CSL_{TA} [DHS09]. A formula of CSL_{LHA} 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 clock (i.e. the clock’s evolution) depends on the model’s states. The expression is based on moments of path random variables associated with path executions. These variables are obtained by operators like time integration on data variables.

We illustrate the expressiveness of CSL_{LHA} on a small example modeling a shared memory system. Finally we describe the statistical verification tool COSMOS that we have developed and we present some experiments. 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 considered generic stochastic model, followed by the description of our logic in section 3. Then we give an overview of the related work, comparing the other logics with CSL_{LHA} in terms of expressiveness. In section 4, we recall the basic principles of statistical model checking and we detail the tool COSMOS for verification of CSL_{LHA} formula against GSPN by giving some results of experimentations. Finally in section 5, we conclude and give some perspectives.

2 Discrete Event Stochastic Process

We first proceed with the description of the generic probabilistic model that we consider. The goal of this genericity is for a potential user of this logic to be able to model systems with the high level formalism he is used to (such as process algebra, Petri nets, Markov chains and others), and which will fit in DESP. Our model is similar to the one introduced by Alur, Coucourbetis and Dill in [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 (Discrete Event Stochastic Process) A DESP is a tuple \( D = (S, \pi_0, E, \text{Ind}, \text{enabled}, \)
target, delay, choice) where:

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

Before giving the formal semantic of a DESP, we informally describe its items. Given a state $s$, $\text{enabled}(s)$ is the set of events enabled in $s$. For an event $e \in \text{enabled}(s)$, $\text{target}(s,e)$ denotes the target state reached from $s$ on occurrence of $e$ and $\text{delay}(s,e)$ is the distribution of the delay between the enabling of $e$ and its possible occurrence. Furthermore if we denote $E' \subseteq \text{enabled}(s)$ the set of events with same earliest delay in some configuration of the process with state $s$, $\text{choice}(s,E')$ describes how the conflict is randomly resolved: for all $e' \in E'$, $\text{choice}(s,E')(e')$ is the probability that $e'$ will be selected among $E'$. We define the subset $\text{Prop} \subseteq \text{Ind}$ of state propositions taking values in $\{0,1\}$. The sets $\text{Ind}$ and $\text{Prop}$ 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.** 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).

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

A *configuration* of a DESP is described as a triple $(s,\tau,\text{sched})$ with $s$ being the current state, $\tau \in \mathbb{R}^+$ the current time and $\text{sched} : E \rightarrow \mathbb{R}^+ \cup \{+\infty\}$ 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 $(s_0,0,\text{sched}_0)$, with $\text{sched}_0(e) = +\infty$, for all $e \in E$ and $s_0$ chosen according to the initial distribution $\pi_0$. If $(s,\tau,\text{sched})$ is the current configuration then the execution of a DESP is a succession of steps where each step consists of:

- For all $e \in \text{enabled}(s)$ such that $\text{sched}(e) = +\infty$ generate $\delta$, a sample from distribution $\text{delay}(s,e)$ and set $\text{sched}(e) := \tau + \delta$. 
• 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 $e' \notin enabled(s)$.

The above description clearly shows that the evolution of a DESP is suitable for discrete event simulation. Observe that several high-level formalisms for describing stochastic processes, like Stochastic Petri Nets (SPN) and Stochastic Process Algebras (SPA), can be mapped on the class of DESP we have introduced here. Our tool COSMOS uses generalized stochastic Petri nets to represent the probabilistic models. A marking of the net thus corresponds to a state, the events change the state of the process through firing a transition. A transition of the net has three attributes: (1) a distribution which randomly determines the delay before firing it, (2) a priority which selects among the transitions scheduled the soonest, the ones that can be selected (3) a weight, that is used in the random choice between transitions scheduled the soonest with the highest priority.

An example of such a stochastic Petri net, inspired from [AMBC+95], is given in figure 1. It will be used in the rest of the paper to introduce properties and to be verified using our tool COSMOS. It describes the behavior of an open system where two classes of processes (namely 1 and 2) want to access a shared memory (resource). One or more tokens (corresponding to processes) can appear in request mode $Request_1$ or $Request_2$ where they wait for the memory to be free (a token in place $Free$) and then a process gets the exclusive access to the memory. When the job is done, the process releases the memory and leaves the system. Observe that the two intermediate transitions have 0-Dirac distribution and when both are enabled, the competition is always won by the one that has highest priority or in case of equal priority is randomly solved according to their weights. All the other transitions have continuous distributions, so weights and priorities are not necessary since the probability that another transition has an identical schedule is null. The example illustrates the variety of distributions supported by the tool COSMOS. The topmost transitions have an exponential distribution with associated rate 1/10 and 1/12, the transitions to reach the critical sections are immediate transitions and the bottom-most transitions have a uniform distribution on a given interval.

![Figure 1: The SPN description of a shared memory system.](image-url)
3 CSL\textsuperscript{LHA}

We intuitively describe the syntax and semantics of CSL\textsuperscript{LHA} before formally defining them in the next subsections. A formula of CSL\textsuperscript{LHA} consists in two parts:

- The first component of a formula is a hybrid automaton that synchronizes with an infinite timed execution 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 formula is a restriction of hybrid automata [ACHH93], namely synchronized Linear Hybrid Automata (LHA), which extends the Deterministic Timed Automata (DTA) used in CSL\textsuperscript{TA} to describe properties of Markov chain models [DHS09]. Simply speaking LHA are automata whose set of locations is associated with a \( n \)-tuple \( X \) of real-valued variables (called data variables) 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 \( \# \) 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 \( \text{flow} \) 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 \( \text{flow}_i(l)(s) \) (where \( \text{flow}_i(l) \) is the \( i \)th component of \( \text{flow}(l) \)). Our model also uses constraints, which describe the conditions under which an edge will 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 \( \sum_{1 \leq i \leq n} \alpha_i x_i + c < 0 \) where \( \alpha_i \) and \( c \) are indicators (\textit{i.e.} in \( \text{Ind} \)), \( < \) stands for either \( =, <, >, \leq \) or \( \geq \). The set of constraints is denoted by Const. Given a location and a state, an expression of the form \( \sum_{1 \leq i \leq n} \alpha_i x_i + c \) 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 \( \lfloor \text{Const} \rfloor \) the set of left closed constraints that are used for the “autonomous” edges (\textit{i.e.} those labelled by \( \# \)). 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 \( u_1, \ldots, u_n \) where each \( u_k \) is of the form \( u_k(X) = \sum_{1 \leq i \leq n} \alpha_i x_i + c \) where \( \alpha_i \in \text{Ind} \) and \( c \) are indicators. The set of updates is denoted by Up.

**Definition 2 (Synchronized Linear Hybrid Automaton)** A synchronized linear hybrid automaton \( (LHA) A = \langle E, L, \Lambda, \text{Init}, \text{Final}, X, \text{flow}, \to \rangle \) comprises:

- \( E \), a finite alphabet of events;
- \( L \), a finite set of locations;
- \( \Lambda : L \to \text{Prop} \), a location labelling function;
• For all sequences, a subset of $L$ called the initial locations;
• Final, a subset of $L$ called the final locations;
• $X = (x_1, \ldots, x_n)$ a n-tuple of data variables;
• a function flow : $L \mapsto \text{Ind}^n$ which associates with each location one indicator for each data variable representing the evolution rate of the variable in this location. flow, denotes the projection of flow on its $i$th component.

$\rightarrow \subseteq L \times ((\text{Const} \times 2^E) \cup (\text{IConst} \times \{\#\})) \times \text{Up} \times L$, 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 $\#$ are called autonomous. Furthermore $A$ fulfills the following conditions.

- **Initial determinism:** $\forall l \neq l' \in \text{Init}, \Lambda(l) \land \Lambda(l') \Leftrightarrow \text{false}$
- **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 $l'' \xrightarrow{\gamma',E_2,U} l \land l' \not\rightarrow \gamma',E_2,U'$, then either $\Lambda(l) \land \Lambda(l') \Leftrightarrow \text{false}$ or $\gamma \land \gamma' \Leftrightarrow \text{false}$.
- **No $\#$-labelled loops:** For all sequences $l_0 \xrightarrow{\gamma_0,E_0,U_0} l_1 \xrightarrow{\gamma_1,E_1,U_1} \ldots \xrightarrow{\gamma_n,E_n,U_n} l_n$ such that $l_0 = l_n$, there exists $i \leq n$ such that $E_i \neq \#$

**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 verify the desired property. In order to ensure that the first time instant at which a constraint is satisfied exists, we require that the constraints on autonomous transitions to be left closed. The automata we consider are deterministic: given a path $\sigma$ of a DESP, there is exactly one synchronization with the linear hybrid automaton. This constraint ensures the synchronized system is still 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 autonomous events at some state of the synchronization.

**Notations.** A valuation $\nu$ maps every data variable to a real value. The value of data variable $x_i$ in $\nu$ is denoted $\nu(x_i)$. Let us fix a valuation $\nu$ and a state $s$. Given an expression $\text{exp} = \sum_{1 \leq i \leq n} \alpha_i x_i + c$ related to variables and indicators, its interpretation w.r.t. $\nu$ and $s$ is defined by $\text{exp}(s, \nu) = \sum_{1 \leq i \leq n} \alpha_i(s) \nu(x_i) + c(s)$. 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 $\gamma \equiv \text{exp} < 0$ be a constraint, we write $(s, \nu) \models \gamma$ if $\text{exp}(s, \nu) < 0$. Let $\varphi$ be a state proposition we write $s \models \varphi$ if $\varphi(s) = \text{true}$.

**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. We now describe how a path $\sigma = s_0 \xrightarrow{e_0, \gamma_0} s_1 \xrightarrow{e_1, \gamma_1} \ldots$ of a DESP $D$ is synchronized with a LHA $A$.

First there are two kinds of configurations for a synchronization:
• Non final configurations. \((s_i, l, \nu, \tau'_i)\) with \(l \notin Final\), \(s_i \models \Lambda(l)\) and \(0 \leq \tau'_i \leq \tau_i\).

• Final configurations. \((s_i, l, \nu, \tau'_i)\) with \(l \in Final\), \(s_i \models \Lambda(l)\) and \(0 \leq \tau'_i \leq \tau_i\) or the implicit final rejecting configuration \(\perp\).

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

An autonomous transition \(l \xrightarrow{\nu,l} l'\) is fireable after letting elapse \(\delta\) if \((s_i, \nu') \models \gamma\) and \(s_i \models \Lambda(l')\). In this case the configuration potentially reached is \((s_i, l', U(s_i, \nu'), \tau'_i + \delta)\). 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 \(\delta \in [0, \tau_i - \tau'_i]\), we fix \(\delta = \tau_i - \tau'_i\) and we consider the synchronized transitions \(l \xrightarrow{\gamma,l',U} l'\) such that \(e_i \in E', (s_i, \nu') \models \gamma\) and \(s_{i+1} \models \Lambda(l')\). Due to the determinism of \(A\), there is at most one such transition. If there exists one, we reach the configuration \((s_{i+1}, l', U(s_i, \nu'), 0)\). Otherwise we reach the final configuration \(\perp\).

It remains to start the synchronization. If there is some \(l_0 \in \text{Init}\) 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 \(l \in \text{Init}\) 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 never reaching a final configuration. We discuss this point in the next section.

Example. The two LHA of figure 2 are meant to synchronize with the shared memory system of figure 1. The first one is designed to decide whether there exists an instant, in an interval \([\alpha, \beta]\), at which the resource has been used longer by the first class of processes than by the second class. Three variables are used. \(x_1\) represents time (with flow 1) and omitted for concision. Variable \(x_2\) and \(x_3\) respectively represent the time during which class 1 (resp. 2) has had access to the resource. They thus have rate 0 or 1 in each location depending on whether this class has or not access to the resource (propositions \(\text{noacc}\), \(\text{acc}_1\) and \(\text{acc}_2\)). The second example uses indicator dependent flows and updates. \(x_1\) counts the cumulated waiting time of processes of class 1 before \(k\) of them have been served. The flow \(\text{nbreq}_1\) corresponds to the number of tokens in place \(\text{Request}_1\) in the current marking whereas event \(\text{Serv}_1\) 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 \(\text{Serv}_1\).

### 3.2 CSL\(^{\text{LHA}}\) expressions

As already stated, the formula consists in a linear hybrid automaton and an expression related to the automaton. Let us describe this expression. A path random variable \(Y\) is defined by an operator over a data variable \(x\). \(\text{A priori}\) we do not want to fix the operators. So we give below a non exhaustive list of relevant operators.

• \(Y \equiv \text{last}(x)\) meaning that we take the last value of \(x\) along the synchronizing path.

• \(Y \equiv \text{min}(x)\) (resp. \(Y \equiv \text{max}(x)\)) meaning that we take the minimal (resp. maximal) value of \(x\) along the synchronizing path.
These path variables are combined with standard arithmetic operators +, \(-\), *, / and with the expectation operator \( E(\cdot) \) in such a way that every occurrence of a variable in the expression occurs in a subformula whose external operator is \( E \). For example, given a random path variable \( Y \), we can consider as an expression the variance of \( Y \) defined by \( E(Y^2) - E(Y)^2 \).

Example. Referring to the LHA of figure 2 (leftmost) we can consider path random variables such as \( Y = last(x_2 - x_3) \) (the final difference of memory usage), or \( Y = avg(x_2 - x_3) \) (the average along paths of such a difference). For estimating the probability of accepted paths we introduce an extra variable which we conventionally denote success: such variable is initialized to 0, killed 0 in every location and is updated to 1 only through edges leading to an accepting location. Thus the expected probability to accept paths is the expectation of \( Y \) for \( \text{success} \). With respect to the second LHA of figure 2 (rightmost) 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.

Semantics. Given \( D \) a DESP and \((A, \exp)\) a CSL\(^{LHA}\) formula, we assume that with probability \( 1 \), the synchronizing path generated by a random execution path of \( D \) reaches a final state. This semantical assumption can be ensured by structural properties of \( A \) and/or \( D \). 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. Due to this assumption, the random path variables are well defined and the expression 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 \((A, \exp)\) for \( D \).

3.3 Expressiveness of CSL\(^{LHA}\)

In this subsection we first give an overview of related logics. Then we discuss the expressiveness of CSL\(^{LHA}\) 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 CSL has been introduced and the decidability of the verification problem over a finite continuous-time Markov chain (CTMC) has been established. This logic is a variant of CTL* where the path quantifiers All and Exists are replaced by a probabilistic operator $P_x \phi$ which returns true if the probability $p$ for a random path to fulfill the path formula $\phi$ verifies $p < r$. In this work, the path formulas are nested Until with time interval constraints. While the problem is decidable, it has a high complexity and has never been implemented. So in [BHHK03], an efficient approximate numerical procedure has been proposed based on transient analysis of Markov chains. However, path formulas are now restricted to a single Until or a single Next limiting the expressive power of the logic.

CSRL. In the logic CSRL introduced by [HCH+02], 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 this logic, 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. 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 now 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 a random path 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.

Summarizing, none of the previous work deals with general stochastic discrete-event systems. In our approach, we overcome the limitations of other logics which were necessary to be solved with numerical methods. For instance, in these logics the stochastic processes are finite Markov chains and the clocks involved in formulas have the same rate. Using our two-phase approach (synchronizing a random path with an hybrid automaton and estimating expectations over the synchronizing path), we extend the expressiveness of existing logics with a model that still can be handled by statistical model checking.

Among these logics, CSRL and DTA are the more expressive ones and are incomparable. In order to compare them with CSL$^{LHA}$, we first discuss some features of these logics. The nesting of probabilistic operators, present in all of them, 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 CSL$^{LHA}$. 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 our model, we not only have 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 CSL\(^{LHA}\), we can state that when we omit nesting and steady state properties, CSL\(^{LHA}\) is at least as expressive as CSRL and DTA: *Every non nested transient CSRL or DTA formula can be expressed with CSL\(^{LHA}\).*

**Unifying approach.** The expression associated with the formula evaluates as a real number depending on some expressions on the data variables conditioned by acceptance of the path. Observe that this generalizes the standard approach where we focus on whether the final state is accepting or not and the expression is simply the mean value of this variable. Our work thus gives a unified framework both for model-checking and for performance and dependability studies.

**Distributions.** We do not restrict ourselves to a particular kind of distributions, such as exponential. The definition of a DESP is fully generic (hence the need for the choice function when two events are scheduled at the exact same time) and the tool accepts various distributions such as uniform, deterministic, exponential, hyper-exponential, hypo-exponential or normal truncated.

**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 sophisticated updates 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 where only one state reward function was considered.

## 4 COSMOS: a software tool for CSL\(^{LHA}\) verification

We have been developing a software tool for CSL\(^{LHA}\) verification, named COSMOS. In the following we describe its main features and present results referring to the verification of the examples discussed in previous sections.

**Tool input formalisms.** The tool takes three inputs: a DESP, expressed in terms of an (extended) GSPN model \(\mathcal{N}\), an LHA \(\mathcal{A}\), representing the random variable of interest and an expression \(e\) based on moments of the random variable represented by \(\mathcal{A}\). The tool outcome is an estimation of the value of \(e\) obtained through repeated sampling of the considered random variable where each sample is obtained by execution of an independent simulation run of the \(\mathcal{N} \times \mathcal{A}\) product process. We recall that path random variables associated with an LHA can be either Bernoulli’s (\(last(success)\)) or generic (e.g. \(last(x^2)\), \(max(x_3)\) etc). In this respect, the statistical verification supported by COSMOS generalize the method featured by popular probabilistic model checkers, such as PRISM and APMC, which support estimation of Bernoulli random variables only. At present no graphical user interface is supported thus both the model \(\mathcal{N}\) and the automaton \(\mathcal{A}\) are described in textual form.

**Implementation details.** The COSMOS tool is implemented in C++ and uses the Boost libraries for generating the random numbers necessary for stochastic simulation. 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, however 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 \(dist_t, pri_t \in \mathbb{N} \cup \{\infty\}\) and \(weight_t \in \mathbb{R}_{>0}\) are, respectively, the delay-distribution, the priority and the weight of \(t\)) becomes enabled and the corresponding
event $e_t = (d_t, pri_t, w_t)$ is created with $d_t, w_t \in \mathbb{R}^+$ generated 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 $(d_t = d_{t'} \land pri_t > pri_{t'})$ or $(d_t = d_{t'} \land pri_t = pri_{t'} \land w_t < w_{t'})$. 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.

4.1 Experiments

We describe some experiments performed with the COSMOS tool. First we describe a simple experiment targeted at verifying the validity of the results calculated by the tool. Then we illustrate more complex examples aimed at demonstrating the expressiveness of the language. All experiments reported have been performed with the following estimation parameters: $confidence = 0.999$ and $approximation = 0.01$ (where approximation is the width of the confidence interval).

![Model and experimental results for the accuracy example](image)

**Figure 3:** Model and experimental results for the accuracy example

*Accuracy testing example.* In order to assess the accuracy of numerical results obtained through COSMOS we consider the finite state variant of the shared-memory system with 2 processes (Figure 3(a)) consisting of 6 exponentially distributed transitions. Such closed system corresponds to a 8 states CTMC whose steady-state distribution can be computed analytically straightforwardly as a function of the transition rates. Through a smaller (2 variables) variant of the LHA of Figure 2, we estimate $E[last(x_2)\mid x_1 \geq T]$ which estimates the expected occupation of the resource by process 1 within a given time bound $x_1 = T$. $E[last(x_2)\mid x_1 \geq T]/T$ is an estimator of the probability that the resource is occupied by process 1 at the steady-state (i.e. $\pi(\text{Access}_1 = 1)$). The plot in Figure 3(b) depict the exact values of $\pi(\text{Access}_1 = 1)$ (as a function of the rate $\mu_2$ of transition $T_{\text{End}2}$) together with the confidence interval estimation of $E[last(x_2)\mid x_1 \geq T]/T$ obtained through COSMOS. They show a very good accuracy of COSMOS.

---

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

2 given that $T$ is chosen large enough so that the transient period of the systems has passed
Other examples. The following results refer to the verification of properties of the shared memory open system of figure 1 by means of the LHA of figure 2 (leftmost). Such automata accepts those paths such that the occupation of the resource by class 1 processes has been longer than by class 2 (acceptance condition). Table 1 shows a number of conditional expectation formulae that we have considered in our experiments. All experiments consider time-bounded executions with bounding interval \([\alpha, 2\alpha]\) \((\alpha \in \mathbb{R}^+)\). Figure 4(a) and Figure 4(b) portray plots corresponding to experiments performed with respect to formulae \(f_1\), respectively \(f_4\) of Table 1. Specifically Figure 4(a) depicts the probability that class 1 processes occupy the resource longer than class 2’s as a function of time-bound interval \((\alpha)\) and of the weight of the immediate transition \(Start_2 (w_2)\). In these experiments both immediate transitions \(Start_1\) and \(Start_2\) have equal priority \((pri_1 = pri_2 = 1)\) but the weights are unbalanced in favor of class 2 processes (the weight of \(Start_1\) is kept constant \(w_1 = 1\)). Plots in Figure 4(a) show (as expected) that the probability of the resource being occupied longer by class 1 processes decreases as the probability that class 2 processes win the contention \((w_2)\) increases. Plots in Figure 4(b) instead refer to the estimation of the average along a path \((avg(x_2 - x_3))\) (positive) discrepancy between the occupation of the resource by class 1 process respectively class 2’s. They show that the expected discrepancy of occupation tends to zero as the contention gets biased in favor of classes 2 processes\(^3\)

\[
\begin{array}{|l|l|}
\hline
\text{id} & \text{estimated quantity} & \text{description} \\
\hline
f_1 & E[success(x_2 > x_3) \land (x_1 \in [\alpha, 2\alpha])] & \text{Expected probability that Res has been used longer by class-1 processes within } t \in [\alpha, 2\alpha] \\
\hline
f_2 & E[last(x_2)][(x_2 > x_3) \land (x_1 \in [\alpha, 2\alpha])] & \text{Expected occupation time of Res by class-1 processes when Res has been used longer by class-1 processes within } t \in [\alpha, 2\alpha] \\
\hline
f_3 & E[max(x_2 - x_3)][(x_2 > x_3) \land (x_1 \in [\alpha, 2\alpha])] & \text{Expected maximum occupation time divergence when Res has been used longer by class-1 processes within } t \in [\alpha, 2\alpha] \\
\hline
f_4 & E[avg(x_2 - x_3)][(x_2 > x_3) \land (x_1 \in [\alpha, 2\alpha])] & \text{Expected average of occupation time divergence when Res has been used longer by class-1 processes within } t \in [\alpha, 2\alpha] \\
\hline
f_5 & E[var(x_2 - x_3)][(x_2 > x_3) \land (x_1 \in [\alpha, 2\alpha])] & \text{Expected variability of occupation time divergence when Res has been used longer by class-1 processes within } t \in [\alpha, 2\alpha] \\
\hline
\end{array}
\]

Table 1: Examples of CSL\(^{\text{LHA}}\) formulae referred to the open shared-memory system with 2 classes of processes

Related work. We briefly summarize the most popular statistical model checkers emphasizing the applied statistical techniques. Younes et al. have proposed and refined the approach based on the sequential acceptance sampling and discrete event simulation but with a focus on time bounded CSL until formula [HR06]. The Ymer tool is based on these works [You05]. The model checking of PCTL and CSL formulas through hypothesis testing is done in the VESTA tool in [SVA05]. Note that statistical model checking based on hypothesis testing, such as with Ymer and VESTA, is concerned with establishing whether or not the probability of a formula \(\phi\) satisfies a a certain threshold i.e. \(Pr(\phi) \sim p (\sim \in \{<, \leq, \geq, >\})\) rather than with estimating the \(Pr(\phi)\). Tools supporting estimation based statistical model checking include APMC, MRMC and PRISM. In APMC tool [HLP06], LTL formulas are considered through probability estimates by means of the Chernoff-Hoeffding bounds. The MRMC tool [KZH+09] is enriched with simulation based verification tool with confidence in-

\(^3\)see Appendix A for larger formats of plots and extra experiment concerning of \(f_3: E[max(x_2 - x_3)][(x_2 > x_3) \land (x_1 \in [\alpha, 2\alpha])]\).
(a) probability that class 1 processes occupy the resource longer than class 2’s

(b) average along path discrepancy between class 1 and class 2 occupation time

Figure 4: Acceptance-probability and average-discrepancy as a function of binding interval $[\alpha, 2\alpha]$ and transitions weight $w_2$

terval based statistical guarantees [KZ09]. The PRISM model checker [PRI] also supports statistical verification of CSL formulae, based on the same approach of APMC.

5 Conclusion

We have presented a new logic for expressing elaborated properties related to stochastic processes. Contrary to previous approaches, a formula of CSL$_{LHA}$ 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.
References


A Figures

A.1 verification of formula \( f_1 \) \( E[success|(x_2 > x_3) \land (x_1 \in [\alpha, 2\alpha])] \)

Expected probability that shared resource is occupied by class 1 processes longer than by class 2’s.

![Figure 5: probability that class 1 processes occupy the resource longer than class 2’s](image)

A.2 verification of formula \( f_4 \) \( E[avg(x_2 - x_3)| (x_2 > x_3) \land (x_1 \in [\alpha, 2\alpha])] \)

Expected occupation time of \( Res \) by class-1 processes when \( Res \) has been used longer by class-1 processes within \( t \in [\alpha, 2\alpha] \)

![Figure 6: average along path discrepancy between class 1 and class 2 occupation time](image)

A.3 verification of formula \( f_3 \) \( E[max(x_2 - x_3)| (x_2 > x_3) \land (x_1 \in [\alpha, 2\alpha])] \)

Expected maximum occupation time divergence when \( Res \) has been used longer by class-1 processes within \( t \in [\alpha, 2\alpha] \)
A.4 Accuracy testing: estimation of $E[\text{last}(x_2)|(x_1 \geq T)]/T$

Figure 7: average along path discrepancy between class 1 and class 2 occupation time

Figure 8: Exact vs Estimated steady-state probability of occupation of $Res$ by process 1