Performability Measure Specification: Combining CSRL and MSL

Alessandro Aldini[1], Marco Bernardo[1], and Jeremy Sproston[2]

[1] Università di Urbino “Carlo Bo” – Italy
[2] Università di Torino – Italy

Abstract. An integral part of the performance modeling process is the specification of the performability measures of interest. The notations proposed for this purpose can be grouped into classes that differ from each other in their expressiveness and usability. Two representative notations are the continuous stochastic reward logic CSRL and the measure specification language MSL. The former is a stochastic temporal logic formulating quantitative properties about states and paths, while the latter is a component-oriented specification language relying on a first-order logic for defining reward-based measures. In this paper, we combine CSRL and MSL in order to take advantage of the expressiveness of the former and the usability of the latter. To this aim, we develop a unified notation in which the core logic of MSL is employed to set up the reward structures needed in CSRL, whereas the measure definition mechanism of MSL is exploited to formalize measure and property specification patterns in a component-oriented fashion.

1 Introduction

The performance modeling process comprises two tasks: describing system evolution over time and specifying performability measures. The former task can be handled with a wide range of mature formalisms ranging from low level ones like continuous-time Markov chains (CTMC) and queueing networks to higher level and general-purpose ones like stochastic automata, stochastic Petri nets, and stochastic process calculi (see, e.g., [8, 7] for a survey).

The latter task consists of expressing the metrics on the basis of which the system model should be assessed at steady state or at a given time instant or interval. The traditional way of doing this is to construct a reward structure [16] by associating real numbers with the states and the transitions of the CTMC underlying the system model. The reward assigned to a state determines the rate at which a gain/loss is accumulated while sojourning in that state, whereas the reward assigned to a transition determines the instantaneous gain/loss implied by the execution of that transition. The value of the measure is then computed as the sum of state probabilities and transitions frequencies, where rewards are used as weights.

In order to cope with the higher level of abstraction of general-purpose system modeling formalisms, more recent approaches to performability measure specification instead resort to a mixture of rewards and logics in a manner that avoids any direct intervention on the underlying CTMC (see, e.g., [4, 3, 1] and the references therein). In this paper, we focus on two representatives of the above mentioned approaches, which are CSRL and MSL, with the aim of combining their expressiveness and usability, thus enabling the modeler to take advantage of their complementary strengths.

CSRL [4, 3] is a stochastic temporal logic suitable for expressing properties on system descriptions with an underlying CTMC semantics. These properties can be state-based, like, e.g., stating whether the steady-state probability of being in states satisfying a certain condition is in a specified relation with a given threshold, and path-based, like, e.g., stating whether the probability of observing paths satisfying a certain condition is in a specified relation with a given threshold.

MSL [1], which is composed of a core logic and a measure definition mechanism, has been conceived to support the specification of performability measures for descriptions of component-based systems with an underlying CTMC semantics. The core logic of MSL defines reward structures by assigning real numbers to states and transitions of the CTMC. Each state of the CTMC is viewed as a vector of local states representing the current behavior of the various components, and can be given a reward either directly on the basis of its constituent local states or indirectly on the basis of the activities labeling its outgoing transitions. The measure definition mechanism of MSL enhances usability by means of a component-oriented level on top of the core logic. The idea is to employ such a higher level to set up libraries of measure definitions that are provided by performability experts, which could then be exploited by nonexperts too.

Both MSL and CSRL suffer from some limitations. On the one hand, MSL is suitable for specifying classical measures such as throughput, utilization, queue length, and response time, but it is not adequate for formulating path properties. On the other hand, CSRL requires familiarity with temporal logic and supports neither the component-oriented formalization of performability measures nor the definition of policies for stating the association of rewards to states, thus complicating the work for nonexperts.

As a solution to such drawbacks, in this paper we propose UMSL, a unified measure specification language arising from the combination of two adequate extensions of MSL and CSRL. The objective is to enable the modeler to take advantage of the usability of the former and the expressiveness of the latter in a single notation. In order to make it affordable the specification of performability measures also by nonspecialists, the development of the unified language aims at a clear separation of concerns by means of (i) a core logic for setting up both reward structures and logical formulas and (ii) a measure definition mechanism for expressing performability measures on top of the core logic.

In the remainder of this paper, we introduce finite labeled CTMCs as a reference model for the assessment of the performability of component-based systems (Sect. 2), we present the core logic of UMSL (Sect. 3), we illustrate the measure definition mechanism of UMSL, which is an extension of that of MSL (Sect. 4), and finally we provide some concluding remarks (Sect. 5).

2 Reference Model

According to the guidelines of [2], the description of a component-based system should comprise at least the description of the individual system component types and the description of the overall system topology. The description of a single component type should be provided by specifying at least its name, its parameters, its behavior, and its interactions. The overall behavior should express all the alternative sequences of activities that the component type can carry out – which can be formalized by means of traditional tools like automata, Petri nets, and process calculi – while the interactions are those activities used by the component type to communicate with the rest of the system.

For performability evaluation purposes, we assume that from the description of a component-based system it is possible to extract a stochastic model in the form of a CTMC. Since the overall behavior of each component can be observed through the activities that are executed, every transition of the CTMC is labeled not only with its rate $\lambda \in \mathbb{R}_{>0}$, but also with an action taken from a set Act. Likewise, every state of the CTMC is labeled with the vector of local states denoting the current behaviors of the $N \in \mathbb{N}_{>0}$ components in the system, with each local state belonging to a set Loc. In this way, the stochastic model reflects the component-oriented nature of the system description. Formally, our reference model is a finite labeled CTMC:

$$\mathcal{M} \;=\; (S, T, L, N, Loc, Act)$$

where S is a finite set of states, $T \subseteq S \times Act \times \mathbb{R}_{>0} \times S$ is a finitely-branching action-labeled transition relation, and $L : S \to Loc^N$ is an injective state-labeling function.

For notational convenience, $s \xrightarrow{a,\lambda}_{\mathcal{M}} s'$ denotes the transition $(s, a, \lambda, s') \in T$ and $L(s) = [z_1, z_2, \dots, z_N]$ the vector of local states labeling $s \in S$. We use the notation $z \in s$ (resp. $z \notin s$) to express that $z = z_i$ for some $1 \leq i \leq N$ (resp. $z \neq z_i$ for all $1 \leq i \leq N$). Moreover, we denote with $negLoc$ (resp. $negAct$) the set of negations $\bar{z}$ (resp. $\bar{a}$) of local states $z \in Loc$ (resp. activities $a \in Act$). Negation is simply a shorthand for expressing that a component is not in a given local state or cannot execute a given activity. Finally, we define predicate sat by letting:

$$s \; sat \; z \text{ iff } z \in s \qquad\qquad s \; sat \; a \text{ iff } \exists \lambda \in \mathbb{R}_{>0}, s' \in S.\, s \xrightarrow{a,\lambda}_{\mathcal{M}} s'$$

$$s \; sat \; \bar{z} \text{ iff } z \notin s \qquad\qquad s \; sat \; \bar{a} \text{ iff } \nexists \lambda \in \mathbb{R}_{>0}, s' \in S.\, s \xrightarrow{a,\lambda}_{\mathcal{M}} s'$$

which is extended to conjunctions and disjunctions of local states and activities in the expected way. As an example, we have that $s \; sat \; z_1 \wedge z_2$ iff $s \; sat \; z_1$ and $s \; sat \; z_2$.

Throughout the paper, we use C as metavariable for components, B for specific current behaviors, a for activities, and the dot notation $C.B$ (resp. $C.a$) to express component behaviors (resp. component activities).

Example 1. Let us consider as a running example a system with two identical servers P_1 and P_2 that process requests arriving at the system with rate $\lambda \in \mathbb{R}_{>0}$. When a request finds both servers busy, it must immediately leave the system; i.e., no buffer is present. When a request finds both servers idle, it has the same probability to be accepted by the two servers. The two servers process incoming requests at rates $\mu_1, \mu_2 \in \mathbb{R}_{>0}$, respectively. They can also fail with rates $\chi_1, \chi_2 \in \mathbb{R}_{>0}$, respectively, and are then repaired with rates $\varrho_1, \varrho_2 \in \mathbb{R}_{>0}$, respectively.

Assuming that the arrival process has a single local state – $Arrivals$ – and that each server has three local states – $Idle$, $Busy$, and $Failed$ – the labeled CTMC underlying this system is as shown in Fig. 1, with the initial state being the leftmost one. For instance, the label $P_1.Idle$ in the initial state denotes that server P_1 is initially idle. ■

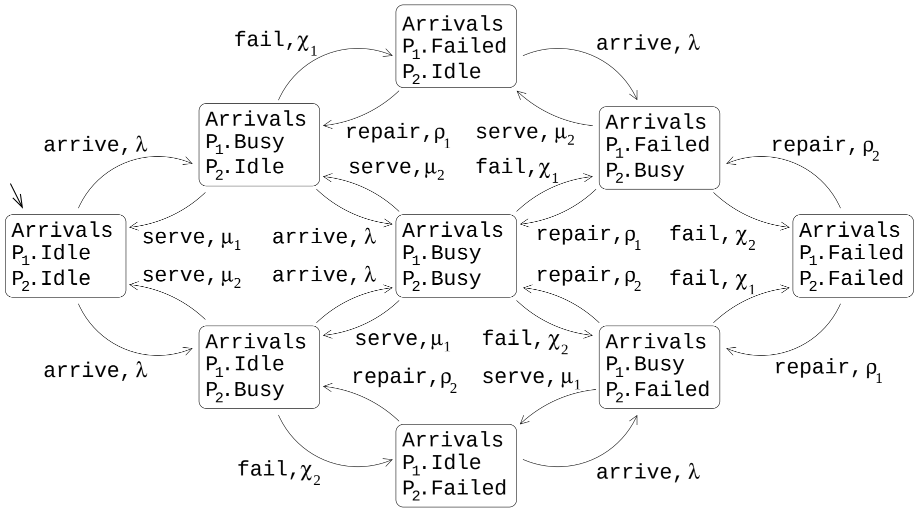

Fig. 1. Labeled CTMC for the running example

3 Core Logic of UMSL

In order to complement the expressiveness of CSRL with the capability of MSL of defining reward structures, in this section we combine the two formalisms into a single logic that will be the core of UMSL. First, we define an extension of MSL (Sect. 3.1) and an extension of CSRL (Sect. 3.2), then we show how they are combined (Sect. 3.3).

3.1 Extending MSL Parameterization

Each formula of the core logic of MSL is a first-order predicate parameterized with respect to a set of local states or activities that contribute to the value of a performability measure [1]. The role of the predicate is to specify how the contributions are combined to set the reward gained while sojourning in each state. In order to generalize the parameterization mechanism, we introduce an extension of MSL, called dnfMSL, in which each predicate is defined with respect to a set of groups of local states or activities expressed in disjunctive normal form for the sake of uniform treatment and usability.

The set *dnfLoc* of disjunctive normal forms on local states comprises elements like:

$$Z = (z_{1,1} \wedge \ldots \wedge z_{1,m_1}) \vee \ldots \vee (z_{n,1} \wedge \ldots \wedge z_{n,m_n})$$

which, to hide connectives to nonexperts, can be abbreviated with $Z = \{Z_1, \ldots, Z_n\}$, where $Z_i = \{z_{i,1}, \ldots, z_{i,m_i}\}$ and each literal $z_{i,j}$ occurring in conjunct Z_i is either a local state or the negation of a local state. In practice, conjunction expresses that a certain group of local states is needed for a state to gain a reward, while disjunction establishes that alternative conjuncts may contribute to assign a reward to the state.

Similarly, the set *dnfAct* of disjunctive normal forms on activities comprises elements such as:

$$A = (a_{1,1} \wedge \ldots \wedge a_{1,m_1}) \vee \ldots \vee (a_{n,1} \wedge \ldots \wedge a_{n,m_n})$$

which can be abbreviated with $A = \{A_1, \dots, A_n\}$, where $A_i = \{a_{i,1}, \dots, a_{i,m_i}\}$ and each literal $a_{i,j}$ occurring in A_i is either an activity or the negation of an activity.

We note that the use of conjunction constitutes a novelty with respect to the parameterization mechanism of [1] that enhances the expressive power. The case $m_i = 1$ for all $1 \leq i \leq n$, in which each conjunct is a singleton, corresponds to an original set parameter for MSL formulas of [1].

We now define four formula schemas that specify the way in which the contributions provided by each element in the set parameter are combined to establish a reward. These formulas arise from the combination of the type of elements forming the set parameter (local states or activities) with the fact that each (or only one) conjunct in the set parameter that is satisfied contributes to the reward. The predicates and functions that will be used subsequently are as follows:

- $\mathit{eq} : \mathbb{R} \times \mathbb{R} \rightarrow \{\text{true}, \text{false}\}$ such that:
$$\mathit{eq}(x, y) \;=\; \begin{cases} \text{true} & \text{if } x = y \\ \text{false} & \text{otherwise} \end{cases}$$
- $\mathit{state\_rew} : S \rightarrow \mathbb{R}$ such that $\mathit{state\_rew}(s)$ is the reward accumulated while sojourning in state s due to either local states of s or activities enabled by s.
- $\mathit{lstate\_rew} : \mathit{Loc} \cup \mathit{negLoc} \rightarrow \mathbb{R}$ such that $\mathit{lstate\_rew}(z)$ is the reward contribution given by local state z and $\mathit{lstate\_rew}(\bar{z})$ is the reward contribution given by the negation of local state $\bar{z}$.
- $\mathit{act\_rew} : \mathit{Act} \cup \mathit{negAct} \rightarrow \mathbb{R}$ such that $\mathit{act\_rew}(a)$ is the reward contribution given by activity a and $\mathit{act\_rew}(\bar{a})$ is the reward contribution given by the negation of activity $\bar{a}$.

The values $\mathit{lstate\_rew}(\_)$ and $\mathit{act\_rew}(\_)$ should be defined by the modeler depending on the measure of interest: in Sect. 4, we will show how to make such an assignment as transparent as possible. Then, each of the following formula schemas states that for a given set parameter the reward $\mathit{state\_rew}(\_)$ results from a specific combination of the contributions provided by local states or activities occurring in the set parameter.

The first formula schema assigns to $s \in S$ a direct state reward, to which all the conjuncts of local states in a set Z that are satisfied by s contribute in an additive way. The contribution of each conjunct $Z_i \in Z$ that is satisfied is obtained by combining the rewards associated with the literals occurring in Z_i through a function $\mathit{af} : 2^{\mathbb{R}} \rightarrow \mathbb{R}$ – like, e.g., sum, min, max, and avg – taken from a set AF of associative and commutative arithmetical functions. The first formula schema asserts the following for each $s \in S$:

$$\boxed{\mathit{eq}(\mathit{state\_rew}(s), \mathit{sum\_lstate\_contrib}(s, Z, \mathit{af}))}$$

where Z and af are given and $\mathit{sum\_lstate\_contrib} : S \times \mathit{dnfLoc} \times AF \rightarrow \mathbb{R}$ is such that:

$$\mathit{sum\_lstate\_contrib}(s, Z, \mathit{af}) \;=\; \sum_{Z_i \in Z \text{ s.t. } s \; \mathit{sat} \; Z_i} \mathit{af}\{\!| \; \mathit{lstate\_rew}(z) \mid z \in Z_i \; |\!\}$$

which is zero if there is no $Z_i \in Z$ satisfied by s.

The second formula schema assigns to $s \in S$ a direct state reward, to which only one among the conjuncts of local states in a set Z that are satisfied by s contributes. The contribution is selected among the contributions of those conjuncts by applying a function $\mathit{cf} : 2^{\mathbb{R}} \rightarrow \mathbb{R}$ – like, e.g., max and min – taken from a set CF of choice functions; i.e., $cf(\emptyset) = 0$ and $cf(\{x_1, \ldots, x_n\}) \in \{x_1, \ldots, x_n\}$ for all $n \in \mathbb{N}_{>0}$. Formally, the second formula schema asserts the following for each $s \in S$:

$$\boxed{eq(state\_rew(s), choose\_lstate\_contrib(s, Z, af, cf))}$$

where cf is given too and $choose\_lstate\_contrib : S \times dnfLoc \times AF \times CF \longrightarrow \mathbb{R}$ is such that:

$$choose\_lstate\_contrib(s, Z, af, cf) \;=\; \mathop{cf}_{Z_i \in Z \text{ s.t. } s \; sat \; Z_i} af\{\!|\; lstate\_rew(z) \mid z \in Z_i \;|\!\}$$

The third formula schema assigns to $s \in S$ an indirect state reward, to which all the conjuncts of activities in a set A that are satisfied by s contribute in an additive way. Formally, the third formula schema asserts the following for each $s \in S$:

$$\boxed{eq(state\_rew(s), sum\_act\_contrib(s, A, af))}$$

where A and af are given and $sum\_act\_contrib : S \times dnfAct \times AF \longrightarrow \mathbb{R}$ is such that:

$$sum\_act\_contrib(s, A, af) \;=\; \sum_{A_i \in A \text{ s.t. } s \; sat \; A_i} af\{\!|\; act\_rew(a) \mid a \in A_i \;|\!\}$$

which is zero if there is no $A_i \in A$ satisfied by s.

The fourth formula schema assigns to $s \in S$ an indirect state reward, to which only one among the conjuncts of activities in a set A that are satisfied by s contributes. Formally, the fourth formula schema asserts the following for each $s \in S$:

$$\boxed{eq(state\_rew(s), choose\_act\_contrib(s, A, af, cf))}$$

where cf is given too and $choose\_act\_contrib : S \times dnfAct \times AF \times CF \longrightarrow \mathbb{R}$ is such that:

$$choose\_act\_contrib(s, A, af, cf) \;=\; \mathop{cf}_{A_i \in A \text{ s.t. } s \; sat \; A_i} af\{\!|\; act\_rew(a) \mid a \in A_i \;|\!\}$$

Example 2. The system throughput for the running example can be determined by assigning to each state an indirect state reward obtained by summing up the rates of the $serve$ activities enabled in that state, as shown by this formula of the third schema:

$$eq(state\_rew(s), sum\_act\_contrib(s, \{\{P_1.serve\}, \{P_2.serve\}\}, \mathrm{sum}))$$

with $act\_rew(P_i.serve) = \mu_i$ for $i = 1, 2$. It is also possible to define the same measure by using direct state rewards as in the following formula of the first schema:

$$eq(state\_rew(s), sum\_lstate\_contrib(s, \{\{P_1.Busy\}, \{P_2.Busy\}\}, \mathrm{sum}))$$

with $lstate\_rew(P_i.Busy) = \mu_i$ for $i = 1, 2$.

The system utilization can be specified by assigning to each state a unitary reward if the state includes at least one server in the $Busy$ local state, as specified by the following formula of the second schema:

$$eq(state\_rew(s), choose\_lstate\_contrib(s, \{\{P_1.Busy\}, \{P_2.Busy\}\}, \mathrm{min}, \mathrm{min}))$$

with $lstate\_rew(P_i.Busy) = 1$ for $i = 1, 2$.

As a variant of the previous measure, consider the utilization of server P_1 whenever server P_2 is not in the busy state, which can be defined as follows:

$$eq(state\_rew(s), choose\_lstate\_contrib(s, Z, \mathrm{min}, \mathrm{min}))$$

with $Z = \{\{P_1.Busy, P_2.Idle\}, \{P_1.Busy, P_2.Failed\}\}$, $lstate\_rew(P_1.Busy) = 1$, and $lstate\_rew(P_2.Failed) = lstate\_rew(P_2.Idle) = 1$. Alternatively, by using negation we have $Z = \{\{P_1.Busy, \overline{P_2.Busy}\}\}$ and $lstate\_rew(\overline{P_2.Busy}) = 1$. ■

We conclude by noting that, as done in [1], analogous formula schemas could be easily included in dnfMSL that assign rewards to transitions rather than states. However, they would not enhance the expressiveness of the logic.

3.2 Extending CSRL with Actions

The stochastic temporal logic CSRL expresses both state properties and path properties based on conditions over the states traversed along a path [4, 3]. In an action-based, component-oriented setting, it is convenient to extend the temporal operators of CSRL with actions. The resulting action-based logic, which we call aCSRL, is actually a combination of CSRL and aCSL [14]. In contrast to aCSL, in aCSRL we allow reference to states. As in CSRL, the syntax of aCSRL features bounds on both the time and the reward accumulated along a path, while, as in aCSL, the syntax of aCSRL can make reference to actions exhibited along the path.

The syntax of the state formulas of aCSRL is as follows:

$$\boxed{\Phi \ ::= \ Z \mid A \mid \Phi \wedge \Phi \mid \neg\Phi \mid \mathcal{S}_{\bowtie p}(\Phi) \mid \mathcal{P}_{\bowtie p}(\varphi) \mid \mathcal{E}_J(\Phi) \mid \mathcal{E}^t_J(\Phi) \mid \mathcal{C}^I_J(\Phi)}$$

where $Z \in \mathit{dnfLoc}$ is a disjunctive normal form on local states, $A \in \mathit{dnfAct}$ is a disjunctive normal form on activities, $\bowtie \in \{<, \leq, \geq, >\}$ is a comparison operator, $p \in \mathbb{R}_{[0,1]}$ is a probability, $t \in \mathbb{R}_{\geq 0}$ is a time value, I is an interval of time values, J is an interval of real-valued rewards, and φ is a path formula (see below).

State formulas are built from: disjunctive normal forms on local states or activities; logical conjunction and negation; the steady-state operator $\mathcal{S}_{\bowtie p}(\Phi)$, which establishes whether the steady-state probability of being in states that satisfy Φ is in relation $\bowtie$ with the threshold p; the probabilistic operator $\mathcal{P}_{\bowtie p}(\varphi)$, which establishes whether the probability of taking paths that satisfy φ is in relation $\bowtie$ with the threshold p; and the expected reward operators $\mathcal{E}_J(\Phi)$, $\mathcal{E}^t_J(\Phi)$, and $\mathcal{C}^I_J(\Phi)$ [4, 3], which establish whether the reward accumulated at steady state or at a given time instant t or interval I while sojourning in states that satisfy a certain state formula Φ is in interval J.

Example 3. The formula $\mathcal{E}_{[0,7]}(P_1.\mathit{Failed})$ is true in a state if the expected reward accumulated in the long run in states in which $P_1.\mathit{Failed}$ holds is not greater than 7. The formula $\mathcal{E}^{10}_{[2,5]}(P_2.\mathit{Busy})$ is true in a state if the expected reward accumulated at time 10 in states in which $P_2.\mathit{Busy}$ holds is between 2 and 5. The formula $\mathcal{C}^{[0,15]}_{[8,\infty)}(\mathrm{true})$ is true in a state if the expected reward accumulated before 15 time units is at least 8. ■

The syntax of path formulas is as follows:

$$\boxed{\varphi \ ::= \ \Phi \ {}_{\mathcal{A}}U^{<t}_{<r} \ \Phi \mid \Phi \ {}_{\mathcal{A}_1}U^{<t}_{<r} \, {}_{\mathcal{A}_2} \, \Phi}$$

where $\mathcal{A}, \mathcal{A}_1, \mathcal{A}_2 \subseteq \mathit{Act}$ are sets of actions, $t \in \mathbb{R}_{\geq 0} \cup \{\infty\}$ is a time value, and $r \in \mathbb{R} \cup \{\infty\}$ is a reward value.

The until formula $\Phi_1 \ {}_{\mathcal{A}}U^{<t}_{<r} \ \Phi_2$ is satisfied by an execution path if the path visits a state satisfying Φ_2 within t time units, while accumulating at most r reward, and visits states satisfying Φ_1 while performing only actions in $\mathcal{A}$ until that point. Similarly, the until formula $\Phi_1 \ {}_{\mathcal{A}_1}U^{<t}_{<r} \, {}_{\mathcal{A}_2} \, \Phi_2$ is satisfied by a path if the path visits a state satisfying Φ_2 within t time units, while accumulating at most r reward, after performing an action in $\mathcal{A}_2$, and visits states satisfying Φ_1 while performing only actions in $\mathcal{A}_1$ until that point. Note that a path satisfying $\Phi_1 \ {}_{\mathcal{A}_1}U^{<t}_{<r} \, {}_{\mathcal{A}_2} \, \Phi_2$ actually makes a transition to a state satisfying Φ_2, whereas this is not required in the case of $\Phi_1 \ {}_{\mathcal{A}}U^{<t}_{<r} \ \Phi_2$ (if the initial state of the path satisfies Φ_2).

Example 4. The formula $\mathcal{P}_{\leq 0.02}((P_1.\mathit{Idle} \vee P_1.\mathit{Busy})\ {}_{\mathcal{A}}U^{\leq 30}_{\leq 7}\ {}_{\{\mathit{fail}\}}\ P_1.\mathit{Failed})$ is true in a state if, with probability at most 0.02, there is a point along the path at which the state property $P_1.\mathit{Failed}$ holds directly after performing any fail action, at which no more than 30 time units have elapsed and at most 7 units of reward have been accumulated, and for which $P_1.\mathit{Idle} \vee P_1.\mathit{Busy}$ holds at all preceding points, and the actions in $\mathcal{A} = \{\mathit{arrive}, \mathit{serve}, \mathit{repair}\}$ are the only actions seen on the path before the action fail. Intuitively, the formula establishes whether certain conditions are met whenever along the path server P_1 fails for the first time (note that $P_1.\mathit{Idle} \vee P_1.\mathit{Busy}$ could be replaced by $\neg P_1.\mathit{Failed}$). ■

We restrict our attention to until formulas featuring upper bounds on time and accumulated rewards for simplicity (note that aCSL has been presented only with respect to upper bounds on time in [14], and that practical techniques for model-checking CSRL formulas feature upper bounds only in [3]).

Before presenting the semantics of aCSRL with respect to the reference model $\mathcal{M} = (S, T, L, N, \mathit{Loc}, \mathit{Act})$, we introduce some notation. For each state $s \in S$, we let the exit rate of s be defined by $\mathrm{E}(s) = \sum\{\!|\, \lambda \in \mathbb{R}_{>0} \mid s \xrightarrow{a,\lambda}_{\mathcal{M}} s' \,|\!\}$. A state s is called absorbing if and only if $\mathrm{E}(s) = 0$. If $s \xrightarrow{a,\lambda}_{\mathcal{M}} s'$ and $t \in \mathbb{R}_{\geq 0}$, then we say that there exists a step of duration t from state s to state s' with action a, denoted by $s \xrightarrow{a,t} s'$. An infinite path is a sequence $s_0 \xrightarrow{a_0,t_0} s_1 \xrightarrow{a_1,t_1} \cdots$ of steps. A finite path is a sequence $s_0 \xrightarrow{a_0,t_0} s_1 \xrightarrow{a_1,t_1} \cdots \xrightarrow{a_{n-1},t_{n-1}} s_n$ of steps such that s_n is absorbing.

Let $\mathit{Path}^{\mathcal{M}}$ be the set of paths of $\mathcal{M}$ and let $\mathit{Path}^{\mathcal{M}}(s)$ be the subset of paths that commence in state s. For any state $s \in S$, let $\mathit{Prob}_s^{\mathcal{M}}$ denote the probability measure over the measurable subsets of $\mathit{Path}^{\mathcal{M}}(s)$ [5]. For any infinite path $\omega = s_0 \xrightarrow{a_0,t_0} s_1 \xrightarrow{a_1,t_1} \cdots$ and any $i \in \mathbb{N}$, let $\omega(i) = s_i$, the $(i+1)$st state of ω, let $\delta(\omega, i) = t_i$, and, for $t \in \mathbb{R}_{\geq 0}$ and i the smallest index such that $t \leq \sum_{j=0}^{i} t_j$, let $\omega@t = \omega(i)$. For $\mathcal{A} \subseteq \mathit{Act}$, let $s_i \xrightarrow{\mathcal{A}} s_{i+1}$ be a predicate which is true if and only if $a_i \in \mathcal{A}$. For any finite path $\omega = s_0 \xrightarrow{a_0,t_0} s_1 \xrightarrow{a_1,t_1} \cdots \xrightarrow{a_{l-1},t_{l-1}} s_l$, the state $\omega(i)$ and duration $\delta(\omega, i)$ are defined only if $i \leq l$, and are defined as in the infinite-path case (apart from $\delta(\omega, l)$, which equals ∞). Furthermore, for $t \geq \sum_{j=0}^{l-1} t_j$, let $\omega@t = \omega(l)$; otherwise, $\omega@t$ is defined as in the infinite-path case. Similarly, predicates of the form $s_i \xrightarrow{\mathcal{A}} s_{i+1}$ are defined only if $i < l$, and are defined as in the infinite-path case.

A transient probability is the probability of being in a certain state s' at time t given an initial state s. In the model-checking context, we can express a transient probability in terms of paths as $\pi^{\mathcal{M}}(s, s', t) = \mathit{Prob}_s^{\mathcal{M}}\{\omega \in \mathit{Path}^{\mathcal{M}}(s) \mid \omega@t = s'\}$. The steady-state probabilities are used to refer to the long-run average probability of the CTMC being in a state, and are defined by $\pi^{\mathcal{M}}(s, s') = \lim_{t\to\infty} \pi^{\mathcal{M}}(s, s', t)$. For $S' \subseteq S$, let $\pi^{\mathcal{M}}(s, S', t) = \sum_{s' \in S'} \pi^{\mathcal{M}}(s, s', t)$ and $\pi^{\mathcal{M}}(s, S') = \sum_{s' \in S'} \pi^{\mathcal{M}}(s, s')$.

Given a reward structure $\rho : S \to \mathbb{R}$, let the instantaneous reward $\rho^{\mathcal{M}}(s, s', t) = \pi^{\mathcal{M}}(s, s', t) \cdot \rho(s')$ and the expected long-run reward $\rho^{\mathcal{M}}(s, s') = \pi^{\mathcal{M}}(s, s') \cdot \rho(s')$. For $S' \subseteq S$, let $\rho^{\mathcal{M}}(s, S', t) = \sum_{s' \in S'} \rho^{\mathcal{M}}(s, s', t)$ and $\rho^{\mathcal{M}}(s, S') = \sum_{s' \in S'} \rho^{\mathcal{M}}(s, s')$. For an infinite path $\omega = s_0 \xrightarrow{a_0,t_0} s_1 \xrightarrow{a_1,t_1} \cdots$ and $i \in \mathbb{N}$, let $\gamma(\omega, i) = t_i \cdot \rho(s_i)$.

$s \models_{\mathcal{M}} Z$	iff	$s\ sat\ Z$
$s \models_{\mathcal{M}} A$	iff	$s\ sat\ A$
$s \models_{\mathcal{M}} \mathit{\Phi}_1 \wedge \mathit{\Phi}_2$	iff	$s \models_{\mathcal{M}} \mathit{\Phi}_1$ and $s \models_{\mathcal{M}} \mathit{\Phi}_2$
$s \models_{\mathcal{M}} \neg\mathit{\Phi}$	iff	$s \not\models_{\mathcal{M}} \mathit{\Phi}$
$s \models_{\mathcal{M}} \mathcal{S}_{\bowtie p}(\mathit{\Phi})$	iff	$\pi^{\mathcal{M}}(s, \mathit{Sat}^{\mathcal{M}}(\mathit{\Phi})) \bowtie p$
$s \models_{\mathcal{M}} \mathcal{P}_{\bowtie p}(\varphi)$	iff	$\mathit{Prob}_s^{\mathcal{M}}\{\omega \in \mathit{Path}^{\mathcal{M}} \mid \omega \models_{\mathcal{M}} \varphi\} \bowtie p$
$s \models_{\mathcal{M}} \mathcal{E}_J(\mathit{\Phi})$	iff	$\rho^{\mathcal{M}}(s, \mathit{Sat}^{\mathcal{M}}(\mathit{\Phi})) \in J$
$s \models_{\mathcal{M}} \mathcal{E}_J^t(\mathit{\Phi})$	iff	$\rho^{\mathcal{M}}(s, \mathit{Sat}^{\mathcal{M}}(\mathit{\Phi}), t) \in J$
$s \models_{\mathcal{M}} \mathcal{C}_J^I(\mathit{\Phi})$	iff	$\int_I \rho^{\mathcal{M}}(s, \mathit{Sat}^{\mathcal{M}}(\mathit{\Phi}), u)du \in J$
$\omega \models_{\mathcal{M}} \mathit{\Phi}_1\ {}_{\mathcal{A}}U_{<r}^{<t}\ \mathit{\Phi}_2$	iff	$\exists k \geq 0.$
		$(\omega(k) \models_{\mathcal{M}} \mathit{\Phi}_2 \wedge$
		$(\forall i < k.\, \omega(i) \models_{\mathcal{M}} \mathit{\Phi}_1 \wedge \omega(i) \xrightarrow{\mathcal{A}} \omega(i+1)) \wedge$
		$t \geq \sum_{i=0}^{k-1} \delta(\omega, i) \wedge r \geq \sum_{i=0}^{k-1} \gamma(\omega, i))$
$\omega \models_{\mathcal{M}} \mathit{\Phi}_1\ {}_{\mathcal{A}_1}U_{<r}^{<t}\ {}_{\mathcal{A}_2}\mathit{\Phi}_2$	iff	$\exists k > 0.$
		$(\omega(k) \models_{\mathcal{M}} \mathit{\Phi}_2 \wedge$
		$(\forall i < k-1.\, \omega(i) \models_{\mathcal{M}} \mathit{\Phi}_1 \wedge \omega(i) \xrightarrow{\mathcal{A}_1} \omega(i+1)) \wedge$
		$\omega(k-1) \models_{\mathcal{M}} \mathit{\Phi}_1 \wedge \omega(k-1) \xrightarrow{\mathcal{A}_2} \omega(k) \wedge$
		$t \geq \sum_{i=0}^{k-1} \delta(\omega, i) \wedge r \geq \sum_{i=0}^{k-1} \gamma(\omega, i))$
where $\mathit{Sat}^{\mathcal{M}}(\mathit{\Phi}) = \{s \in S \mid s \models_{\mathcal{M}} \mathit{\Phi}\}$.		

Table 1. Semantics of aCSRL

After enriching the reference model $\mathcal{M}$ with a reward structure ρ on states only, the formal semantics of aCSRL is given by the satisfaction relation $\models_{\mathcal{M}}$ defined in Table 1.

3.3 Intertwining aCSRL and dnfMSL

The objective of UMSL is to combine aCSRL and dnfMSL in order to join their complementary advantages. In fact, on the one hand, dnfMSL is not expressive enough to establish that a state is given a reward only if it satisfies a complex condition formalized by a temporal logic formula that includes not only logical connectives. On the other hand, aCSRL is intended to specify logical properties but it does not help the modeler to understand which rewards must be attached to every state for any occurrence of until and expected reward operators.

In order to overcome these drawbacks, we propose two different ways of combining aCSRL formulas and dnfMSL formula schemas, which result in two intertwined notations, called dnfMSL+ and aCSRL+, that constitute the core of UMSL. Formally, a formula of UMSL can be a dnfMSL+ formula schema ν or an aCSRL+ formula $\mathit{\Phi}$, such that:

$$
\begin{array}{rl}
\nu ::= & \mathit{eq}(\mathit{state\_rew}(s), \mathit{sum\_lstate\_contrib}_{\mathit{\Phi}}(s, Z, \mathit{af})) \\
\mid & \mathit{eq}(\mathit{state\_rew}(s), \mathit{choose\_lstate\_contrib}_{\mathit{\Phi}}(s, Z, \mathit{af}, \mathit{cf})) \\
\mid & \mathit{eq}(\mathit{state\_rew}(s), \mathit{sum\_act\_contrib}_{\mathit{\Phi}}(s, A, \mathit{af})) \\
\mid & \mathit{eq}(\mathit{state\_rew}(s), \mathit{choose\_act\_contrib}_{\mathit{\Phi}}(s, A, \mathit{af}, \mathit{cf}))
\end{array}
$$

where:

$$sum\_lstate\_contrib_{\Phi}(s, Z, af) \;=\; \begin{cases} sum\_lstate\_contrib(s, Z, af) & \text{if } s \models \Phi \\ 0 & \text{otherwise} \end{cases}$$

and with the other three functions that are defined similarly, while:

$$\boxed{\Phi \;::=\; Z \mid A \mid \Phi \wedge \Phi \mid \neg\Phi \mid \mathcal{S}_{\bowtie p}(\Phi) \mid \mathcal{P}_{\bowtie p}(\nu, \varphi) \mid \mathcal{E}_J(\nu, \Phi) \mid \mathcal{E}^t_J(\nu, \Phi) \mid \mathcal{C}^I_J(\nu, \Phi)}$$

where φ is a path formula and ν provides probabilistic and reward operators with the reward structures that are needed for their interpretation.

A formula of dnfMSL+ parameterized with respect to Φ associates with each state $s \in S$ satisfying Φ the reward defined by the underlying dnfMSL formula schema. Note that dnfMSL+ extends dnfMSL in a conservative way. Indeed, the dnfMSL+ formula schema $eq(state\_rew(s), sum\_lstate\_contrib_{\Phi}(s, Z, af))$ and the dnfMSL formula schema $eq(state\_rew(s), sum\_lstate\_contrib(s, Z, af))$ define the same reward structure whenever either $\Phi = \mathrm{true}$ or $\Phi = Z$.

A formula of aCSRL+ is an extension of a formula of aCSRL in which every subformula requiring a reward structure ρ for its interpretation is paired with a dnfMSL+ formula schema ν defining such a reward structure. The semantics of aCSRL+ is exactly as shown in Table 1 in the case of aCSRL, with the assumption that regarding the reward structure ρ we have that $\forall s \in S.\, \rho(s) = state\_rew(s)$, where $state\_rew(s)$ is the reward assigned to s by ν.

Example 5. The formula $\mathcal{E}_{[0.5,0.7]}(\nu, P_1.Busy \vee P_2.Busy)$, where ν is the dnfMSL+ formula corresponding to the third case of Ex. 2, is true in a state if the system utilization, in the long run, is in the interval $[0.5, 0.7]$. ■

Thanks to the way in which dnfMSL+ and aCSRL+ are defined, the core logic of UMSL allows for a controlled form of nesting according to which a formula schema of dnfMSL+ embeds a formula of aCSRL+, while in turn a formula of aCSRL+ may embed a formula schema of dnfMSL+. In this way, UMSL offers the same expressiveness as aCSRL, whose operators are fully integrated in UMSL. Moreover, as we will see in the next section, UMSL includes additional capabilities concerned with the use of nested reward structures.

4 Measure Definition Mechanism of UMSL

MSL is equipped with a component-oriented measure definition mechanism built on top of the core logic for enhancing its usability [1]. In this section, we show that the same mechanism can be applied on top of dnfMSL+. Moreover, we show that it can be extended to aCSRL+ formulas so as to develop a component-oriented property definition mechanism. The combination of the two mechanisms makes the specification of UMSL formulas an easier task with respect to using CSRL, especially for people not familiar with (temporal) logics. The syntax for specifying in MSL a performability measure as a macro definition possibly parameterized with respect to a set of component-oriented arguments is as follows:

`MEASURE` ◁ *name*▷ (◁*arguments*▷) `IS` ◁ *body*▷

The metric is given a symbolic name and is parameterized with respect to component behaviors and component activities to be used in its body, which is defined in terms of MSL core logic formulas in the case of a basic measure. Assuming that the identifier of a metric denotes the value of the metric computed on a certain finite labeled CTMC, it is then possible to define derived measures whose body comprises metric identifiers combined through the usual arithmetical operators and mathematical functions. The idea is that libraries of basic measure definitions should be provided by performability experts, which could then be exploited by nonexperts too upon defining derived measures. In any case, when the definition of a measure is available, the modeler is only asked to provide component-oriented parameters without having to consider which numbers have to be associated with which entities.

We now show how this mechanism is inherited by dnfMSL+. The syntax for defining a performability measure in UMSL is extended as follows:

$$\boxed{\texttt{MEASURE } \triangleleft\, \mathit{name} \triangleright (\triangleleft \mathit{arg1}; \mathit{arg2}; \mathit{arg3} \triangleright) \texttt{ IS } \triangleleft\, \mathit{body} \triangleright}$$

where the body of a basic measure is a dnfMSL+ formula schema ν, while the arguments are divided into three parts:

- $\mathit{arg1} ::= Z \mid A$
- $\mathit{arg2} ::= \Phi$
- $\mathit{arg3} ::= \infty \mid t \mid [t_1, t_2]$

The first two arguments represent the sequence of lists of component behaviors (or activities) parameterizing ν and the aCSRL+ formula embedded in ν, respectively. The third argument does not define the reward structure underlying the measure description. Instead, it is used for analysis purposes to specify the time at which the measure should be computed: ∞ denotes steady-state analysis, $t \in \mathbb{R}_{\geq 0}$ denotes instant-of-time analysis, and $[t_1, t_2]$ (with $t_1, t_2 \in \mathbb{R}_{\geq 0}$ and $t_1 < t_2$) denotes interval-of-time analysis.

The definition of Φ may be hard from the viewpoint of a modeler who is not familiar with temporal logics like aCSRL. For this reason, we now introduce a property definition mechanism on top of aCSRL+, which is inspired by the measure definition mechanism of MSL. The syntax for defining a property in UMSL is as follows:

$$\boxed{\texttt{PROPERTY } \triangleleft\, \mathit{name} \triangleright (\triangleleft \mathit{arguments} \triangleright) \texttt{ IS } \triangleleft\, \mathit{body} \triangleright}$$

where the body is defined as an aCSRL+ formula parameterized with respect to the arguments that are provided. Arguments are given in form of a list ℓ defined as follows:

$$\begin{aligned} \ell &::= \ell' \mid \ell', \ell \\ \ell' &::= Z \mid A \mid \bowtie p \mid t \mid r \mid I \mid J \end{aligned}$$

where Z (resp. A) is a sequence of lists of component behaviors (resp. activities) forming disjunctive normal forms, p is a probability and $\bowtie$ is a comparison operator used in steady-state and probabilistic operators, t (resp. r) is a time (resp. reward) value used in path formulas, while I (resp. J) is an interval used to specify time (resp. reward) bounds in expected reward operators.

In order to illustrate the measure definition mechanism of UMSL, let us consider three basic definitions, which will be used in the following examples.

The property determining whether a state satisfies a disjunctive normal form on local states $\{Z_1, \ldots, Z_n\}$, where $Z_i = \{z_{i,1}, \ldots, z_{i,m_i}\}$ for all $1 \leq i \leq n$, can be represented by the following definition:

$$\texttt{PROPERTY}\ \mathit{sat\_elem}(Z_1, ..., Z_n)\ \texttt{IS}\ (z_{1,1} \wedge ... \wedge z_{1,m_1}) \vee ... \vee (z_{n,1} \wedge ... \wedge z_{n,m_n})$$

The property that states whether the steady-state probability of being in a certain combination $Z \in \mathit{dnfLoc}$ of component behaviors is less than p is given by:

$$\texttt{PROPERTY}\ \mathit{ss\_beh}(Z, <p)\ \texttt{IS}\ \mathcal{S}_{<p}(Z)$$

The property establishing whether the probability of being for the first time in the component behavior $C.B$ by time t after having consumed at most an amount r of resources, with each unitary resource usage expressed by the execution of $C'.a$, is less than p, can be defined as:

$$\texttt{PROPERTY}\ \mathit{path\_beh}(C'.a, C.B, t, r, <p)\ \texttt{IS}\ \mathcal{P}_{<p}(\nu, \neg C.B\ {}_{Act}U^{\leq t}_{\leq r}\ C.B)$$

where ν is the formula $eq(\mathit{state\_rew}(s), \mathit{sum\_act\_contrib}_{\text{true}}(s, C'.a, \text{sum}))$ such that $\mathit{act\_rew}(C'.a) = 1$. This property generalizes Ex. 4 and emphasizes the support provided by dnfMSL+ to the definition of the reward structure needed by the interpretation of an until formula of aCSRL+.

Similar to the measure definition mechanism, assuming that the identifier of a property denotes the truth value of the corresponding aCSRL+ formula computed on a certain finite labeled CTMC, it is then possible to define derived properties whose body comprises property identifiers combined through the usual logical operators.

A property identifier can in turn be used as a macro in the definition of performability measures, in a way that masks the temporal logic formula that constitutes the body of the property. We illustrate this fact by considering two typical performance measures: system throughput and resource utilization.

As observed in Sect. 3.1, the definition of the throughput is parameterized with respect to the component activities $C_1.a_1, \ldots, C_n.a_n$ that contribute to the throughput. We refine this definition by assuming that a side condition – expressed by an aCSRL+ formula Φ – can be introduced to represent a guard that must be satisfied to count the activity contribution. Then the rate at which each state satisfying Φ accumulates reward is the sum of the rates of the contributing activities that are enabled at that state. If Φ is given a property definition *prop* included in a library of properties, then we have the following measure definition:

$$\begin{array}{c}\texttt{MEASURE}\ \mathit{throughput}(C_1.a_1, \ldots, C_n.a_n; \mathit{prop}(\_); \_)\ \texttt{IS} \\ eq(\mathit{state\_rew}(s), \mathit{sum\_act\_contrib}_{\mathit{prop}(\_)}(s, A, \text{sum}))\end{array}$$

where $A = \{\{C_1.a_1\}, \ldots, \{C_n.a_n\}\}$ and $\mathit{act\_rew}(C_i.a_i) = \lambda_i$ for all $1 \leq i \leq n$ if the rate associated with $C_i.a_i$ is λ_i.

Example 6. Given the basic measure definition *throughput*, the average system throughput for the running example can be determined through the following invocation:

$$\mathit{throughput}(P_1.\mathit{serve}, P_2.\mathit{serve}; \text{true}; \infty)$$

which specifies the reward structure with $\mathit{act\_rew}(P_i.\mathit{serve}) = \mu_i$ for $i = 1, 2$. If the contribution of a server activity must be counted only when the other server is under repair, then the invocation becomes:

$$\begin{array}{l}\mathit{throughput}(P_1.\mathit{serve}, P_2.\mathit{serve}; \\ \qquad \mathit{sat\_elem}(\{P_1.\mathit{Busy}, P_2.\mathit{Failed}\}, \{P_2.\mathit{Busy}, P_1.\mathit{Failed}\}); \\ \qquad \infty)\end{array}$$

■

In the case of the utilization of a resource, as seen in Sect. 3.1 we have to specify the set of component activities $C_1.a_1, \dots, C_n.a_n$ modeling the utilization of that resource, while a unit reward is transparently associated with each state in which this activity is enabled. As in the case of the system throughput, we enrich the definition by adding a side condition:

$$\texttt{MEASURE}\ \mathit{utilization}(C_1.a_1, \dots, C_n.a_n; \mathit{prop}(\_); \_)\ \texttt{IS}$$
$$\mathit{eq}(\mathit{state\_rew}(s), \mathit{choose\_act\_contrib}_{\mathit{prop}(\_)}(s, A, \mathrm{sum}, \mathrm{min}))$$

where $A = \{\{C_1.a_1\}, \dots, \{C_n.a_n\}\}$ and $\mathit{act\_rew}(C_i.a_i) = 1$ for all $1 \leq i \leq n$.

Example 7. The utilization of server P_1 by time t is specified as follows:

$$\mathit{utilization}(P_1.\mathit{serve}; \mathrm{true}; [0, t])$$

Assume to be interested in counting the use of P_1 only if the probability of observing a subsequent P_1 failure by time t' after having been used at most r times is less than p. In this case, the invocation of the basic measure definition $\mathit{utilization}$ becomes:

$$\mathit{utilization}(P_1.\mathit{serve}; \mathit{path\_beh}(P_1.\mathit{serve}, P_1.\mathit{Failed}, t', r, {<}p); [0, t]) \qquad \blacksquare$$

An interesting set of measure definitions refers to the problem of determining the probability of being in specific component behaviors. In this case, it should be enough for the modeler to specify these component behaviors in terms of $Z \in \mathit{dnfLoc}$:

$$\texttt{MEASURE}\ \mathit{beh\_prob}(Z; Z; \_)\ \texttt{IS}$$
$$\mathit{eq}(\mathit{state\_rew}(s), \mathit{choose\_lstate\_contrib}_Z(s, Z, \mathrm{min}, \mathrm{min}))$$

such that $\mathit{lstate\_rew}(z) = 1$ when z is one of the component behaviors occurring in Z.

Example 8. The probability on the long run of being in a state in which both servers are under repair or both servers are idle is determined by taking:

$$Z = \{\{P_1.\mathit{Failed}, P_2.\mathit{Failed}\}, \{P_1.\mathit{Idle}, P_2.\mathit{Idle}\}\}$$

and then by using the invocation $\mathit{beh\_prob}(Z; Z; \infty)$. $\blacksquare$

The measure $\mathit{beh\_prob}$ can be generalized to express more complex measures like:

$$\texttt{MEASURE}\ \mathit{beh\_prob}(Z; \mathit{prop}(\_); \_)\ \texttt{IS}$$
$$\mathit{eq}(\mathit{state\_rew}(s), \mathit{choose\_lstate\_contrib}_{\mathit{prop}(\_)}(s, Z, \mathrm{min}, \mathrm{min}))$$

The measure above quantifies the probability of being in states satisfying the property $\mathit{prop}(\_)$ and including a combination of component behaviors occurring in Z.

Example 9. Consider the probability of being in the component behavior $P_1.\mathit{Failed}$ at time t, provided that with probability less than p we observe again action $P_1.\mathit{fail}$ by n time units while accumulating a number of arrivals less than r. The value is given by:

$$\mathit{beh\_prob}(P_1.\mathit{Failed}; \mathit{path\_beh}(\mathit{Arrivals.arrive}, P_1.\mathit{fail}, n, r, {<}p); t)$$

Now, consider an extension in which a failure state $P_1.\mathit{Failed}$ is taken into account also in the case that the related steady-state probability of being in a total failure state – i.e., both servers are under repair – is less than q. In this case, the invocation becomes:

$$\begin{aligned}&\mathit{beh\_prob}(P_1.\mathit{Failed};\\ &\qquad \mathit{path\_beh}(\mathit{Arrivals.arrive}, P_1.\mathit{fail}, n, r, {<}p) \vee\\ &\qquad\ \mathit{ss\_beh}(\{\{P_1.\mathit{Failed}, P_2.\mathit{Failed}\}\}, < q);\\ &\qquad t)\end{aligned} \qquad \blacksquare$$

Finally, the parameterization of the measure definition mechanism of dnfMSL+ can be further extended by assuming that each literal occurring in Z (resp. A) is possibly associated with a real number expressing the reward contribution of the local state (resp. activity) to be used in the definition of function $\mathit{lstate\_rew}$ (resp. $\mathit{act\_rew}$).

For instance, the overall energy consumption with respect to the component behaviors $C.B_1, \dots, C.B_n$ is the sum of the probabilities of being in the various local states, each multiplied by a reward that describes the rate at which energy is consumed in that state. Hence, we extend the basic measure $\mathit{beh\_prob}$ to derive the following measure:

MEASURE $\mathit{energy\_consumption}(C.B_1(l_1), C.B_2(l_2), \dots, C.B_n(l_n); \text{true}; \_)$ IS
$\mathit{beh\_prob}(C.B_1(l_1); \text{true}; \_) \; + \dots + \mathit{beh\_prob}(C.B_n(l_n); \text{true}; \_)$
where $\mathit{lstate\_rew}(z) = l_i$ when $z = C.B_i$ for some $1 \leq i \leq n$.

Example 10. Assuming that the energy consumed in the busy state is 50% more than the energy consumed in the failure state, while no energy is consumed in the idle state, the overall energy consumption with respect to server P_1 in the interval $[0, t]$ is given by the following invocation of the derived measure $\mathit{energy\_consumption}$:

$\mathit{energy\_consumption}(P_1.\mathit{Idle}(0), P_1.\mathit{Busy}(3), P_1.\mathit{Failed}(2); \text{true}; [0, t]).$ ■

5 Conclusion

In this paper, we have shown how to combine two orthogonal extensions of the logics CSRL and MSL in order to obtain UMSL, a unified measure specification language trading expressiveness and usability.

From the expressiveness standpoint, the core logic of UMSL – i.e., the combination of dnfMSL+ and aCSRL+ – offers interesting features.

On the one hand, the expressiveness gain with respect to MSL core logic is twofold. Firstly, the combinations of local states and activities that contribute to the reward structure are now formalized as more expressive propositional logic formulas in disjunctive normal form. Secondly, it is possible to add conditions stating that certain states are given certain rewards only if they satisfy temporal logic formulas expressed in aCSRL+.

On the other hand, observing that aCSRL+ inherits the same expressiveness as CSL-like stochastic logics [14, 3, 18], several examples of Sect. 4 – see Exs. 7 and 9 – show that UMSL supports the definition of nested, independent reward structures. This feature is not considered in other mechanisms proposed in the literature for the definition of reward structures in the setting of stochastic model checking [10, 19, 9, 17, 18].

To complete the expressiveness analysis, it would be interesting to compare UMSL with alternative mechanisms for the specification of performance measures, like, e.g., Performance Trees, which have been proved to be more expressive than CSL [20].

Finally, from the usability standpoint, we observe that the objective of the definition mechanisms presented in Sect. 4 is to manage both reward structures and logic operators as transparently as possible, especially through the macro mechanism used for derived definitions. In this way, while basic definitions represent a task for experts, their use within derived definitions should be affordable by non-specialists. Along this line of research, it would be interesting to employ the measure and property definition mechanisms of UMSL to define specification pattern systems inspired by, e.g., ProProST [13]. Indeed, since measure and property definition mechanisms of UMSL can be combined to realize nested patterns in a component-oriented fashion, we believe that UMSL-based specification patterns would help practitioners to apply in a correct and easy way model-checking techniques for the verification of the performability of component-based systems.

Acknowledgment: This work has been funded by MIUR-PRIN project *PaCo – Performability-Aware Computing: Logics, Models, and Languages*.

References

1. A. Aldini and M. Bernardo, *"Mixing Logics and Rewards for the Component-Oriented Specification of Performance Measures"*, in Theoretical Computer Sc. 382:3–23, 2007.
2. A. Aldini, M. Bernardo, and F. Corradini, *"A Process Algebraic Approach to Software Architecture Design"*, Springer, 2010.
3. C. Baier, L. Cloth, B. Haverkort, H. Hermanns, and J.-P. Katoen, *"Performability Assessment by Model Checking of Markov Reward Models"*, in Formal Methods in System Design 36:1–36, 2010.
4. C. Baier, B. Haverkort, H. Hermanns, and J.-P. Katoen, *"On the Logical Characterisation of Performability Properties"*, in Proc. of the *27th Int. Coll. on Automata, Languages and Programming (ICALP 2000)*, Springer, LNCS 1853:780–792, 2000.
5. C. Baier, B. Haverkort, H. Hermanns, and J.-P. Katoen, *"Model-Checking Algorithms for Continuous-Time Markov Chains"*, in IEEE Trans. on Software Eng. 29:524–541, 2003.
6. C. Baier and J.-P. Katoen, *"Principles of Model Checking"*, MIT Press, 2008.
7. M. Bernardo and J. Hillston, eds., *"Formal Methods for Performance Evaluation"*, Springer, LNCS 4486, 2007.
8. E. Brinksma, H. Hermanns, and J.-P. Katoen, eds., *"Lectures on Formal Methods and Performance Analysis"*, Springer, LNCS 2090, 2001.
9. T. Courtney, D. Daly, S. Derisavi, S. Gaonkar, M. Griffith, V. Lam, and W. Sanders, *"The Möbius Modeling Environment: Recent Developments"*, in Proc. of the *1st Int. Conf. on Quantitative Evaluation of Systems (QEST 2004)*, IEEE-CS Press, pp. 328–329, 2004.
10. G. Clark and J. Hillston, *"Towards Automatic Derivation of Performance Measures from PEPA Models"*, in Proc. of the *12th UK Performance Engineering Workshop*, 1996.
11. R. De Nicola, J.-P. Katoen, D. Latella, M. Loreti, and M. Massink, *"Model Checking Mobile Stochastic Logic"*, in Theoretical Computer Science 382:42–70, 2007.
12. R. De Nicola and F. Vaandrager, *"Action Versus State Based Logics for Transition Systems"*, in Proc. of the *LITP Spring School on Theoretical Computer Science*, Springer, LNCS 469:407–419, 1990.
13. L. Grunske, *"Specification Patterns for Probabilistic Quality Properties"*, in Proc. of the *30th Int. Conf. on Software Engineering (ICSE 2008)*, ACM Press, pp. 31–40, 2008.
14. H. Hermanns, J.-P. Katoen, J. Meyer-Kayser, and M. Siegle, *"Towards Model Checking Stochastic Process Algebra"*, in Proc. of the *2nd Int. Conf. on Integrated Formal Methods (IFM 2000)*, Springer, LNCS 1945:420–439, 2000.
15. H. Hermanns, J.-P. Katoen, J. Meyer-Kayser, and M. Siegle, *"Model Checking Stochastic Process Algebra"*, Technical Rep. IMMD7-2/00, University of Erlangen-Nürnberg, 2000.
16. R.A. Howard, *"Dynamic Probabilistic Systems"*, John Wiley & Sons, 1971.
17. J.-P. Katoen, M. Khattri, and I.S. Zapreev, *"A Markov Reward Model Checker"*, in Proc. of the *2nd Int. Conf. on Quantitative Evaluation of Systems (QEST 2005)*, IEEE-CS Press, pp. 243–244, 2005.
18. M. Kwiatkowska, G. Norman, and D. Parker, *"Stochastic Model Checking"*, in *Formal Methods for Performance Evaluation*, Springer, LNCS 4486:220–270, 2007.
19. W. Obal and W. Sanders, *"State-Space Support for Path-Based Reward Variables"*, in Performance Evaluation 35:233–251, 1999.
20. T. Suto, J.T. Bradley, and W.J. Knottenbelt, *"Performance Trees: Expressiveness and Quantitative Semantics"*, in Proc. of the *4th Int. Conf. on the Quantitative Evaluation of Systems (QEST 2007)*, IEEE-CS Press, pp. 41–50, 2007.

A Model Checking Support for UMSL

For completeness, we discuss the model-checking algorithms that are needed to solve the formulas of UMSL. We first present a least fixed-point characterization of the two forms of the until operator (for non-infinite time and reward bounds) of aCSRL by combining previous work on aCSL [14] and CSRL [4, 3]. Let $s \in S$ be a state, $t, r \in \mathbb{R}_{\geq 0}$ be time and reward bounds, respectively, and $k = \min\{t, \frac{r}{\rho(s)}\}$. Note that k depends on s, t, and r; however, the value of k will be clear from the context in the following. Given states $s, s' \in S$ and set $\mathcal{A} \subseteq \mathit{Act}$ of actions, we let $\mathbf{R}_{\mathcal{A}}(s, s') = \sum_{a \in \mathcal{A}} \{\!| \lambda \in \mathbb{R}_{>0} \mid s \xrightarrow{a,\lambda}_{\mathcal{M}} s' |\!\}$. Let Φ_1, Φ_2 be aCSRL formulas and $\mathcal{A} \subseteq \mathit{Act}$. We use $p(s, t, r)$ to denote the probability:

$$\mathit{Prob}_s^{\mathcal{M}}\{\omega \in \mathit{Path}^{\mathcal{M}} \mid \omega \models_{\mathcal{M}} \Phi_1 \,{}_{\mathcal{A}}U^{<t}_{<r}\, \Phi_2\}$$

The probabilities $p(s, t, r)$ are obtained from the least solution in the interval $[0, 1]$ of the following integral equation system:

- if $s \models_{\mathcal{M}} \neg\Phi_1 \wedge \Phi_2$, then $p(s, t, r) = 1$;
- if $s \models_{\mathcal{M}} \Phi_1 \wedge \neg\Phi_2$, then $p(s, t, r) =$
$$\int_0^k e^{-\mathrm{E}(s)\cdot x} \sum_{s' \in S} \mathbf{R}_{\mathcal{A}}(s, s') \cdot p(s', t - x, r - \rho(s) \cdot x)\,dx \ ;$$
- otherwise $p(s, t, r) = 0$.

Let $\mathcal{A}_1, \mathcal{A}_2 \subseteq \mathit{Act}$. We use $p'(s, t, r)$ to denote the probability:

$$\mathit{Prob}_s^{\mathcal{M}}\{\omega \in \mathit{Path}^{\mathcal{M}} \mid \omega \models_{\mathcal{M}} \Phi_1 \,{}_{\mathcal{A}_1}U^{<t}_{<r}\,{}_{\mathcal{A}_2} \Phi_2\}$$

The probabilities $p'(s, t, r)$ are obtained from the least solution in the interval $[0, 1]$ of the following integral equation system:

- if $s \models_{\mathcal{M}} \Phi_1$ then $p'(s, t, r) =$
$$\begin{aligned} &\int_0^k e^{-\mathrm{E}(s)\cdot x} \cdot \Big(\sum_{s' \models_{\mathcal{M}} \Phi_2} \mathbf{R}_{\mathcal{A}_2}(s, s') \\ &+ \sum_{s' \models_{\mathcal{M}} \Phi_1} \mathbf{R}_{\mathcal{A}_1}(s, s') \cdot p'(s', t - x, r - \rho(s) \cdot x) \\ &- \sum_{s' \models_{\mathcal{M}} \Phi_1 \wedge \Phi_2} \mathbf{R}_{\mathcal{A}_1 \cap \mathcal{A}_2}(s, s') \cdot p'(s', t - x, r - \rho(s) \cdot x)\Big)\,dx \ ; \end{aligned}$$
- otherwise $p(s, t, r) = 0$.

After having described a UMSL formula to be used within a performance evaluation process, it remains to resolve the formula on the CTMC modeling the system in order to extract the performance indices of interest. The division of UMSL formulas into two types, namely dnfMSL+ and aCSRL+, calls for two different approaches. On the one hand, a formula of dnfMSL+ is used to assign rewards to individual states of the CTMC. On the other hand, a formula of aCSRL+ identifies a set of states of the CTMC, namely the set of states which satisfy the formula. In this section, we describe a method for the model checking of aCSRL+ formulas in order to compute such state sets.

As in traditional approaches for model checking branching-time temporal logic formulas (see [6]), we proceed bottom-up within the parse tree of the aCSRL+ formula;

therefore, when considering a given aCSRL+ formula, we assume that the set of states of its subformulas have been computed. In the context of aCSRL+, observe that, through dnfMSL+, formulas can include additional references to aCSRL+ subformulas compared to previous stochastic temporal logics such as CSL or CSRL. For example, a property Φ of the form:

$$\mathcal{P}_{\bowtie p}(eq(state\_rew(s), sum\_lstate\_contrib_{\Phi_1}(s, Z, af)), \Phi_2 \ {}_{\mathcal{A}_1}U^{<t}_{<r} \ {}_{\mathcal{A}_2} \Phi_3)$$

has three aCSRL+ subformulas, namely Φ_1, which is used within a dnfMSL+ formula to identify which states should be assigned (possibly non-zero) rewards, and Φ_2 and Φ_3, which have the standard role within the path formula $\Phi_2 \ {}_{\mathcal{A}_1}U^{<t}_{<r} \ {}_{\mathcal{A}_2} \Phi_3$. Hence, in order to obtain the set of states satisfying Φ, we first compute the sets $Sat^{\mathcal{M}}(\Phi_i)$ of states satisfying Φ_i, for $i \in \{1, 2, 3\}$.

We now describe briefly model-checking algorithms for aCSRL, which we will subsequently use as a basis for our model-checking algorithms for aCSRL+. The Boolean operators and the operators $\mathcal{S}_{\bowtie p}(\Phi)$, $\mathcal{E}_J(\Phi)$, $\mathcal{E}^t_J(\Phi)$, and $\mathcal{C}^I_J(\Phi)$ are independent of actions (given the computation of $Sat^{\mathcal{M}}(\Phi)$), and can be handled as for CSL and CSRL [3]. Furthermore, the operator $\mathcal{P}_{\bowtie p}(\Phi_1 \ {}_{\mathcal{A}}U^{<t}_{<r} \ \Phi_2)$ can be handled as for CSRL if $\mathcal{A} = Act$. Similarly, formulas of the form $\mathcal{P}_{\bowtie p}(\Phi_1 \ {}_{\mathcal{A}}U^{<t}_{<\infty} \ \Phi_2)$ and $\mathcal{P}_{\bowtie p}(\Phi_1 \ {}_{\mathcal{A}_1}U^{<t}_{<\infty} \ {}_{\mathcal{A}_2} \Phi_2)$ can be handled as for aCSL [14]. Formulas of the form $\mathcal{P}_{\bowtie p}(\Phi_1 \ {}_{\mathcal{A}}U^{<\infty}_{<r} \ \Phi_2)$ and of the form $\mathcal{P}_{\bowtie p}(\Phi_1 \ {}_{\mathcal{A}_1}U^{<\infty}_{<r} \ {}_{\mathcal{A}_2} \Phi_2)$ can be subject to the duality result of [3], followed by an application of the aCSL model-checking algorithm [14].

In the case of general aCSRL formulas of the form $\mathcal{P}_{\bowtie p}(\Phi_1 \ {}_{\mathcal{A}}U^{<t}_{<r} \ \Phi_2)$ and of the form $\mathcal{P}_{\bowtie p}(\Phi_1 \ {}_{\mathcal{A}_1}U^{<t}_{<r} \ {}_{\mathcal{A}_2} \Phi_2)$, we follow the approach of applying to models *with* action-labeled transitions verification techniques developed for models *without* actions [12]. This approach was adapted to action-based extensions of CSL in [15, 11]. We first define a *state-labeled CTMC* $\mathcal{S}$ as a tuple $(\mathsf{S}, \mathsf{T}, \mathrm{AP}, \mathsf{L})$, where S is a set of states, $\mathsf{T} \subseteq \mathsf{S} \times \mathbb{R}_{>0} \times \mathsf{S}$ is a finitely branching transition relation, AP is a set of atomic propositions, and $\mathsf{L} : \mathsf{S} \to 2^{\mathrm{AP}}$ is a state labeling function. A state-labeled CTMC, together with a reward structure $\eta : \mathsf{S} \to \mathbb{R}_{\geq 0}$ and a CSRL formula, can be used as input to the standard CSRL model-checking algorithm [3].

Let $\mathcal{M} = (S, T, L, N, Loc, Act)$ be a labeled CTMC. We now define a state-labeled CTMC corresponding to $\mathcal{M}$, for which information regarding actions is encoded within states (and their labels). We define $\mathcal{S}_{\mathcal{M},\Phi_1,\Phi_2}$ to be the state-labeled CTMC $(\mathsf{S}, \mathsf{T}, \mathrm{AP}, \mathsf{L})$ such that:

- $\mathsf{S} = \{(s, a) \in (S \times A) \mid \exists \lambda \in \mathbb{R}_{>0}, s' \in S. (s', a, \lambda, s) \in T\} \cup \{(s, \bot) \mid s \in S\}$;
- $\mathsf{T} = \{((s, a), \lambda, (s', a')) \mid (s, a', \lambda, s') \in T\}$;
- $\mathrm{AP} = Loc^N \cup Act \cup \{\Phi_1, \Phi_2\}$;
- $\mathsf{L}(s, a) = L(s) \cup \{a\} \cup \bigcup_{i \in \{1,2\}, s \models_{\mathcal{M}} \Phi_i} \{\Phi_i\}$ for all $(s, a) \in \mathsf{S}$.

Furthermore, for a reward structure ρ associated with $\mathcal{M}$, we let η be the reward structure associated with $\mathcal{S}_{\mathcal{M},\Phi_1,\Phi_2}$ such that $\eta(s, a) = \rho(s)$ for each $(s, a) \in \mathsf{S}$.

When considering "until" formulas, the standard (not action-based) CSRL model-checking algorithm applied to a state-labeled $\mathcal{S}$ consists in deciding whether $s \models_{\mathcal{S}} \mathcal{P}_{\bowtie p}(\Phi'_1 \ U^{<t}_{<r} \ \Phi'_2)$ for each state $(s, a) \in \mathsf{S}$ and some CSRL formulas Φ'_1 and Φ'_2, where $\models_{\mathcal{S}}$ is the standard semantics of CSRL [3] with respect to $\mathcal{S}$. Given an aCSRL formula Φ and $\mathcal{A} \subseteq Act$, let $\Phi^{\mathcal{A},\bot} = \Phi \wedge (\bot \vee \bigvee_{a \in \mathcal{A}} a)$ and let $\Phi^{\mathcal{A}} = \Phi \wedge \bigvee_{a \in \mathcal{A}} a$. Then the

following result can be obtained by using the reasoning employed in [15] extended with rewards.

Proposition 1. *Let $s \in S$ be a state of $\mathcal{M}$. Then:*

$$s \models_{\mathcal{M}} \mathcal{P}_{\bowtie p}(\Phi_1 \, {}_{\mathcal{A}}U^{\leq t}_{<r} \; \Phi_2) \;\Leftrightarrow\; (s, \bot) \models_{\mathcal{S}_{\mathcal{M},\Phi_1,\Phi_2}} \mathcal{P}_{\bowtie p}(\Phi_1^{\mathcal{A},\bot} \; U^{\leq t}_{<r} \; \Phi_2^{\mathcal{A},\bot})$$

and:

$$s \models_{\mathcal{M}} \mathcal{P}_{\bowtie p}(\Phi_1 \, {}_{\mathcal{A}_1}U^{\leq t}_{<r} \, {}_{\mathcal{A}_2} \Phi_2) \;\Leftrightarrow\; (s, \bot) \models_{\mathcal{S}_{\mathcal{M},\Phi_1,\Phi_2}} \mathcal{P}_{\bowtie p}(\Phi_1^{\mathcal{A}_1,\bot} \; U^{\leq t}_{<r} \; \Phi_2^{\mathcal{A}_2}) \qquad \blacksquare$$

Prop. 1 provides the following algorithm for computing *Sat*$^{\mathcal{M}}(\mathcal{P}_{\bowtie p}(\Phi_1 \, {}_{\mathcal{A}}U^{\leq t}_{<r} \; \Phi_2))$: construct $\mathcal{S}_{\mathcal{M},\Phi_1,\Phi_2}$, then apply the standard CSRL model-checking algorithm of [3] to $\mathcal{S}_{\mathcal{M},\Phi_1,\Phi_2}$ for the property $\mathcal{P}_{\bowtie p}(\Phi_1^{\mathcal{A},\bot} \; U^{\leq t}_{<r} \; \Phi_2^{\mathcal{A},\bot})$. Then let:
Sat$^{\mathcal{M}}(\mathcal{P}_{\bowtie p}(\Phi_1 \, {}_{\mathcal{A}}U^{\leq t}_{<r} \; \Phi_2)) = \{s \in S \mid (s, \bot) \models_{\mathcal{S}_{\mathcal{M},\Phi_1,\Phi_2}} \mathcal{P}_{\bowtie p}(\Phi_1^{\mathcal{A},\bot} \; U^{\leq t}_{<r} \; \Phi_2^{\mathcal{A},\bot})\}$
The model-checking algorithm for computing *Sat*$^{\mathcal{M}}(\mathcal{P}_{\bowtie p}(\Phi_1 \, {}_{\mathcal{A}_1}U^{\leq t}_{<r} \, {}_{\mathcal{A}_2} \Phi_2))$ proceeds similarly.

Finally, these algorithms for aCSRL can be adapted to provide model-checking algorithms for aCSRL+. As Boolean operators, formulas of the form Z and A and $\mathcal{S}_{\bowtie p}(\Phi)$ are independent of rewards, hence standard methods can be used in their case. Now consider, for example, the formula $\mathcal{P}_{\bowtie p}(\nu, \varphi)$. In order to compute the set of states of $\mathcal{M}$ satisfying $\mathcal{P}_{\bowtie p}(\nu, \varphi)$, we execute the algorithm for the aCSRL formula $\mathcal{P}_{\bowtie p}(\varphi)$ on $\mathcal{M}$ using the reward structure defined by ν. A similar approach can be taken with the other formulas $\mathcal{E}_J(\nu, \Phi)$, $\mathcal{E}^t_J(\nu, \Phi)$, and $\mathcal{C}^I_J(\nu, \Phi)$.